UNDERSTANDING COORDINATION IN DISTRIBUTED AGILE SOFTWARE DEVELOPMENT

A B M Nurul Afser Talukder

A thesis submitted to the Auckland University of Technology
in fulfilment of the requirements for the degree of
Doctor of Philosophy (PhD)

Supervisors
Mali Senapathi
Jim Buchan, Professor Stephen G. MacDonell

September 2022

School of Engineering, Computer and Mathematical Sciences

Intellectual Property Rights

In loving memory of

my Father, my Banyan tree,

Md. Anisur Rahman Talukder

Gone, But not forgotten!

Acknowledgements

"*It is easy to acknowledge, but almost impossible to realize for long, that we are mirrors whose brightness, if we are bright, is wholly derived from the sun that shines upon us.*"

~ C. S. Lewis

It is true that what I have achieved so far and am going to achieve in future are the endless blessings from my Creator, the Almighty Allah (Subhanahu wa ta'ala). He blessed me with the ability to learn and apply knowledge, and accept the truth that is around us. He blessed me with so many loving people in my life who are like the Sun of my life shining upon me. Thanks to the Almighty for being so kind to me and for all your blessings upon me.

Thanks to my loving parents who brought me up, always encouraged me and supported me to achieve my dreams. Even though it was difficult for them to bear the consequences at times, they never hold me back in chasing after my dreams. What I am today is only because of your sacrifice, love and support. Though my father will not be able to see me completing this longest and hardest journey of my life, I believe he would have been the proudest father to see me holding this book. Thank you, Abbu and Ammu, for being the brightest sun of my life.

To my precious and lovely wife Lisa, Thank you. Your love and patience sustains me, your encouragement never ceases. You are the one who first believed in me and encouraged me to start this PhD journey. During this roller-coaster journey, there were so many ups and downs, and we passed the hardest times of our partnership. Several times, I thought of quitting, but still, you never stopped believing that this could all be brought to completion if we just kept going. Above all, our shared faith in the Almighty helped all this make sense. I admire and respect you for the amount of sacrifice you have made for me and our relationship, especially, bearing all the pains to bring the best gift from Allah, our little Princess. We worked side-by-side for so many years and now here we are, almost at the end of this journey. Thank you

for being brave enough to leave it all behind and start a whole new journey together with me. I love you.

To my cute little princess, Manha; thank you. You are the most precious gift from the Almighty, and we are so proud of you. You have been a warrior from the first day of your life on this earth, and you fought and won so many battles during the last 3 years. You have shown us how to survive in difficult situations and beat all the odds. You are the smallest, but the brightest sun of our life who is continuously shining upon and making us happy. Mamma and Baba, love you so much.

To my primary supervisor Mali Senapathi; thank you. You graciously took me under your wing as a PhD candidate and supported me in every step of this journey. Your soft and kind words, while I was struggling, gave me hope and strength. Thank you for your wisdom, constant kindness, encouragement, support and guidance. I shall always treasure this time we worked together solving these “wicked problems” that lie at the heart of our shared research interests.

To my second supervisor Jim Buchan; thank you. You were always there with me to guide me by asking a lot of questions. Initially, I was quite nervous and afraid of meeting you as I had little knowledge to answer your simplest questions. But gradually I learnt that the more questions you ask, the quicker the problems are solved. You were always ready with a thoughtful timely word, spoken patiently, that cut to the heart of the research problems we were wrestling with. So often, your wisdom turned the problem right around so we could look at it from a whole new perspective. Though officially you are no longer acting as my supervisor, your contribution to this research and to my PhD journey is remarkable. Thank you so much for always being with me and for all so graciously sharing with me during this time.

To my third supervisor Professor Stephen MacDonell; thank you. You are the *treasure* of the supervision team. I truly admire the humbleness and the respect you give to your fellow researchers. It again proves that a truly knowledgeable person is always humble and down to the earth. You are not only a mentor for this research team but also for me and I will always try to be a person like you. Your unique perspectives

and thoughts constantly guided us throughout this journey and without your contributions, this work would not have reached this stage. Thank you for being with us and shining upon us in your most gracious way, especially, when it is needed most.

Finally, thanks to the rest of my family members, (especially my two brothers, my sister, and in-laws) and friends, who constantly believed me and supported me throughout my life and shining upon me to be a brother, a friend and, all in all, a good human being. Thank you.

Nurul Afser Talukder, September 2022

Abstract

Coordination in modern Software Development involves high levels of collaboration between multiple individuals having different capabilities, skills, and knowledge to achieve common goals. Coordination in this work environment requires harmonisation between tasks and deliverables, knowledge sharing among team members, and a shared understanding of how to work together. As a result, effective coordination in co-located software development projects is associated with several contemporary issues and challenges. These issues are even more complicated in geographically distributed contexts because several coordination mechanisms supporting co-located projects are either absent or inadequate. In addition to the challenges associated with spatial, temporal, and configurational differences, projects in distributed settings face increased coordination complexity. Having witnessed the success of agile methods in small co-located projects, organisations are striving to blend agile methods and distributed development to reap both benefits. Therefore, this thesis explores how coordination is managed in Distributed Agile Software Development projects.

Evidence was drawn from a multi-case study approach to develop a comprehensive coordination model with four key concepts (i.e. dependency type, dependency risk, coordination strategy, and coordination effectiveness), along with their constituting components and associations. The model explains that there are knowledge, activity, and resource type dependencies in distributed agile software development. These dependency types are influenced by the dependency relationship and situational factors, and may involve vulnerabilities that could lead to dependency risks. A well-formulated coordination strategy contributes to eliminating dependency risks and, thus coordination effectiveness.

It contributes to the body of knowledge in several ways: first, it presents an empirically validated coordination model in distributed agile software development. The model includes three essential concepts of coordination: a taxonomy of dependency, coordination strategy, and coordination effectiveness. Second, it introduces the

dependency risk concepts associated with the key dependencies that influence coordination effectiveness. This relationship also guides the formation of the coordination strategy in achieving coordination effectiveness in distributed agile software development. Finally, it prescribes a repertoire of mechanisms for the practitioners to improve coordination performance by alleviating dependency risks and coordination challenges.

The framework of analysis developed in this research can be adapted and applied in future coordination research in different contexts. The study's findings also have important implications for practitioners. It suggests strategies for coordination effectiveness by considering the influence of the dependency antecedents and associated risks. The coordination strategy concept can be used to select agile practices that ensure effective coordination in distributed contexts.

Publications from this research

Coordination in Distributed Agile Software Development: A Systematic Review	Talukder, A. B. M. N. A., Senapathi, M., & Buchan, J. (2017). Coordination in Distributed Agile Software Development: A Systematic Review. *28th Australasian Conference on Information Systems (ACIS 2017)*, 1–12.
Coordination in Distributed Agile Software Development: Insights from a COTS-based Case Study	Buchan, J., Talukder, A. B. M. N. A., & Senapathi, M. (2019). Coordination in Distributed Agile Software Development: Insights from a COTS-based Case Study. *Australasian Conference on Information Systems*, 942–952.

Contents

List of Tables

List of Figures

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person (except where explicitly defined in the acknowledgements), nor material which to a substantial extent has been submitted for the award of any other degree or diploma of a university or other institution of higher learning.

Signature of candidate

Acronyms

AEA	Agile Enterprise Architecture
ASD	Agile Software Development
AUTEC	Auckland University of Technology Ethics Committee
CE	Coordination Effectiveness
CI	Continuous Integration
CoP	Community of Practice
COTS	Commercial-Off-The-Shelf
CSCW	Computer Supported Collaborative Work
CT	Coordination Theory
CVS	Concurrent Versioning System
DASD	Distributed Agile Software Development
ERP	Enterprise Resource Planning
FDD	Feature Driven Development
GDASD	Globally Distributed Agile Software Development
GDSD	Globally Distributed Software Development
GSD	Global Software Development
GT	Grounded Theory
IS	Information System
IT	Information Technology
MTS	Multi-Team System
NDA	Non-Disclosure Agreement
OOP	Object Oriented Programming
PP	Pair Programming
QA	Quality Assurance
RQ	Research Question
SD	Software Development
SDLC	Software Development Life Cycle
SE	Software Engineering
SLR	Systematic Literature Review
SoS	Scrum-of-Scrums
SP	Sprint Planning
SPR	Sprint

STC	Socio-Technical Congruence
TDD	Test Driven Development
TMM	Team Mental Model
UAT	User Acceptance Testing
US	User Story
WIP	Work In Progress
XP	eXtreme Programming

Chapter 1 : Introduction

"*In order to be successful, you have got to have coordination*" – George Potter

The above statement designates the profound influence on the topic of this research as coordination is one of the many antecedents to software project success. Coordination in software development typically involves high levels of collaboration between multiple teams and their members and customers to achieve common goals. The complexity of modern software necessitates the involvement of many individuals with different capabilities, skills, and knowledge to understand the problem domain as well as design and deliver a software solution. The division of labour and integration of work to deliver a single coherent software product within a reasonable time box will, by its nature, involve the coordination of deliverables between development tasks, knowledge between teams and members, shared understanding of how to work together, and re-synchronisation of understanding as requirements change.

Well-coordinated development offers a number of benefits such as shorter time frames, and well-integrated, high-quality software that can be produced with less rework, and at a lower cost (Andres & Zmud, 2002a; Espinosa et al., 2007b). Conversely, poor coordination can lead to a number of development problems. For example, poorly coordinated dependencies between features worked by different teams could create misalignment of dependent works and integration problems that can lead to possible delays. Additionally, insufficient coordination between developers and testers in a team could result in testers being idle or overworked. Most contemporary software systems development involves teamwork, with individuals in teams and other teams collaborating with the clients to fulfil their requirements. Clients are often required to make decisions about requirements during software development and if this is poorly coordinated, misunderstandings or delays can result. Therefore, coordination in software development can be challenging because of such complex dependencies and

the potential high impact of poorly coordinated dependencies. This risk of coordination can be further exacerbated if the clients or teams or their members are geographically dispersed, because of fewer coordination opportunities at a distance.

Due to the potential access to multi-skilled workforces and lower development cost, Distributed Software Development (DSD) has become a common phenomenon in modern software development (Ågerfalk Pär J. et al., 2008). Moreover, by leveraging the time-zone effectiveness, distributed modularized software development and closer proximity to the market, organisations could potentially be more productive (Ågerfalk et al., 2009). However, organisations are frequently challenged by various dispersion factors such as geographic distance, time and cultural differences, and organisational, and functional boundaries that create significant barriers to the DSD activities (Jiménez et al., 2009; Nguyen-Duc et al., 2015). Coordinating the development work with distributed parties is more challenging because many of the mechanisms for dealing with the dependencies in co-located teams are either absent or inadequate (Talukder et al., 2017). With DSD, where many stakeholders cannot easily meet face-to-face and have fewer opportunities for communication, some practices may need to be adapted and some new practices adopted to effectively manage their dependencies. The exacerbation of coordination problems due to the significant differences between co-located and distributed development has been well recognised in the literature (Ågerfalk et al., 2009; Alyahya et al., 2022; Bendix & Pendleton, 2014; Espinosa & Boh, 2009; Espinosa & Pickering, 2006; Jiménez et al., 2009; Verner et al., 2014).

Agile Software Development (ASD) methods reflect the values articulated in the agile manifesto (Fowler & Highsmith, 2001). They place more emphasis on working software over comprehensive documentation, customer collaboration over contract negotiation, on people rather than on processes and tools, and embracing change over following a plan (Cockburn & Highsmith, 2001). Agile methods are characterised by short iterative and incremental development cycles, high levels of collaboration within a development team, and a focus on mechanisms to improve an organisation's ability to embrace and

respond quickly to changing requirements (Beck, 2000; G. Lee & Xia, 2010). In recent years, the use of ASD methods in organisations has made a significant contribution to improving software development practice (Dingsøyr et al., 2019). Part of this success and contribution may relate to the emphasis on coordination through rich communication channels and incorporating good coordination practices into the development process. For example, studies have reported on the successful incorporation of agile practices such as pair programming, refactoring, and automated builds, and artefacts such as information workspace (wallboard), product backlog, sprint backlog, and burn-down charts can effectively support coordination (Sharp & Robinson, 2008; Wagenaar et al., 2015).

Based on their success in co-located projects, it is therefore natural that agile methods are now being increasingly used in distributed environments (Hoda et al., 2018; Jalali & Wohlin, 2012). Software development organisations are striving to blend ASD methods such as Scrum and distributed development to reap the benefits of both (Hoda et al., 2018; Saxena et al., 2016). However, compared to each other, agile and distributed development approaches are based on significantly different tenets. For example, agile methods were proposed for small and collocated teams, whereas DSD incorporates stakeholders that are globally distributed; while agile methods mainly rely on informal processes to facilitate coordination, distributed development relies on formal mechanisms to coordinate tasks. These differences contribute to specific coordination issues and challenges in addition to the challenges associated with spatial, temporal, and configurational differences (Talukder et al., 2017). Some of these challenges include lack of shared understanding, delay in the management of project artefacts, conflicts due to misalignment of work, reduced trust, increased coordination overload in synchronising distributed effort, and increased coordination complexity due to miscommunication (Bannerman et al., 2011; Nguyen-Duc et al., 2015).

1.1 Research Problem & Motivation

The efforts to adopt ASD in distributed contexts are not as simple as in co-located situations. There are inherent challenges in the adoption of agile practices in DSD due to contextual factors. For example, because of the physical and temporal differences, some of the agile recommended practices, such as frequent and regular face-to-face meetings and continuous participation of clients or their proxies, are difficult to adopt in a distributed environment. While a number of studies report on the 'as-is' adoption of agile practices in distributed development (Bannerman et al., 2011; Bick et al., 2016; Hossain, 2008b; Hossain et al., 2009), there are studies that suggest tailoring several agile practices could improve practitioners' experience (Berntzen et al., 2022; Bick et al., 2017; Dingsøyr et al., 2019). Some of them suggest the adoption of a hybrid approach (mixing traditional and agile methods or including Community of Practices (CoPs) with agile), while others recommend tailoring the recommend agile practices for maximum benefit. However, evidence on the actual outcomes of such use has been limited and there are reports of findings of failure outcomes (Rolland et al., 2016).

While ASD has been adopted to optimize how a team delivers software, its use in scaled and distributed contexts is still the object of research, with some emphasis on planning and inter-team coordination (Moe & Dingsøyr, 2017). In particular, the extant literature provides little understanding or evidence about how agile practices contribute to effective coordination in distributed development.

The process of delivering software using more than one development team, often distributed, faces issues of dependencies, boundaries, coordination and/or synchronization. By identifying the main research challenges in DSD, Herbsleb (2007) highlights that the "*geographic distribution of a project seriously impairs critical coordination mechanisms*" (J. D. Herbsleb, 2007, p. 1). The challenges include making decisions, setting goals, communicating, building trust and managing the team (Owen, 2016). With the adoption of ASD, the process of coordination was rethought (Dingsøyr,

Bjørnson, et al., 2018) and require further empirical research on effective coordination strategies in the DSD context (Marthe Berntzen et al., 2021).

Therefore, it is timely to reinvestigate the important concept of coordination in the light of agile distributed software development. The purpose of this study is to understand the coordination in Distributed Agile Software Development (DASD) projects to (1) understand the key concepts related to coordination, (2) identify the means of coordination in DASD and (3) investigate how these means could support effective coordination. In order to achieve these goals, the study aims to explore how distributed software projects are coordinated when using an agile development approach and what are the mechanisms and strategies that contribute to coordination effectiveness.

1.2 Research Purpose & Research Questions

Despite the growing popularity of agile methods, there is a lack of useful descriptive, explanatory, or predictive theory about ASD, and its relationship with project coordination (Bick et al., 2016; Niederman et al., 2018; D. Strode, 2012). While most related research has focused on how coordination occurs in agile projects, research that contributes to developing theoretical and conceptual foundations of coordination in agile development has been limited (Dingsoyr et al., 2012; Stray et al., 2022). In an attempt to address this gap, Strode et al. (2012) investigated how software projects are coordinated when using a co-located agile development approach and the impact of coordination strategy on coordination effectiveness. While Strode's model provides a starting point for understanding coordination in co-located projects, recent studies have been conducted to investigate coordination in the DSD context (Sablis et al., 2020; Stray & Moe, 2020).

There are a number of recent studies that explored various aspects of coordination in Large-scale Agile Development (Henrik Vedal, Viktoria Stray, Marthe Berntzen, 2021; Morken, 2014a; Nyrud & Stray, 2017; Sekitoleko et al., 2014). Some of these studies inherently cover DASD and its coordination, but it is difficult to extract and use that knowledge in research and practice due to the lack to precise guidelines. Moreover,

our systematic literature review indicates an apparent gap in the current body of knowledge on *how coordination can be supported effectively in distributed agile projects*.

Therefore, the purpose of the research is to develop an in-depth understanding of coordination in DASD projects, based on empirical data, that would help both researchers and practitioners. The aim of this research can be framed in terms of one primary question, supported by four sub-questions:

RQ1: *How are the key dependencies coordinated effectively in DASD?*

To answer this overarching question, we will first explore the key dependencies that could be found in a DASD project by answering the following question:

RQ1.1 What are the key dependencies in DASD?

Once the key dependencies are identified, it is necessary to investigate the mechanisms used in the project to coordinate the dependencies, which would be answered by:

RQ1.2 How are they coordinated?

The effectiveness of these mechanisms could be measured by the '*coordination effectiveness*' model defined in the research literature on ASD (Strode et al. 2011). However, its relevance and applicability in DASD settings are yet to be evaluated. Therefore, this study would focus on evaluating and redefining the concept of coordination effectiveness in distributed agile development by answering the following question:

RQ 1.3 Is this coordination effective?

Finally, an in-depth understanding of the problems and challenges faced by practitioners is deemed to have a significant impact on coordination. This would be answered by the final research question:

RQ1.4 What are the challenges and factors associated with the coordination mechanisms?

1.3 Research Methodology

To fulfil the research aim, this work first focuses on identifying the key dependencies in the DASD projects (to answer RQ1.1). In the second phase, this work establishes the relationship between the techniques and tools adopted for coordination (to answer RQ1.2) and the components of coordination effectiveness i.e. which techniques and tools contribute to which components (to answer RQ1.3). While doing so, this research also identifies the challenges that can hinder the effectiveness of coordination and opportunities that emerge from technological evaluation to mitigate those challenges (to answer RQ1.4). Finally using those multi-phased outcomes, this work fulfils the aim of the research by interpreting the outcomes to answer the primary research question (i.e. RQ1) of this study.

This research uses a case study approach since it aligns with our aim of acquiring an in-depth understanding of the occurrence of certain phenomena in a real-world context (Yin, 2009). An exploratory multiple case-study approach is applied to produce the expected outcome underpinning the theoretical perspectives. This research adopts a combination of positivist and interpretivist epistemological approaches following the guidelines proposed by Lee(1991) to achieve a combination of positivist, interpretivist and subjective understandings.

The research activities follow the well-established case study research framework proposed by Eisenhardt (1989). These activities are performed in multiple phases following the case study protocol templates crafted from pre-existing study templates. The coordination process of DASD projects is the unit of analysis selected for this research. The two case organisations of this research were selected based on the open invitation of participation presented at a meeting of a professional network of Agile practitioners. The primary selection criteria were that each organisation was

undertaking agile software development projects where some team members were globally distributed and were involved in a complex software project.

Semi-structured interviews conducted at the organisations' premises are the primary data collection method used in this research. Additionally, participants' activity observations, project documents and artefacts are used to achieve data triangulation. The data are analysed using thematic analysis following the steps in Fereday & Muir-Cochrane (2006a). Additionally, a framework of analysis is developed based on the extant knowledge of coordination and preliminary data analysis findings.

During the data analysis, the dependencies are classified and grouped using the dependency taxonomy from Strode (2016). The coordination mechanisms for each dependency are inferred inductively from the data, following the approaches suggested by (Crowston, 1997; Malone & Crowston, 1994). Supporting technologies noted for these coordination activities are extracted from data snippets also, and duplications removed. The use of coordination techniques and supporting technologies are also observed during the fieldwork and serves to triangulate and deepen the understanding of the data from the interviews. Challenges associated with the dependencies are also identified from the interview data and coded and classified using thematic analysis involving the same open coding technique and then grouped into themes based on patterns. The quality of the research activities and findings are ensured to meet the validity and reliability constraints following the guidelines of (Dubé & Paré, 2003; Yin, 2018).

In sum, this research adopts a qualitative multi-case study approach to generate a conceptualisation of coordination that can analyse and explain the occurrences of the phenomena (i.e. coordination) and predict solutions for achieving effective coordination in DASD based upon empirical knowledge gathered from grounded data analysis and literature-based research.

1.4 Contributions of this study

"*Know how to solve every problem that has been solved*" – Richard Feynman

The above famous imperative of Feynman indicates that the first step of novel contributions starts by knowing what has already been done to solve a problem. A systematic literature review has been conducted on coordination in the DASD context. Conducting a systematic review is important as it provides a useful classification and structured overview of the current research. It can also be used to provide a valuable baseline for scoping new research efforts (Kitchenham et al., 2010). Our review shows that the majority of research has been on the application of general coordination theories to understanding various aspects of coordination in DASD, e.g. activities, roles, mechanisms and strategies (Talukder et al., 2017). However, there is relatively limited research on the effective implementation and management of these aspects in DASD. Some studies have contributed to gaining a better understanding of coordination mechanisms and their effectiveness in general; however, there is an apparent lack of specific guidelines on the selection and application of appropriate mechanisms and measuring and monitoring of their effectiveness in DASD. This research is posed to address some of these gaps by gaining a better understanding of the concepts of coordination and strategies for achieving effective coordination in DASD projects.

During the data analysis phase, this study identifies the need for a framework or model that could guide the analysis of dependencies, coordination activities and coordination effectiveness. This work develops a framework of analysis based on the theoretical and practical understanding of the researcher about coordination. This model is further refined while conducting the first round of data analysis for the first case. This framework is discussed in detail in the methodology chapter (Chapter 3) and the components are illustrated in Figure 3.6. This is a novel framework, and we seek to encourage researchers to adopt this framework in their data analysis process while conducting research on coordination in other contexts.

The key aim of this study is to enrich the understanding of coordination in DASD which is fulfilled by developing a theoretical model of coordination presented in Chapter 7. There are four main concepts in the model and those concepts, and their relationships are discussed and illustrated in Figure 7.12. This model presents a significant enhancement of some existing concepts and at the same time, presents new concepts that contribute to the knowledge of coordination in the DASD context.

Another motivation of this research is to provide guidance to practitioners on choosing appropriate coordination mechanisms and applying them to achieve coordination effectiveness in this context. As part of fulfilling this aim, this work presents recommendations for practice in Chapter 7. This work encourages practitioners to adopt these strategies that are believed to support them in mitigating dependency risks while improving their coordination effectiveness.

1.5 Thesis Structure

This chapter has introduced the concepts of coordination in DASD. The research problem has been discussed to set the ground for the motivation of this study. A summary of the research aims is presented with the research questions that have been answered. Finally, the methodology applied in this research is discussed and the contributions of this research are presented. The rest of the chapters contributing to this thesis are as follows and an overall structure of this thesis is summarised in a diagram in Figure 1.1, which elucidates how the different chapters are interrelated and build from each other.

Chapter 2 Literature Review

This chapter presents the outcomes of the review of the existing literature on coordination in general, followed by reviews of works on ASD methods and their application in DSD. The systematic literature review reflects on the concepts of coordination in DASD leading to the research gap addressed in this study. Finally, a preliminary conceptual model of coordination in DASD is presented that guides this research.

Chapter 3 Research Methodology

This chapter presents the research design by defining the methodology selection and material development process and the implementation of the design detailing the units of analysis and case and participant selection process. The data collection and data analysis sections are presented in separate sections detailing the process of collecting data from multiple sources and the methods applied to analyse them. The crafted framework of analysis is also presented here that is used in the data analysis. The final sections present the ethical considerations and quality aspects of this study.

Chapter 4 Case 1 Findings

This chapter introduces the project context for the first case and presents the analysis outcomes using the pre-established framework. An overview of the software development process of the project and the within-case data analysis process is briefed before presenting the findings. Finally, the key findings of this case are presented and discussed in detail.

Chapter 5 Case 2 Findings

This chapter presents the findings of the second case data that follows the same structure as the previous case.

Chapter 6 Comparison and Contrast between cases

This chapter presents the cross-case comparison outcomes of the cases. An overview of the cross-case analysis process is presented, and then the results are reported. The results present the outcomes from the comparison of the context, dependencies, coordination mechanisms and effectiveness and challenges in coordination. The emerging concepts from the findings are introduced under each section and further discussed in the next chapter.

Chapter 7 Conceptual Model of Coordination in DASD

This chapter presents the conceptual model of coordination in DASD grounded on the within- and cross-case analysis findings. This chapter outlines the concepts of

dependency types and their antecedents, dependency risks, coordination strategy and coordination effectiveness. The integral components of each concept are discussed by defining their associations, boundaries and states. The prescriptions for practical implementations in DASD projects are also presented, followed by a discussion about the contributions of this study.

Chapter 8 Discussion

This chapter presents the discussion of the overall findings starting with the answers to the research questions. A number of lessons learned from the case studies are presented with recommendations. An evaluation of this research methodology and interpretations is presented to justify the quality and rigor of this research.

Chapter 9 Conclusion

This final chapter of the thesis concludes the research by revisiting the research objective and achievements, followed by a discussion of contributions of this research in relation to IS and SE knowledge and practice. The limitations of this study are also discussed, and guidance is provided for future research opportunities.

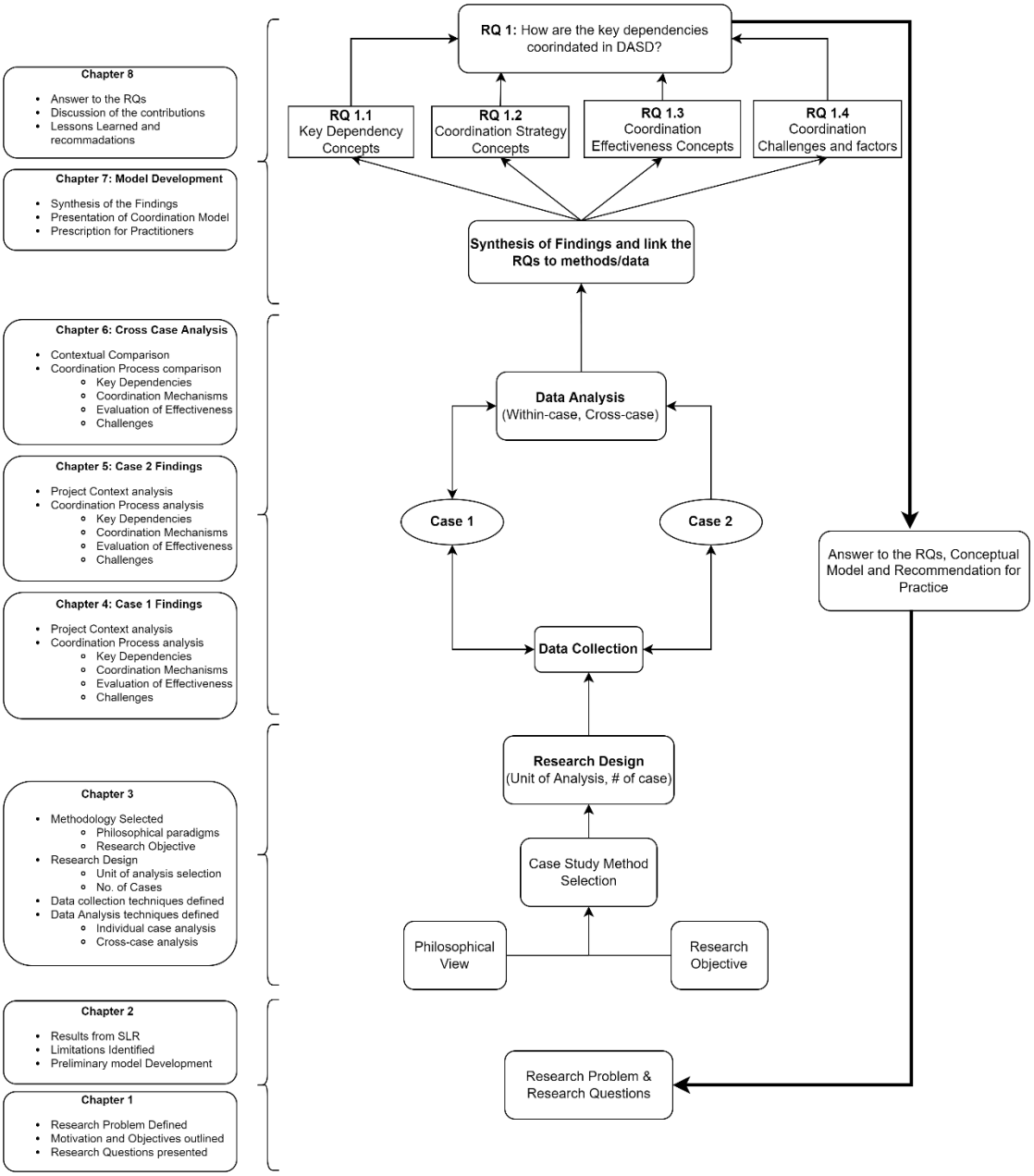

Figure 1.1: Overall Structure of this thesis

Chapter 2 : Literature Review

This literature review investigates the contemporary literature to explore the current state of knowledge relevant to coordination in Distributed Agile Software Development (DASD), particularly emphasising the struggles and strategies to effectively coordinate DASD projects. Part of this review was published at the 28th Australasian Conference on Information Systems (ACIS 2017) under the title *Coordination in Distributed Agile Software Development: A Systematic Review* (Talukder et al., 2017).

This chapter first discusses the backgrounds of the core concepts covering the three central tenets of this research: coordination, Agile Software Development (ASD) methods and Distributed Software Development (DSD). The chapter then presents the systematic literature review process and results to highlight the current state of knowledge regarding coordination in DASD.

The background study begins with an overview of traditional software development methods and discusses how different agile methods can benefit the software development process using their inherent practices. The review further investigates coordination studies and discusses supportive mechanisms to establish the foundation of this research from the non-agile perspective, then from the co-located agile perspective, and finally shifts towards the focus area of this research, i.e. the DASD context. Further, the review discourse on effective coordination in DASD which is the utmost focus of this research, includes a review of the most relevant coordination study in agile software projects (D. E. Strode et al., 2011) and DSD (Espinosa, Lerch, et al., 2002).

Before presenting the systematic literature review results, this chapter outlines the review process, including review protocol development, search string formation and conducting the search, screening and selection of relevant papers, and data extraction and classification scheme development. Then, the results are presented to highlight the

current state of knowledge in coordination. These results summarise the limitations and gaps in the contemporary literature to justify the motivation for positioning this empirical research.

2.1 Background of Core Concepts

The background of the study is essential to establish the context of the research. Therefore, this section elaborates on the core concepts of this study. The following two subsections provide a brief introduction to traditional software development and ASD methods. Coordination and coordination mechanisms in general software development are then introduced, followed by a discussion of the means of coordination in co-located and distributed agile software context. The coordination effectiveness concept is presented with a discussion of a well-known work in the co-located ASD project context. While discussing each concept, several domain-specific keywords and terms are also introduced that are being used throughout this thesis. While considering the DASD projects, any large-scale agile development project not covering the distributed aspects is not included in this study. Similarly, this research only considers studies covering non-scaled agile methods, ignoring the hierarchical complexities of coordination in a scaled-agile framework.

2.2.1 Traditional Software Development Methods

Several heavyweight development models were introduced in the early era of methodological software development approaches. These methods are termed heavyweight as their success mainly relies on extensive planning, detailed documentation and expansive design (Awad, 2005). This methodological approach consists of a fixed sequence of processes, such as requirements definition, designing, developing solutions, testing, and deployment. Each process is ordered strictly so that each step cannot be separated and must be performed in sequential order. For example, the testing process cannot be started until the whole solution development process is finished. Several traditional methods have gained attention, such as the Waterfall model, Spiral model, and Unified process. The essential features of these

methodologies emphasise a predictive design approach with comprehensive documentation, and the underlying approach is process and tool oriented.

In such a traditional plan-driven method, coordination is supported by administrative coordination where a particular person assigns tasks, allocates shared resources, schedules and regulates the integration of outputs (Faraj & Sproull, 2000). The success of coordination using these methods depends on hierarchical decision-making and well-structured documentation, which requires less communication. This particular feature is appropriate and beneficial for projects where requirements are reasonably predictable and stable. Hence it does not require frequent and ad hoc communication. However, an increase in size increases the complexity and cost of coordination at the administrative level, and as complexity grows, it is likely to originate administrative problems of coordination (van de Ven et al., 1976). Moreover, due to uncertainty, several unanticipated issues may arise that need to be supported by ad hoc communication. Using a traditional coordination approach to manage this volatile dependency is insufficient. Therefore, the formal method of coordination needs to be complemented by interpersonal coordination mechanisms (Cataldo & Herbsleb, 2013), which introduces the urge to adopt flexible software development methods, e.g. ASD method.

2.2.2 Agile Software Development Methodology

In the past decades, the software development industry has witnessed the rapid development of many software methods, tools, and techniques. Among all of them, the agile methodology has received remarkable attention and it has become a norm in the current software development domain. The key reasons for this attention include features such as speed-to-market support, shorter product development timelines with iterative and incremental life cycles to adopt new changing requirements, and prompt reaction to customer feedback (Boehm, 2002). In a traditional, rigid, pre-planned development process, accepting requirement changes and integrating customer feedback in an earlier stage are considered barriers to on-time and within-cost project

completion. The agile principles come with several practices to support volatile project conditions, making their application attractive in current software development projects. Table 2.1 illustrates the differences between agile and plan-driven (traditional) software development methods.

Table 2.1: Differences in Agile and Traditional SD Methods

	Agile methods	Traditional methods
Approach	Adaptive	Predictive
Success Management	Business Value	Conformation to Plan
Project Size	Small	Larger
Management Style	Decentralised	Autocratic
Perspective of Change	Change Adaptability	Change Sustainability
Culture	Leadership-collaboration	Command-Control
Documentation	Low	Heavy
Emphasis	People-Oriented	Process-Oriented
Cycles	Numerous	Limited
Domain	Unpredictable	Predictable
Upfront Planning	Minimal	Comprehensive
Return on Investment	Early in the project	End of project
Team Size	Small/Creative	Large

The Agile Manifesto (Fowler & Highsmith, 2001) articulates four core values that underline distinct concepts based on twelve principles. The first one emphasises software developers' empowerment and technical excellence and interactions. The second value focuses on continuous, informal and close customer collaboration throughout the development process. The third value designates prioritising working software over detailed planning or documentation. And the fourth value promotes continuous response to customer changes instead of upfront planning. Several evolutionary software development methods encompassing these core values have been introduced, such as Scrum, eXtreme Programming (XP), Kanban, Crystal, and Feature Driven Development.

Each agile method is aligned with the core values and principles of agile methodology and suggests a set of practices, roles, work products, techniques, and notations. Each

of them is different in some distinguishing factors among them. *Scrum* was introduced by Schwaber and Beedle (2002) for managing agile software projects and was mostly designed for primarily co-located teams. Scrum is an iterative, incremental method that is 'a *lightweight framework that helps people, teams and organisations generate value through adaptive solutions for complex problems*' (Schwaber & Sutherland, 2020, p.3). Scrum promotes collective teamwork, engaging people with the necessary skills and expertise, and their success relies on the competence in adopting the scrum values: *commitment*, *focus*, *openness*, *respect* and *courage*. Scrum prescribes practices that create opportunities to inspect and adapt to cope with uncertainties. For example, Scrum suggests iterative planning meetings such as sprint planning meetings for defining goals for an iteration (called a Sprint in Scrum), daily scrum meetings for updating the current status and blocking points, and sprint retrospective meetings for reviewing and adjusting the team actions.

The *eXtreme Programming (XP)* methodology is suitable for Object-Oriented programming that is built over five underlying values: communication, simplicity, feedback, courage, and respect (Beck, 2000). This method emphasises 12 primary technical practices satisfying agile values, such as pair programming, collective code ownership, informative workspace, ten-minute build, test-first programming (e.g. Test-Driven Development), and coding standards. Sharp and Robinson, from an ethnographic study on XP, state that rote implementation of the prescribed XP practices may fall short in fulfilling the core values. Instead, a holistic approach, intertwining multiple practices, is required to develop a shared understanding of purpose and responsibility in a rhythmic way that produces high-quality code (Sharp & Robinson, 2004).

Kanban is an adaptive method developed by Anderson (2010) which is gaining significant popularity as an agile method in the software industry. The reason is that the Kanban method is adopted based on ease of implementation, acute focus on working flow and waste minimization principles. For instance, the Kanban methodology can be

adopted by implementing three basic steps: visualise the process, limit work in progress and manage lead time. These steps can be implemented following practices such as visualising the workflow using the Kanban board, setting WIP limits, and tracking the lead times. A summary of the popular agile methods along with their distinguishing factors is reported in Table 2.2.

Table 2.2: Summary of the recommended practices of popular agile methods

Agile Methodology	Core Practices
Scrum	Product backlog, Sprint backlog for requirements elicitation Sprint planning meetings for defining goals Daily Scrum meeting for Status sharing Burndown chart to display progress Roles: Product Owner, Scrum Master, Development team
eXtreme Programming (XP)	Informative workspace The Planning game Metaphor Pair Programming Coding standards Collective code ownership Continuous Integration Rapid customer and developer feedback loops with on-site customer
Kanban	Simple visual design Limits the Work-In-Progress (WIP) Continuous delivery following just-in-time Waste minimization through prioritized feature development Kanban board for workflow visualization

In sum, agile methods support iterative and evolutionary practices that are said to better support software development through shorter development and release cycles to ensure customers' competitive advantage. Agile methods value interaction, collaboration, and adaptability by suggesting practices that not only better support SD life cycles but also enrich coordination between stakeholders. For example, in a survey study of over 500 developers, Maruping et al. (2009) found that collective ownership and coding standards practices (i.e. XP practices) mediate expertise coordination. In another study, Yu & Petter (2014) reported that specific agile practices like daily stand-up meetings and on-site customer can act as a coordination mechanism in software development.

Though the primary focus of agile methods is small and co-located teams, their success in this context motivates others to scale and adopt agile approaches in a distributed environment (Hoda et al., 2018). As this research aims to understand coordination in a distributed agile development context, knowledge of different agile methods and their core practices helps us to better understand their applicability in achieving effective coordination in this particular context.

2.2.3 Coordination and Coordination mechanisms in Software Development

Coordination has long been an interesting topic in organizational studies, economics, teamwork, Information Systems (IS) study, and particularly in software development projects (Kraut & Streeter, 1995; Malone & Crowston, 1994; Mintzberg, 1989). Different studies have proposed a particular definition of coordination underpinning varying theoretical perceptions. The most influential theory of coordination has been established by Malone and Crowston (1994) designated as Coordination Theory (CT). CT has pinpointed that it is important to organize the work approaches in any collaborative work and offered a taxonomy including vocabularies to characterize related activities. In addition, CT has posed the definition of coordination as the act of '*managing interdependencies between activities*' (Malone & Crowston, 1994, p. 90). There are also other definitions of coordination in the literature that present different perspectives. For example, Van de Ven et al. (1976) defined coordination as 'integrating or linking together different parts of an organization to accomplish a collective set of tasks'. In this research, we have used the definition of coordination proposed in CT since it is more suitable to explain and predict related dependencies and the mechanisms to manage those dependencies. The term dependency refers to the bi-directional relationship between the activities that have impacts on both sides and require mutual consciousness to accommodate the coordinating behaviour depending on the relationship; therefore can be called as *interdependency*. In this research, these two terms are used interchangeably.

CT identified three different types of dependencies that arise during teamwork: fit, flow and shared resource. Fit dependency arises when the same resource is being produced or used by multiple activities. Flow dependency arises when one activity depends on the resource produced by another activity. Shared resource dependency arises when multiple activities need to share the same resource. To resolve these three types of dependencies, proper management of the activities and resources is necessary. This management process involves participating people including their inherent specialized skills and knowledge (Faraj & Sproull, 2000; Kudaravalli et al., 2017), their interdependent activities, and the resources entailed (Crowston, 1994; Fuks et al., 2008).

In Software Development (SD), activities are carried out by multiple individuals and teams and formed into a single product by integrating them. While different people are working in different parts of the system, they have to agree to a common understanding of what they are developing, what is the current progress, what other members are doing, how each other's works are dependent and how their work fits in the final integrated product (Kraut & Streeter, 1995). Cataldo et al. (2008) propose a coordination process as a socio-technical system that involves two fundamental elements: one is technical, and the other is social. Technical properties include the software development processes, tasks, and technology employed, and social elements are constituted by the organization and individuals involved in the development activities. According to their framework, effective coordination requires congruence of this socio-technical system to ensure the successful integration of team members' efforts into a working product without redundant work, within time and budget constraints. Studies have reported that a higher level of coordination requirement makes the software development process challenging (Espinosa et al., 2007b; Kudaravalli et al., 2017) and requires suitable mechanisms to satisfy the coordination needs. For example, while working in a team-based structure, team members need to build a team knowledge about the task and the team. While working

on a joint task, team members implicitly gain expertise, as well as develop knowledge about the team and task, which further acts as a coordination mechanism. Socio-Technical Theory of Coordination acknowledges that the more suitable the mechanisms are, the more effectively the project will be coordinated (J. Herbsleb, 2016).

Coordinating software development teams requires effort and different activities or techniques are adopted for this purpose which can be termed as a “coordination mechanism”. In general, coordination mechanisms can be described as ‘*the activities carried out by team members when managing dependencies*’ (Espinosa, Lerch, et al., 2002). For example, when different developers are working on different components of the software, they can share their current progress while communicating with each other in group meetings or personal conversations, which can be entitled as mechanisms for sharing the progress of the dependent activities. Based on this shared understanding of the process, teams can reform their course of action to achieve the goal, if needed.

Data from several studies suggest a number of coordination mechanisms depending on patterns of prevailing dependencies. Espinosa et al. have segregated coordination mechanisms into three main types: *mechanistic*, *organic* and *cognitive* (Espinosa et al., 2007b). Mechanistic and Organic mechanisms are more explicit whereas cognitive mechanisms are implicit in nature. Mechanistic mechanisms support coordination by task programming (i.e. division of labour, schedules, plans, tools). For example, when development team members document the software or design requirements and share these with each other, they are mechanistically coordinating the product goals. This mechanism is effective when product requirements are certain and work processes are structured, i.e. suitable for traditional hierarchical software development environments. On the other hand, when tasks or requirements are uncertain, team members need to coordinate through continuous interaction and mutual adjustment, i.e. via organic mechanisms. Organic mechanisms support coordination by feedback or mutual

adjustment (i.e. formal or informal interactions) which is the key idea of agile methodology. On the other hand, Cognitive mechanisms consist of activities that are not consciously performed for coordination but help to build shared knowledge about tasks and about other members. These mechanisms are mostly useful when team members have less chance to interact with each other and need to adjust their workflow based on their anticipation of what needs to be done and when i.e. mostly suitable for asynchronous and distributed development environments. Studies, discussed in the following paragraphs, describe coordination from different research paradigms proposing mechanisms that can be categorised under any of these three mechanisms.

Most of the classical organizational research has focused on explicit mechanisms, either mechanistic or organic. One of the earliest works on this theme published by Thompson (Thompson, 1967) presented three particular types of mechanisms-coordination by standardization, planning and mutual adjustment corresponding to three types of dependencies consecutively: pooled, sequential and reciprocal. In a later study, Van De Ven et al. (1976) attempted to test the propositions of Thompson's work and presented three alternative mechanisms: *impersonal* (coordination by programming), *personal* (coordination via vertical or horizontal channels) and *group* (coordination by mutual adjustment). They also identified three fundamental determinants of modes of coordination such as task uncertainty, task interdependence, and size of the work unit. Another work by Mintzberg (1989), known as the work coordination model, has argued for three similar ways of coordination: Mutual adjustment, Direct supervision, and Standardization of work process, output, skills, and norms. Based on empirical evidence collected from 69 software teams, Faraj and Sproull (2000) documented that administrative coordination (similar to task programming mechanism) and expertise coordination (i.e. management of knowledge and skills of team members) can facilitate effective team performance. They separated team performance into two dimensions: efficiency and effectiveness, and their study

findings suggest that while administrative coordination strongly contributes to efficiency measures of team performance, expertise coordination has a significant role in explaining team performance. Overall, studies have reported that organic mechanisms tend to be more productive than mechanistic mechanisms while coordinating traditional software projects (Andres & Zmud, 2002b).

Cognitive mechanisms are reported in recent team cognition research and literature suggests that while working in a team, members build team knowledge through daily communication, task experience, and critical problem solving together. This knowledge helps individual team members develop a mental model of the team that helps them to coordinate effectively without frequent interactions (Klimoski & Mohammed, 1994; Mohammed et al., 2010). Cognitive mechanisms are termed as implicit as members are not employing these mechanisms consciously for coordinating, but this helps to have a shared cognition (Rico et al., 2008). Previous research has already established distinct conceptualisations of such implicit mechanisms as a Shared Mental Model (SMM) (Converse et al., 1993), collective mind (Weick & Roberts, 1993), Team Mental Model (TMM) (Mohammed et al., 2010), team situation awareness (Endsley, 1995), transactive memory (Wegner, 1987). While explaining the outcome of their field study, Espinosa et al. (2007b) generalized implicit mechanisms as a form of team knowledge and divided them into two general categories: shared knowledge and team awareness. In their work, the authors summarized that both the shared knowledge of the team and presence awareness can effectively mitigate some of the negative impacts of dispersion. Moreover, Scheerer et al. (2014) concluded from a multi-team perspective that cognitive mechanisms can influence the effectiveness of both mechanistic and organic coordination which makes it promising to apply in uncertain and challenging environments.

Overall, study findings indicate that a coordination strategy intertwining the mechanistic, organic and cognitive mechanisms can play a vital role in successful coordination. Several coordination studies conducted in different SD contexts present

evidence to indicate that one type of mechanism is not enough to support all forms of dependencies and adopting the right combination of mechanisms can help the team to attain effective coordination (Andres & Zmud, 2002b; Espinosa, Lerch, et al., 2002; D. Strode & Huff, 2015). For example, Espinosa et al. (2002) claim that an effective strategy for successful coordination in a distributed SD context would involve amalgamating coordination mechanisms that are well suited for the task. Additionally, the strategy might need to be changed with task progress over time and distance separation (i.e. co-located vs. distributed) which refers that the strategy for co-located teams might not be effective for distributed team coordination. For this reason, identifying an appropriate mix of mechanisms for any particular context at a particular time is an important requirement for effective coordination and one of the key goals of this research.

2.2.4 Coordination in Agile Software Development (ASD)

The key motivations underpinning the development of agile methodology include support for uncertain environments caused by rapid requirement changes, unstable markets, frequent customer feedback and time-to-market pressure (Schmidt, 2016). Due to its in-built support for uncertainty and other aspects of software development, the adoption of agile practices in software development projects has become a norm in recent years. As coordination is designated as one of the most influencing factors of software development success, a growing number of studies have focused on investigating the applicability of agile practices to support coordination.

To understand coordination in ASD projects, it is important to understand and identify the dependencies that are managed by agile practices (Viktoria Stray et al., 2022). The majority of the research on coordination applied and adopted coordination theory to understand the dependencies in different contexts. For example, in an empirical work with two co-located agile projects, Pikkarainen et al. (2008) used the interdisciplinary Coordination Theory (Malone & Crowston, 1994) while exploring the impacts of agile practice on communication. There had been little attention to the theoretical aspects of

coordination in agile development projects until Strode (2016) developed a dependency taxonomy from her research conducted on ASD projects. In her work, she proposed eight types of dependencies including requirements, expertise, historical, task allocation, activity, business process, entity, and technical dependencies. Since understanding the dependencies and choosing appropriate mechanisms to manage that dependency can lead to effective coordination, Viktoria et al. (2022) recommends a set of agile practices as coordination mechanisms that address multiple dependencies that can create a pathway for effective coordination in co-located ASD. Since then, this evolutionary dependency taxonomy has been used by coordination researchers in this context. Nonetheless, the applicability of this taxonomy in understanding the coordination of distributed agile projects is yet to be established.

Several studies have investigated the application of various agile practices to manage different types of dependencies in this context. In the earlier stage of agile adoption, Cao and Ramesh (2007) investigated the consistency of agile practices with the sound principles of organizational and management theories and proved a linkage between Van De Ven's (1976) coordination mechanisms and the mechanisms proposed by agile methods (e.g. customers involvement, short iterations) in small co-located projects. They also noted the effectiveness of each coordination mode presented in Van De Ven's work depends on task uncertainty, task interdependence and the size of the working group. The findings of this study refer that agile practices are aligned with management and organization theories creating an opportunity to analyse their applicability in different project contexts, e.g. distributed software projects.

While conducting a single-case study of a co-located scrum-based software project, Moe et al. (2010) have drawn on Dickinson and McIntyre's teamwork model (Dickinson & McIntyre, 1997) along with trust and Shared Mental Models (SMMs). In their study, the authors claimed that problems related to team orientation, and team leadership such as the high degree of individual autonomy, isomorphic team structure, lack of performance monitoring and continuous feedback, can act as barriers to achieving

team coordination. Their findings emphasise developing a shared understanding of teamwork and task increases the trust among team members that will eventually lead to better coordination. This work recommends that combining practices from multiple agile methods, such as Scrum & XP, would be supportive of efficient teamwork which in turn will improve team coordination.

Another study reported by Yu & Petter (2014) used the shared mental model to conceptually analyse three agile practices from XP and Scrum (i.e. system metaphor, stand-up meeting, and on-site customer) and found that these practices are useful for developing a shared understanding of the tasks, team process, and team interactions. For example, engaging the customer as part of the development process enables frequent interactions between the customer and the development team which can build a shared understanding of the overall requirements, which can reduce conflicts and increase the opportunity for managing changes successfully. This study depicts the fact that several agile practices are capable of developing a shared mental model which is effective for resolving coordination issues (Levesque et al., 2001). However, the effectiveness of these practices is only acknowledged in small co-located development teams, meaning they need to be verified for distributed teams.

In an earlier version of her work, Strode et al. (2012) proposed a theoretical model of coordination in co-located agile software projects which contains three parts: *coordination strategy*, *coordination effectiveness,* and *their relationships*. This work defined coordination strategy as a group of coordination mechanisms that manage dependencies in a particular situation and showed that a coordination strategy combining three types of coordination mechanisms: *synchronization*, *structure,* and *boundary spanning*, can enhance coordination effectiveness (see Figure 2.1).

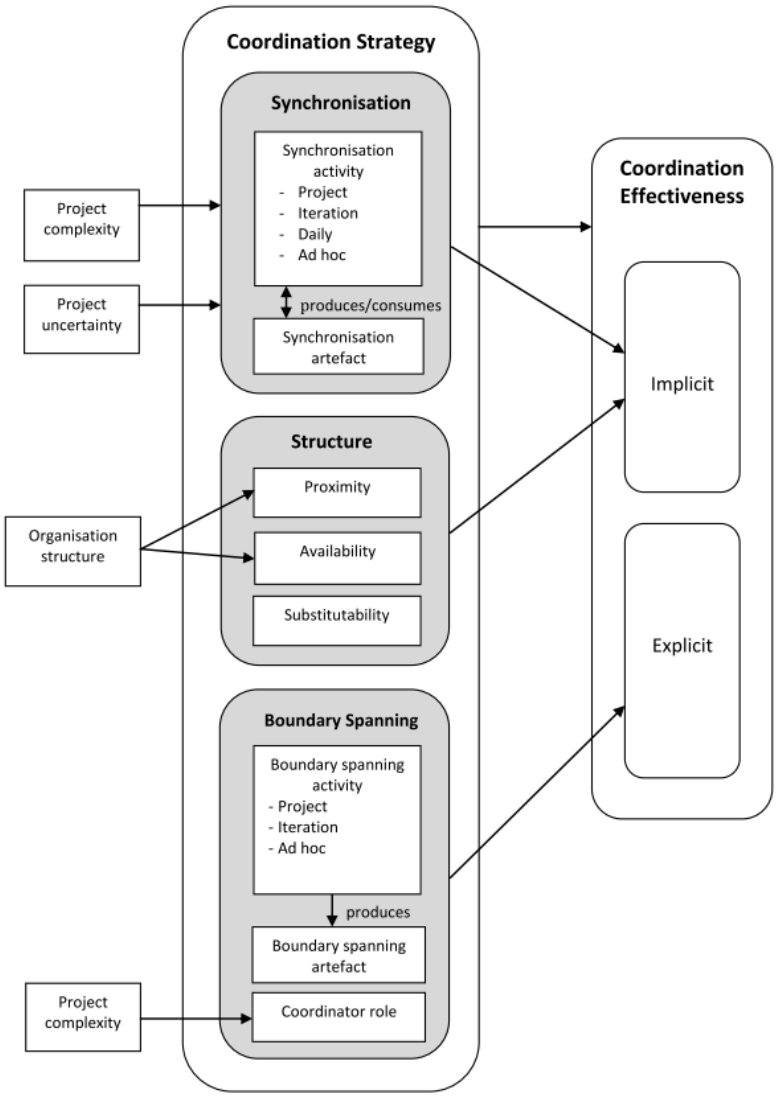

Figure 2.1: Strode's Coordination perspective in co-located ASD projects

The *synchronization* mechanism has two components: synchronization activity and synchronization artefact, the *structure* mechanism has three components: proximity, availability and substitutability, and finally, the *boundary spanning* mechanism has three components: boundary spanning activity, boundary spanning artefacts and coordinator role. The authors also presented a formal definition of coordination effectiveness containing two components: implicit and explicit, and highlighted that the synchronization and structure mechanism increases implicit coordination effectiveness, whereas boundary-spanning mechanisms increase explicit coordination effectiveness. The findings of this research are based on co-located agile teams and so might not be effective in the case of distributed agile teams. For example, as distributed team

members are separated by geographical and temporal distance, close proximity, availability, and substitutability mechanisms are unlikely to be as effective as co-located teams. Moreover, studies already reported that the boundary-spanning mechanisms are more critical in distributed teams than in co-located teams (Rolland et al., 2016). These assumptions are aligned with the limitations mentioned by the authors and suggested further research to validate the applicability in a distributed agile SD context.

Literature also suggests that the success of agile coordination mostly relies on implicit (Rolland et al., 2016) components in the form of shared understanding of tasks and team interactions and trust among stakeholders. Yu and Petter (2014) noted that agile team members can develop shared understanding through agile practices such as daily stand-up meetings and on-site customer participation. Similar results are also reported from an exploratory case study which states that daily stand-up, iteration planning, and iteration retrospective can facilitate building trust among team members (McHugh et al., 2011). Though the fundamental values of agile and scrum highly recommend self-organisation and self-management by feedback, coordination and backup, there is no proper guideline on how to achieve those values using agile practices (Moe et al., 2010), particularly in achieving effective coordination.

The success of agile coordination mostly relies on implicit mechanisms; however, the literature suggests several agile practices can act as explicit mechanisms that support effective coordination. While investigating the impacts and effectiveness of agile practices as coordination mechanisms, Xu and Cao (2006) observed that codified knowledge through artefacts and documents (e.g. user stories, iteration plans) play a crucial role in vertical coordination, whereas the sense of collective identity through pair programming and coding standards can influence horizontal coordination. Maruping et al. (2009) argued that a pre-established coding standard suggested by XP facilitates expertise coordination within software development teams. Strode et al. (2012) hypothesised from their exploratory case study that the boundary-spanning mechanism

consisting of boundary-spanning activities and artefacts and a coordinator role facilitates explicit mechanisms of coordination. For example, a designated person for coordinating with the external parties, e.g. customers and users, contributes to the coordination effectiveness in agile software projects.

Therefore, it has been evidenced that a repertoire of agile practices can act as a coordination mechanism, either implicit or explicit, to support one or multiple dependencies in ASD. Several studies in the literature focused on this area of interest from co-located agile context; however, their applicability and effectiveness in a distributed context are not evident yet.

Along with practices, the artefacts prescribed in different agile methods can also support coordination in software development. An artefact can be defined as "*a deliverable that is produced, modified, or used by a sequence of tasks that have value to a role*" (Fernández et al., 2010, p.11). While reporting results from an ethnographic study on eXtreme Programming (XP) practices, Sharp and Robinson (Sharp & Robinson, 2008) argued that artefacts of XP, such as story cards, design documents, and story walls can serve as information radiators that improve coordination. Another study on XP by Mackenzie & Monk (2004) also supports their argument. They identified that shared artefacts such as story cards, notice boards, software code, and screens can facilitate team members' shared understanding which is a vital component of coordination work (Aranda, 2010). In their theory of coordination in agile SD, Strode et al. (2012) identified synchronisation and boundary spanning artefacts as part of the coordination strategy that could contribute to coordination effectiveness. For example, in one of the projects under investigation, team members described user stories and design specifications shared as documents to understand project design that played the role of synchronisation artefact for them. These findings are also backed by another case study performed on three scrum-based projects where the authors noticed that in co-located agile projects production of artefacts such as product backlog, sprint backlog, backlog items, source code, and burn-down charts contributes to coordination

mechanisms such as synchronization and boundary spanning (Wagenaar et al., 2015). A recent article in an agile development project attempted to scrutinise the characteristics and informational value of the coordination artefacts used that can alleviate coordination challenges in different phases of development (Zaitsev et al., 2020). This article identified seven (7) different types of artefacts that are grouped under four (4) high-level categories depending on their purpose: foundational, projective, exposition and indicative. This article claimed that these artefacts have the potential to alleviate six different types of coordination challenges. However, additional research is required to generalise the application and effectiveness of these artefacts in general and, particularly, in distributed agile projects.

Agile software methods not only propose specific practices but also emphasise the role of several actors that are crucial for coordination. For example, Scrum suggests three specific roles: Scrum master, product owner, and development team. On the other hand, the eXtreme programming method suggests an XP coach role for project coordination purposes. Hoda et al. (2010) suggested a specific 'Coordinator role' for teams to be "self-organizing" which is an important attribute of the agile manifesto (Fowler & Highsmith, 2001). They stated that the coordinator role will act as a *'representative of the self-organizing Agile team to coordinate communication and change requests from customers'*. Consistent with this statement, Cao and Ramesh (2007) argued that project leaders in ASD take part in a significant role in facilitating coordination. Furthermore, D. E. Strode et al. (2012) highlighted the importance of the coordinator role for supporting boundary spanning activities and included it as a component in their coordination strategy model. All these studies and their findings signalled that considering specific roles, along with several practices and artefacts, is important for achieving effective coordination.

In summary, studies have summarised that several practices, artefacts and roles mentioned in agile methods, both individually or in combination, can act as coordination mechanisms, and hence, can facilitate effective coordination. These studies, both

explicitly and implicitly, indicated the feasibility of these mechanisms in DASD projects. Thus, it is imperative to further investigate and recognise potential strategies for effective coordination in DSD projects.

2.2.5 Coordination in Distributed Agile Software Development

Due to the current trend of globalisation and recognition of the benefits promoted by distributed development, software development in multisite, multicultural and globally distributed contexts has become commonplace in recent years. In particular, DSD may be characterised by stakeholders from different organizational and cultural backgrounds, with geographical, temporal and cultural dispersion collaborating through technology-mediated tools to gain DSD advantages (Hossain et al., 2009). The advantages include access to a larger pool of expert labour, cost-effectiveness, increased productivity using 'round-the-clock' development to reduce time-to-market pressure, improved product quality, and proximity to the market utilizing local resources (Nguyen-Duc et al., 2015; Saxena et al., 2016).

Contextually, DSD is different from co-located software development. In co-located software development, members are more likely to share a common organizational culture and structure that facilitates rich and frequent interactions, both formal and informal, which in turn promotes effective and efficient coordination. In this type of development setting, all members with the required technical skills and experience are available and accessible in one place, and they are more likely to have a common understanding and knowledge of each other's expertise, methods, and tools used for the project (J. D. Herbsleb, 2007). In distributed development, the aforementioned characteristics are threatened due to geographical, temporal and socio-cultural dispersion, which makes coordination challenging (Berntzen & Wong, 2021).

Ågerfalk et al. (2005) described geographical distance as a situation where team members are not co-located, but are rather geographically separated from each other, whereas socio-cultural distance arises when different people describe a particular situation differently based on their socio-cultural background, and temporal distance is

a situation when team members cannot communicate face-to-face. They argue that though geographical distance may imply increase socio-cultural and temporal distance, it is not the only primary reason for these distances. For example, both national culture and organizational culture may introduce socio-cultural distance and shifting work might be responsible for temporal distance. Each type of distance has effects on how team members coordinate their activities in this context and a better understanding of these impacts should help to choose appropriate mechanisms for improved coordination.

As already mentioned, the scope of this research is limited to exploring the impacts of geographical and temporal distance in distributed agile team coordination. The reason is that socio-cultural distance factors can arise due to the cultural differences of the team members' where they are from, whereas on the other hand, organizational culture involves the norms and values of the organization and the culture of the systems development (Carmel & Agarwal, 2001). Exploring the reasons and effects of these norms and values on distributed team coordination from both national and organizational perspectives will exaggerate the scope of this research. Though cultural and linguistic effects are out of the scope of the research, their influences on the overall coordination process cannot be ignored. The researcher accepts this issue and so has designed the study to minimise the impacts in the final outcome, e.g. choosing sites in a culturally similar location.

While introducing DSD will add some exciting opportunities, on the other hand, it can negatively affect team coordination activities. Due to dispersion factors, some mechanisms available in co-located teams are not available in distributed teams, for instance, co-located team members can communicate frequently, which is not possible in the case of distributed team members. Moreover, the effectiveness of some mechanisms, that are present regardless of dispersion, is affected by lack of co-presence. For example, distributed team members can communicate in real-time using video conferencing, but a good amount of contextual references are lost through this

technology (Espinosa et al., 2007b; Talukder et al., 2017). Ågerfalk et al. (2005) identified several negative impacts that can result from three types of dispersion such as coordination complexity, lack of practices for shared understanding, lack of awareness and reduced trust. Although mitigating these challenges is not a straightforward task, integrating agile practices in the DSD context makes the situation more complicated.

While reporting the dispersion impacts in distributed agile projects, Kahya & Seneler (2018) observed that difficulty in role definition, feedback delay and testing bottlenecks are the main difficulties in task coordination in this setting. Their findings are similar to that of (Smite & Dingsøyr, 2012) who identified that defined roles and responsibilities and a shared awareness of those authorities across sites are vital for distributed coordination. Besides, limited opportunity for face-to-face communication, lack of trust and team spirit are the other effects of temporal, geographical and socio-cultural distance that indirectly affect coordination.

Since the promotion of agile principles, agile methods have shifted the traditional software development practices by its evolutionary principles, and due to its continuous success in small, co-located teams, distributed software projects are striving to blend agile approaches to gain the benefits of both (Dingsøyr et al., 2017; Hoda et al., 2018). Based on a literature review study, Temitayo et al. (2017) reported that Scrum is the most popular agile method being adopted in DSD. A combination of Scrum and XP is the next most popular way of adopting agile in DSD projects. The authors reported several studies on the successful implementation of Scrum methods, such as daily scrum meetings involving distributed team members, to resolve some of the DSD challenges. They also identified that being the prominent agile method used in DSD, there is a clear knowledge gap in the literature investigating the fundamental challenges in adopting Scrum in DSD projects and strategies to address those challenges. This study also indicated the importance of studying the contextual factors and their impact on the effective use of Scrum practices in DSD projects. The key

researchers on this topic also iterated the same and suggested future studies in deepening the knowledge of coordination in this popular software development context (Moe et al., 2016).

There are significant differences in the coordination approaches of agile and distributed development due to their key tenets. For example, agile methods are best suited for small and co-located teams as it relies on informal communications for coordination, whereas distributed development demands more formal mechanisms for coordination (Ramesh et al., 2006a). In spite of the issues produced by combining agile and distributed development, studies have reported successful integration of agile in DSD (Altaf et al., 2019; Bose, 2008; Jain & Suman, 2016; Sureshchandra & Shrinivasavadhani, 2008; Young & Terashima, 2008).

For example, based on a literature review, Jain & Suman (2016) reported 19 benefits of using agile practices in DSD projects. This study revealed that 20 popular agile practices are reportedly effective in DSD among which 'Daily Scrum' and 'Sprint' are the most beneficial. Several of these benefits are related to coordination and communication, which are summarised in Table 2.3. These findings are followed-up by an industrial survey which indicated that agile practices are effective to enhance communication between sites that potentially resolve misunderstandings and communication issues.

A promising outcome of agile adoption in GSD projects has been reported in the literature. While investigating the usage of agile practices, Vallon et al. (2018) reported from an SLR study that the number of distributed agile teams has doubled since 2012 (i.e. 35% in 2012 and 82% in 2016). The qualitative analysis results of this study highlighted that an increasing number of agile adoption success stories are being reported in recent years, which include detailed descriptions of Scrum implementations and benefits in the GSD environment. This review study also indicated that Scrum practices need to be adapted for maximum benefits depending on the project and distribution context. For example, joint sprint planning and review sessions or an

altered number of sprint planning sessions per sprint could be beneficial for coordinating between multi-site development teams. Replacing the user story with *delivery story* is another popular modification suggested for complex distributed projects.

Table 2.3: Summary of benefits of Distributed Agile practices adapted from (Jain & Suman, 2016)

SL#	Benefits related to Coordination and Communication	Agile Practices (*)
1	Improved quality of communication between the sites	TDD, US, SPR, DS, SP
2	Enhance frequent communication between sites	PR, TDD, SPR, DS, SP, SR, RE, BAC, SOS
3	Reduce Communication Delay	SPR, DS
4	Focus on solving issues, misunderstandings, and impediments timely	CI, TDD, SPR, DS, SP, SR, RE, BAC, SOS
5	Reduce knowledge management difficulties	PP, TDD, DS
6	Induce more one-to-one communication between the sites	DS
7	Enhance visibility of project activities	SPR, DS, SP, SR, BAC, SOS
8	Improve stakeholder collaboration	PP, SPR, DS, SP, SOS
9	Improve trust among stakeholders	CI, SPR, DS, SR, BDC, SOS
10	Increase team awareness and cohesion	PP, SPR, DS, SP
11	Improved shared understanding among distributed developers	PP, TDD, DS, SP, SR, BAC
12	Increase motivation and engagement of team members	PP, CI, SPR, RE, SOS
* TDD- Test-Driven Development, US- User Story, SPR- Sprint, DS- Daily Scrum, SP- Sprint Planning, SR- Sprint Review, RE- Retrospective, BAC- Product and Spring Backlog, SOS- Scrum-of-Scrums, CI- Continuous Integration, BDC- Burndown chart		

Another stream of literature argues that the ability to mitigate some of the DSD challenges, corresponding to coordination, communication and cooperation, also intrigues the adoption of agile methods in this context (Ågerfalk & Fitzgerald, 2006; Bannerman et al., 2011; Paasivaara et al., 2012; Temitayo et al., 2017; Vallon et al., 2018). For example, in spite of fewer opportunities for face-to-face informal communication, several scrum practices such as daily scrum meetings, sprint iteration, sprint planning, and review meetings can help to develop both task and team awareness among the team members, implicitly supporting coordination. Several

studies also highlighted various agile methods that promote effective coordination by mitigating different types of coordination challenges. Bannerman et al. (2011) evaluated the effectiveness of seven scrum practices by their role in mitigating four coordination challenges: 1) increased coordination costs, 2) reduced informal contact to lack of critical task awareness, 3) inconsistent work practices across sites, and 4) reduced cooperation arising from misunderstanding, and reported that scrum practices play a distinctive role in mitigating both geographical and socio-cultural challenges. For example, scrum meetings such as time-adjusted daily scrum, scrum-of-scrums, sprint review and retrospective sessions, could alleviate spatial dispersion impacts by improving shared understanding and awareness of critical tasks. However, the scrum model fails to provide any advantage in reducing temporal dispersion challenges.

While reporting their journey, the majority of the studies also shared the problems and relevant practices to overcome and suggestions for future practitioners that are discussed in the following paragraphs. However, the adoption of agile practices in the distributed project is still a significant challenge reported in these studies (Booch, 2015; Jain & Suman, 2016; Vallon et al., 2018) that demands more empirical evidence for successful project coordination.

Most of the literature on DASD has identified communication as the primary feature that is challenged in adopting agile in DSD. The core principles of agile methods (Fowler & Highsmith, 2001) emphasise continuous communication and response to frequent changes in requirements, therefore team members need to communicate efficiently and effectively to coordinate themselves. Due to the inherent nature, the success of agile implementation heavily depends on informal face-to-face communication and coordination among team members, which is difficult to implement in a DSD environment; hence, a number of studies in the literature highlighted various types of communication challenges and their impacts on coordination.

The complexities in DASD communication are primarily caused by the physical and temporal dispersion that negatively impacts team members' availability. For example,

Cataldo et al. (2008) report that distributed developers face difficulties in identifying appropriate people to contact while coordinating interdependent activities. This finding is consistent with a later study in which Šablis & Šmite (2016) attempted to understand whether team members in a distributed agile context are isolated or connected, and highlighted that the poor availability of remote contacts is a perceived challenge in this development environment. In addition to physical and temporal dispersion, Ghani et al. (2019) identified two other types of dispersion: socio-cultural and knowledge, which negatively impact coordination. Socio-cultural dispersion refers to the difference in language, culture, ethics and work practices which may result in misunderstanding and personal conflicts. Socio-cultural distance could be present in co-located team members; however, the impacts are exaggerated by the physical distance between them due to the lack of informal communication opportunities. Knowledge dispersion is caused by differences in the level of knowledge or experience about the customer organisation. Nonetheless, the current body of literature has acknowledged fewer impacts of the latter types of dispersion than the former types.

Another important factor for effective communication is the 'Organisational culture', which is defined as '*the values, attitudes, and behaviour that represent an organisation's working environment, vision, and subjective*' (Hofstede et al., 2005). Organisational culture influences communication architecture since it controls the tools and technical compatibilities across the sites (Alzoubi & Gill, 2022). A bureaucratic and rigid organisational culture is a perceived challenge for communication effectiveness; whereas a flexible organisational culture promotes rapid communication by providing rich technological support, which enhances trust among the distributed members. Agile Enterprise Architecture (AEA) is a prominent strategy that could provide a blueprint for an entire organisation's operational environment for a flexible, responsive and lean communication structure (Alzoubi & Gill, 2020); this could potentially aid in mitigating DASD communication challenges.

Several studies have investigated communication challenges and suggested strategies for overcoming those challenges in DASD. Dorairaj et al. (2011), based on grounded theory research, assigned the causes of Lack of effective communication into four categories: *time zone difference*, *lack of communication tools*, *language barriers* and *lack of teamwork*. They suggested several strategies for effective communication, such as increasing overlapped working hours, creating formal and informal communication opportunities using various rich communication tools, building personal relationships and regular site visits for developing trust. In a systematic review study, Alzoubi et al. (2016) categorised six types of communication challenges perceived in DASD projects and suggested techniques to overcome those challenges. For example, team configuration issues are the most impacting on DASD coordination which can be reduced by encouraging face-to-face meetings at the beginning of the project, frequent product demo sessions and increased virtual meetings between the distributed teams. Since distributed team members spent 38% more time on such group meetings (Stray & Moe, 2020), it is important to define and ensure the effectiveness of these type of communication, which is yet to be discovered (Kostin & Strode, 2022).

Furthermore, while conducting the literature review on coordination in DASD, the researcher of this study and his team discovered that about half of the selected studies have mentioned several coordination challenges in their studies (Talukder et al., 2017). The results show that all the dispersion factors to some extent challenge the coordination process, and challenges related to coordination are interrelated because one reason can affect other aspects of coordination at the same time, such as communication challenges can create challenges in project management, team cohesion, and bonding. The researcher has explored whether all these reported challenges can be characterized using HOT (Human, Organization, Technical) perspectives that have been used by Hazzan and Dubinsky (2009). For a better understanding of the effects of dispersion in coordination, all the reported challenges are mapped using the HOT perspectives as shown in Table 2.4. To alleviate these

coordination challenges, appropriate coordination mechanisms need to be applied to effectively mitigate the coordination challenges.

Table 2.4: HOT classification of DASD Coordination challenges adapted from (Hazzan & Dubinsky, 2009)

		HOT Perspective		
		Human	**Organization**	**Technical**
Distance Factors	**Geographical**	- Unavailability of customer or proxy - effective remote conferences - misunderstanding complex requirements	- Team cohesion and bonding - Skill and experience Difference - organizational standards - Cost of coordination	- Increase documentation - Infrastructure difference - Lack of standard artefacts
	Temporal	- Meeting synchronization and planning - delay in Feedback	- Public holiday and vacation - Lack of standardization - Lack of control - Lack of project visibility	- Delay in artefact management
	Socio-technical	- Directness and Honesty - Notion of responsibilities - Language - lack of critical awareness - Reduced trust	- Traditions - Reduced Cooperation - Lack of shared understanding	- Inconsistent work practice - Task conflicts

More recent attention has focused on effective inter-team coordination in DASD projects. While this study notifies coordination, in general, as managing interdependent activities, inter-team coordination refers to the coordination of the dependent activities between teams. As multiple, interdependent software teams are involved in the development of the same software product, they form a team-of-teams structure that is closely similar to the Multi-Team System (MTS) architecture (Mathieu et al., 2002). A single team in the MTS structure has its proximal goals, and at the same time, collectively with other teams shares a common end goal. In DASD, these teams and their members are distributed across different geo-locations, therefore challenges of effective management of the dependencies between cross-site, distributed teams have been exaggerated.

While investigating inter-team coordination in a large-scale DSD setting, studies reported several types of challenges that had an impact on coordination performance. Bick et al. (2016) identified that a lack of proactive identification of interdependencies, possible priority conflicts, and inefficient centralized control are the main challenges of

inter-team coordination. A static and programmed way of coordination is not sufficient to identify and resolve all types of dependencies, rather a proactive dependency identification mechanism should be in place to avoid inter-team work priority conflicts due to last-minute ad-hoc coordination needs and work blockers. Collaborative planning sessions including planning workshops and cross-team testing and bug-solving under a dedicated product owner are effective mechanisms for proactive management of dependencies. Moreover, regular joint planning activities prior to each sprint incorporating distributed team representatives are effective to identify the dependencies and improve the dependency awareness that could be managed in subsequent inter-team coordination meetings. These findings are analogous to the top-down planning and bottom-up adjustment suggested by (Scheerer & Kude, 2014) which is suitable for the hybrid coordination approach where traditional and agile development practices are effectively adopted in coordination (Bick et al., 2017).

Fostering shared awareness across sites is another key challenge for inter-team coordination. There are five types of awareness that are vital to fostering cross-site coordination, these are: *who* (presence, identify, roles and responsibilities), *what* (action, intention, artifact), *where* (location), *how* (work procedure, artifact history), *when* (event notification) (Smite & Dingsøyr, 2012). Having a shared awareness of the presence and availability of remote site individuals, and being familiar with their identity, roles, and responsibilities support knowledge and expertise sharing across sites. Knowledge about what is going on in other teams including their work progress, and final goals (e.g. code changes, features) fosters awareness. Besides, knowing the location of the remote members such as local times, work hours and holidays, how they work, and when changes occurred could effectively support inter-team coordination. Improved accessibility to the different technical solutions and dedicated strategies are required to develop these types of shared awareness. However, it is important to shift the mindset, work habits and culture alongside the adoption of technological solutions.

It is a general belief that self-organising teams can control and adjust their communication to effectively manage their interdependencies (Hoda, 2011). However, it is challenging to control the communication structure while coordinating between distributed autonomous teams (Berntzen & Wong, 2021). The authors identified that distributed autonomous teams could perceive a high level of coordination when the received task interdependence is low. The received task interdependence represents the ad hoc needs for coordination caused by the workflow from other teams' jobs, whereas initiated task interdependence is analogous to the flow dependency mentioned in Coordination Theory (Malone & Crowston, 1994). This finding is consistent with a recent literature survey (Ekasari et al., 2022) which states that managing collaboration and coordination between distributed autonomous teams is the primary communication challenge reported in the literature. Thus, it is established that distributed autonomous teams face difficulty in coordinating ad-hoc coordination needs, which negatively impacts their coordination effectiveness; therefore there is a need to constantly adjust the team communication structure by establishing dynamic collaboration and coordination mechanisms.

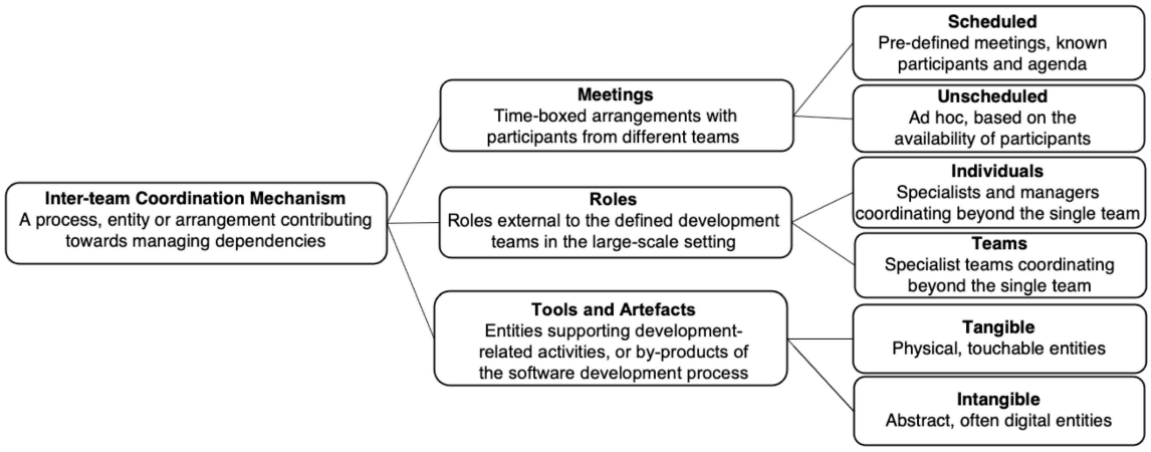

Figure 2.2: A proposed taxonomy of inter-team coordination mechanisms

A number of previous studies investigating inter-team coordination in large-scale distributed agile development have proposed several coordination mechanisms. The mechanisms suggested in these studies include scheduled and unscheduled meetings (Dingsøyr, Moe, et al., 2018; Moe et al., 2018), different communication tools such as

Slack and Jira, and artefacts such as a shared task board, backlogs, source code (Bass, 2016; Stray & Moe, 2020), designated boundary-spanner roles such as TAR, product owners (Gustavsson, 2017; Paasivaara & Lassenius, 2014). In continuation of this line of research, Berntzen et al. (2022) proposed a taxonomy of inter-team coordination mechanisms composed of three primary types of mechanisms: meetings, roles, tools and artefacts (as shown in Figure 2.2). Meetings could be either planned or unplanned and involve different types of participants depending on the purpose. For example, weekly demo sessions are regular inter-team meetings to showcase and demonstrate the work features, which facilitates team awareness (i.e. 'what is going on' and 'which team is doing what'). The coordination roles could be individual or team that are external to the development team and mainly coordinate with other teams and stakeholders. For example, a development manager is responsible for collaborating at the team leaders' level having knowledge of high-level team goals and inter-team work priorities. Finally, several coordination tools such as communication tools (e.g. email, slack), documentation management tools (e.g. Confluence, Jira) and artefacts such as Burndown chart, Release or product Roadmap, and Task boards collectively act as a coordination mechanism. This study also presented a framework to characterise these mechanisms based on technical, organisational, physical and social aspects that are believed to support the formation of an effective coordination strategy.

Another influential aspect of successful coordination is the formulation of coordination strategies that can lead to achieving effective coordination. Coordination strategy can be defined as a combination of coordination mechanisms that are consciously selected rather than occurring by chance and fit together to manage the dependencies in a situation (D. Strode, 2012). From an organisational perspective, Okhuysen and Bechky (2009) claimed that an effective coordination strategy could solve coordination issues that fulfil three integrating conditions: accountability, predictability and common understanding. However, this study could not provide any guidance to the process of fulfilling these conditions in achieving effective coordination. Espinosa et al. (Espinosa,

Lerch, et al., 2002) first conceptualized the coordination strategy for DSD using an integrated framework. Focusing on the DSD project context, they proposed that a mix of explicit and implicit coordination mechanisms can provide a better way of managing dependencies effectively. They also underscored that while both types of mechanisms are important to form an effective coordination strategy, explicit coordination mechanisms have a greater influence on implicit coordination in this context.

A later study by Li & Maedche (2012) adopted the coordination strategy concepts of Espinosa et al. (2002) and proposed a process of strategy formulation for DASD settings. They proposed that a combination of three main types of coordination mechanisms (i.e. mechanistic, organic and cognitive) could formulate an effective coordination strategy. In their preliminary understanding, they found that combining agile and plan-driven methods (i.e. hybrid approach) can serve as explicit mechanisms which are eventually influenced by or influence the implicit mechanisms and further lead to coordination effectiveness. Their preliminary results show that several situational factors such as task, team, technology, and organization have similar effects as in conventional GSD; however, several critical situational factors emerged that were not present in the literature. For example, customer requirements change has become an important factor that requires both mechanistic and organic coordination. Though several of these studies underlined the importance of coordination strategy formulation for different contexts and suggested several mechanisms (Espinosa, Lerch, et al., 2002; D. Strode, 2012; D. Strode & Huff, 2015), there is no formal conceptualisation of coordination strategy for DASD projects.

A growing body of literature proposed mechanisms for coordination in DASD drawing on different organizational theories. For example, from a multiple case-based study, Hossain et al. (2009) and Hole & Moe (2008) have drawn on Mintzberg's theory to identify coordination issues due to spatial, temporal and socio-cultural distribution and revealed positive impacts of agile practices to reduce coordination risks. From their studies, it appears that formal mechanisms, i.e. coordination by standardization and

direct supervision, are more common in traditional DSD. Whereas mutual adjustment and feedbacks are more applicable in DASD since agile methods mostly rely on people and creativity and emphasize informal communication. While reporting the outcomes, they have shown that agile practices such as daily stand-up meetings with the aid of a synchronous communication environment can help to build mutual adjustment as well as minimize project coordination overhead (Hossain et al., 2009).

Another longitudinal study applied work theory (Strauss, 1988) to analyse the success of Scrum in distributed project management activities (Pries-Heje & Pries-Heje, 2011). The articulation work theory explains that in order to pursue a common goal, all the dependent tasks performed by multiple actors within the project need to be articulated effectively. Alongside, an understanding should be established about who is doing what, when they do it, and when and how to coordinate their work. While working in a distributed environment, team members face issues related to articulating and establishing this understanding and four scrum practices (i.e. product backlog, sprint backlog, scrum board and daily scrum meeting) seem to be able to address these issues. For example, the tasks, their sequence, and their status are visible through sprint backlog and the scrum board that allows the team members to get an overview of the tasks and understand who is doing what, and further enables them to approach other team members directly to resolve any common issue. Additionally, scrum practices allow team members to develop a shared understanding of the development activities and realize the need for coordination (Bjørnson et al., 2018).

While some studies have advocated agile practices as coordination mechanisms, their effectiveness is also criticised while coordinating in the DSD context. Paasivaara & Lassenius (2014) based on a case study performed in Ericsson, have reported that Scrum-of-Scrum (SoS) meetings are not effective in coordinating distributed team members and often proved time-consuming and not productive. Another four-year case study involving 120 participants has also reported similar results using a paradigmatic assumption and stated that scrum-of-scrums are not an appropriate scaling mechanism

for distributed agile projects. Furthermore, agile methods do not promote formal communication in distributed projects. One study pointed out that when using agile methods in distributed development, a mismatch of adequate communication mechanisms sometimes hinders team coordination (Pikkarainen, M. et al., 2008).

Another stream of research suggests that adopting a hybrid approach, combining traditional plan-driven and contemporary agile development, can provide better coordination support in distributed development projects. Literature supporting this combined approach argues that hybrid approaches provide predictability, dependability, stability and effective use of resources (Moreno, 2013) that could create an ambidextrous strategy by balancing the flexibility-stability conflict (Saxena et al., 2016). In a theory-based approach, Barlow et al. (2011) presented a framework for a hybrid approach with recommendations for successful implementation in large organizations. They proposed several hybrid methods that include practices such as segmented upfront design, surrogate customer, flexible pair programming, and Short release cycles with a layered approach and recommended that organizations should apply these hybrid methods based on project size, volatility, and project interdependencies. However, these theoretical-based hybrid methods are actually proposed for large-scale organizations, and their applicability in DSD is not understood. Therefore, it would be a good approach to investigate which mechanisms are effective in what situations in this complex development environment.

Furthermore, due to the distributed arrangements, coordination activities in DASD are dependent on technology and computer-mediated tools. Procter et al. (2011) observed that coordination in this paradigm adopts different mechanisms for effective communication, such as Skype chat, Wiki, mailing lists, and Concurrent Versioning System (CVS). Many studies also reported that several collaboration tools are already been used in practice (Alyahya et al., 2011, 2013; Schuemmer & Lukosch, 2009). For example, while proposing a computer-based automated progress awareness tool, Alyahya et al. (2011) presented a review of several agile project management tools that

are popular to support agile practices, such as Rally, VersionOne, and XPlanner (*Rally Software®*, 2022; *VersionOne*, 2022; *XPlanner*, 2022). Recent coordination studies revealed that distributed team members heavily rely on informal communication tools, such as Slack for their ad hoc communication, which potentially reduces the need for a formal mode of communication (Henrik Vedal, Viktoria Stray, Marthe Berntzen, 2021; Stray & Moe, 2020).

Another line of research investigated the usefulness of existing commercial and non-commercial collaboration tools. Though there are already a number of tools available and being used to support various aspects of the coordination process, they lack support for effective coordination which probes innovative solutions. For supporting coordination in distributed Scrum teams, Feiner (2016) developed an open-source tool called Scrumpy which can support the coordination of requirements, time schedules, to-dos and code artefacts. In another solution proposal, Mak & Kruchten (2007) propose the NextMove tool to support asynchronous task prioritization and allocation for DASD projects that could resolve conflicts in inter-and intra-team task coordination. Nonetheless, their applicability and effectiveness to support DASD coordination are yet to be verified. Moreover, supporting tools and artefacts used in distributed agile projects have been less frequently discussed in the literature (Jain & Suman, 2016), which is particularly true in the context of coordination.

Coordination in the DASD setting possesses risk resulting from different aspects. Several studies discuss risks associated with the adopti(Damian & Moitra, 2006; Emam Hossain et al., 2009)(Damian & Moitra, 2006; Emam Hossain et al., 2009). Lack of coordination, communication delays from time zone differences, inefficient tool support, and lack of shared understanding of the dependencies are the key risks in agile adoption. Several contextual aspects also generate risks that potentially affect project outcomes. For instance, the coordination and communication risks are exaggerated by geographical and temporal dispersion; the higher the dispersion, the greater the risks (Ambler, 2012). Likewise, other project contexts such as project size, product maturity,

and task complexity could result in risks which in turn impact the coordination. Shrivastava & Rathod (2015) categorized the DASD risk factors into five categories: software development lifecycle, project management, group awareness, external stakeholder collaboration and technology setup. All these risk categories include factors that are directly associated with effective coordination. For instance, coordination, collaboration, communication and trust are key elements of group awareness and a lack of such awareness, potentially creates risks impacting those elements. Their study presented a mapping of agile practices to the risk factors that could potentially mitigate those risks. These risk factors are aligned with the risks reported by Emam Hossain et al. (2009). Based on a review of extant literature, they proposed a framework to present the key risks due to GSD project contextual factors; however, the impacts of these risks on the dependencies and their coordination are not highlighted.

In summary, all of the studies reviewed in this section confirm that DSD melding agile practices have gained significant attention in recent years. Until now most of the research in coordination in this setting has been focused on identifying challenges, proposing mechanisms, tools, and techniques by adopting existing agile practices to form strategies. While these studies have proposed solutions for coordination in DASD, they have also highlighted that practices and tools tend to have varying strengths in different areas of coordination support and little is known as to why they succeed in some cases and fail in others, which motivates a thorough investigation of the effectiveness of these coordination approaches. Moreover, several factors are also contributing risks and challenges to the coordination process that also need to be considered for effective coordination. This research attempts to explore these particular areas by answering the key research questions (as discussed in Chapter 1).

2.2.6 Coordination Effectiveness

'*Effective coordination means that the right actions are performed by the right agents at the right time and place*' (Heylighen, 2016, p. 8).

The above statement defines the characteristics of coordination effectiveness and is consistent with a preliminary definition proposed by Crowston et al. (2006). In that study, the authors denoted coordination effectiveness as the extent to which different dependencies are well managed. That means the more effectively the dependencies are managed, the more effective the coordination. Another study on coordination from an organisational perspective elaborates that three distinctive conditions, *accountability*, *predictability* and *common understanding* need to act in concert to achieve effective coordination (Okhuysen & Bechky, 2009). Accountability indicates that each team and their members would have a shared understanding of their individual responsibilities (i.e. what to do and when) that will contribute to the overall goal. Predictability reflects that teams will be able to predict potential dependencies and parties (i.e. who is dependent and for what) that need to be involved earlier so that they get enough lead time to communicate and coordinate with dependent teams without causing delays by blockers. Common understanding refers to the shared understanding of the collective and interim goals (i.e. know why) and understanding how their and others work fit together to accomplish the goals.

These three integrating conditions of coordination closely relate to each other. For instance, accountability promotes proactive behavior that initiates mutual relationship, shared knowledge of tasks and their progress which creates shared awareness of the tasks and actions of involved parties; thus making the dependencies predictable. Likewise, accountability generates common understanding about the tasks and goals that help to position each other's actions in a coordinated fashion. This study also identified five distinct categories of coordination mechanisms: plans and rules, objects and representations, defined roles, routines, proximity and familiarity, that if applied simultaneously and combinedly, could serve the above integral conditions of effective coordination.

In an empirical research study based on asynchronous and geographically distributed task context, Espinosa et al. (2002) designated coordination as the '*state of*

coordination' which depends on the extent to which the mechanisms effectively satisfy coordination needs; the more efficiently the dependencies are managed, the higher the '*state of coordination*' can be achieved. They argued that a mix of explicit and implicit coordination mechanisms might lead to achieving coordination success (i.e. higher state of coordination) in DSD (see Figure 2.3). While explicit coordination mechanisms emphasise overt activities, implicit coordination mechanisms focus on constructing a shared cognition of team members, their expertise, and availability, task allocation, and status which enables them to know what to do next. In the case of geographically DSD, shared mental models of activities and goals, task awareness (i.e. what is happing and when), knowledge of expertise location (i.e. who knows what), and presence awareness (i.e. who is around) can act as an aid in coordinating effectively. They have also underscored that explicit coordination mechanisms have a greater influence on implicit coordination. This emphasises the importance of combining both explicit and implicit mechanisms for effective coordination. The explicit and implicit mechanisms and their relation to coordination effectiveness presented in this prior study can be a prospective resource for taking one step forward by applying in Distributed Agile context.

Another stream of coordination research in an ASD context claims that formulating a coordination strategy integrating multiple mechanisms is helpful for achieving coordination effectiveness. Strode et al. (2011) proposed a conceptual framework of coordination in a co-located ASD context. While proposing a ubiquitous theory of coordination, they proposed a unified definition of coordination effectiveness based upon their case study research and exploiting the work of Espinosa et al. (2002). In their study, Strode et al. urged for two essential considerations while conducting any coordination research irrespective of contexts (e.g. co-located, distributed) and methodologies (e.g. agile, waterfall). One is to "*find out what activities and artefacts in a situation support coordinated action*", and another is to "*identify the features and characteristics of a highly coordinated state"* – i.e., what such a state "looks like".

According to their research, for achieving coordination effectiveness (i.e. a highly coordinated state) we should have a better understanding of what are the features and characteristics of coordination effectiveness and what coordination mechanisms can support achieving that state of coordination. Finally, they proposed a definition of coordination effectiveness that is applicable for co-located agile development projects which further needs to be evaluated in Distributed Agile context. Since this definition closely aligns with this particular research aim, we have discussed the model and its component's in the following subsection.

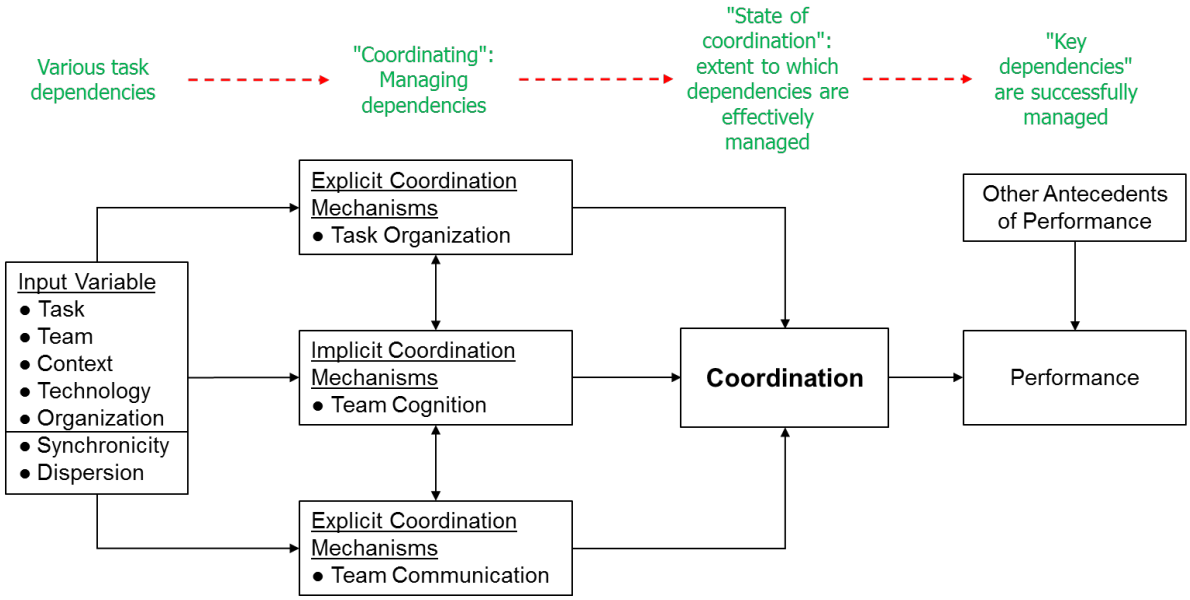

Figure 2.3: Espinosa et al.'s (2002) *integrated Framework for Coordination in DSD*

Later, Darnell (Darnell, 2015), in his quantitative study, attempted to evaluate the components of coordination effectiveness proposed by Strode et al. (2011). Darnell's aim was to predict the success of ASD projects. Based on the analysis of non-experimental, cross-sectional survey results, it is evident that both implicit and explicit coordination mechanisms are important for co-located agile project success. However, explicit mechanisms have shown a greater impact on agile project success. As the effectiveness of explicit mechanisms is severely reduced in distributed contexts (Yadav et al., 2007), the validity of these findings warrants further research.

Further, Li & Maedche (2012) posed an initial framework of coordination strategies in DASD. Their findings are based on theoretical examination and literature review of DASD studies. The framework combined organic, mechanistic and cognitive mechanisms to form a coordination strategy that can lead to coordination effectiveness. Their framework also articulates the influence of several situational factors (i.e. task, team, technology, and organization) on the effectiveness of distributed agile team coordination.

While exploring coordination in agile global software development, Scheerer & Kude (2014) presented a conceptual framework of coordination strategies. This framework adopted Espinosa et al.'s (2002) prior work to define coordination effectiveness. Their research findings present different archetypes of mechanistic, organic and cognitive mechanisms to form a coordination strategy. However, the findings are based on conceptual insights that need to be evaluated in a real-life context.

The extant literature has made several attempts to shed light on coordination effectiveness and its importance on project and team success. A summary of the current literature on coordination effectiveness, mentioning their context and primary focus, is shown in Table 2.5. The underlying aim of this research is to specifically focus on realizing the process of achieving effective coordination in the DASD context. In doing so, the researcher will start by analysing of most relevant literature i.e. the work of Espinosa et al. and Strode et al.'s work (Espinosa, Lerch, et al., 2002; D. E. Strode et al., 2011), which will be used as a baseline for this research to gain an in-depth understanding of this concept.

Table 2.5 Summary of Literature on Coordination Effectiveness:

Authors	Year	Context	Research Focus
Xu and Cao	2006	ASD	Coordination mechanisms and their impact on the effectiveness
Strode et al.	2011	ASD	Definition of Coordination Effectiveness
Strode et al.	2012	ASD	Theory of coordination
Darnell	2015	ASD	Evaluating Strode's coordination effectiveness model to project success

Espinosa et al.	2002	DSD	Coordination mechanisms in DSD
Espinosa et al.	2007	DSD	Effect of team cognition on coordination effectiveness
Li & Maedche	2012	DASD	Formulation of Coordination strategy for Coordination Effectiveness
Scheerer et al.	2014	DASD	The interplay between Coordination Strategy and inter-team coordination effectiveness
Stavrou et al.	2014	DASD	Effects of structural incongruence on coordination effectiveness in MTS

* DSD= Distributed SD, ASD= Agile Software Development, DASD= Distributed Agile Software Development

2.2.6.1 Strode's Model of Coordination Effectiveness

The most recent and relevant study to define coordination effectiveness has been reported by Strode et al. (2011), presenting a theoretical concept of coordination effectiveness in an agile software project context (see Figure 2.4). As an outcome, a definition of agile project coordination effectiveness has been developed that states:

"*Coordination effectiveness is a state of coordination wherein 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 individual's work fits in with other team members work. In addition, every object (thing or resource) needed to meet a project goal is in the correct place or location at the correct time and in a state of readiness for use from the perspective of each individual involved in the project*".

According to the definition, coordination effectiveness consists of an explicit component, i.e. the '*right thing*' is in the '*right place*' at the '*right time*', and an implicit component, i.e. know why, know what is going on and when, know what to do and when, know who is doing what, and know who knows what. This concept of coordination effectiveness has been derived from a broader work by Strode (2012) where a theory of coordination in ASD has been presented. In that work, the author stated that coordination effectiveness is an outcome of a particular coordination strategy (i.e. synchronization, structure, and boundary spanning) and the coordination strategy varies depending on the customer association in the project. If the customer or their representative is present, coordination strategy including synchronization and

structural coordination mechanisms will lead to a higher state of coordination. On the other hand, if the customer or their representative is absent, additional boundary-spanning mechanisms are needed along with synchronization and structural mechanisms (Espinosa et al., 2007b).

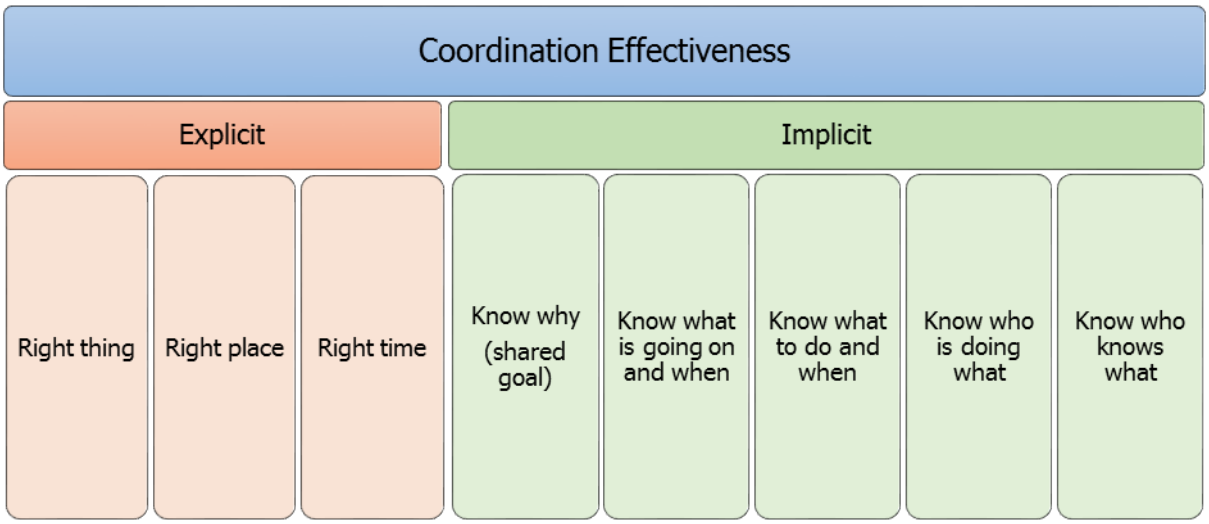

Figure 2.4: Strode's Coordination Effectiveness Model

Drawing upon the above two strands of research, Strode et al.'s works provide a pragmatic theory of coordination along with a clear definition of coordination effectiveness and its relationships in an ASD context. Moreover, while defining coordination effectiveness components, the present research adapted Espinosa et al.'s (2002) work, which focused on DSD projects. Therefore, it is more likely to be applicable in a distributed agile context. However, several contradictory characteristics of the distributed context require a review of this definition to generalise for distributed agile settings.

Firstly, some of the coordination mechanisms of the suggested strategy are not available in DASD settings. For example, in the case of structures, achieving proximity, availability, and substitutability of remote members is not feasible in distributed settings, and synchronization activities such as performing daily stand-up meetings from all locations at the same time or informal interaction via face-to-face conversation are not possible due to geographical and temporal separation. Another effect of

distribution is that, though some of the mechanisms are still present in both co-located and distributed settings, the effectiveness of those mechanisms is affected due to a lack of co-presence. For example, distributed team members can use video conferencing for communicating in real-time, but several physical and contextual references are missing in this virtual communication (Espinosa et al., 2007b). Since coordination via team cognition mechanisms in co-located and distributed members differs significantly (Espinosa et al., 2007b), implicit components of coordination effectiveness might need to be re-assessed and adjusted accordingly. For example, *presence awareness* or *knowing who is available* can promote effective coordination (Steinfield et al., 1999), which is not considered in Strode et al.'s model.

The overall understanding drawn from the above analysis shows that coordination effectiveness in ASD requires both explicit and implicit components, and to secure a highly coordinated state, an appropriate coordination strategy should be employed to satisfy both components. As per the discussion, this conceptualization is worth applying in the DASD context and finding new insights that will help to extend this concept complementing DASD project characteristics. Moreover, there is a lack of comprehensive understanding of the risks associated with different dependency types and their impacts on coordination effectiveness in DASD settings (Bick et al., 2017).

2.2 Systematic Literature Review

This systematic review is a part of the groundwork for this research and the goal was to get a useful classification and structured overview of the current research on coordination in DASD. It is also a valuable baseline to assist new research efforts (Kitchenham & Charters, 2007a) which was another purpose of this review. This review process follows the established guidelines and procedures proposed in the literature (Dybå et al., 2007; Kitchenham & Charters, 2007; Okoli & Schabram, 2010). The review process consists of the following steps: establish review protocol, conduct search, screening and selection of papers, and data extraction and classification.

2.2.1 Establish literature review protocol

A systematic literature review is described as a "*systematic, explicit and reproducible method for identifying, evaluating and synthesizing the existing body of completed and recorded work produced by researchers, scholars and practitioners*" (Fink, 2019, p. 3). Therefore, it should follow a clearly defined protocol that will make this process rigorous and able to be replicated by other researchers. A clearly defined review protocol should have a specific rationale and intention, literature searching and selection criteria, and an established data extraction and classification scheme.

In this research, the motivation is to identify the state-of-the-art of coordination research in DASD. The main objectives of this review are to (i) establish the body of knowledge by identifying and classifying the available research on coordination in DASD, and (ii) identify the main types of research published.

2.2.2 Searching Relevant Studies

The search strategy applied in this review included electronic databases and manual searches of conference proceedings. The search process was conducted in two main phases. In the first phase i.e. the preparation phase, we used a basic search string to identify an initial set of studies referred to as primary studies. These primary studies were the basis for refining the search string that would provide the same set of studies in all the databases. Finally, the keywords for the search were finalised based on the results received from Google Scholar and Scopus databases.

In the second phase, the final search string was applied to four comprehensive and widely used databases: *Scopus*, *ACM Digital Library*, *IEEE Xplore*, and *AIS Electronic Library* (*AISeL*). In addition, all volumes of the XP and Agile Development Conference proceedings were searched manually. Search strings were formulated by combining three key concepts using the 'AND' operator. Keywords in each key concept were combined using the 'OR' operator to ensure good coverage of papers related to that particular context area. The final search string, illustrated in Table 2.6, was adopted to meet the specific idiosyncrasies of each database search form.

Table 2.6: Search terms used in this study

	'A' AND 'B' AND 'C'
A	Agile OR scrum OR XP OR extreme programming OR lean OR Kanban
B	Global* OR distributed OR disperse* OR "large-scale" OR "large scale"
C	coordination OR "co-ordination"

Agile methods were limited to the most commonly used methods in practice, i.e. agile, scrum, extreme programming, lean and Kanban, and all types of distributed contexts were included (e.g. large-scale, global, and dispersed development). Any previous review studies, either a systematic literature review or mapping studies (Petersen et al., 2015) were also considered.

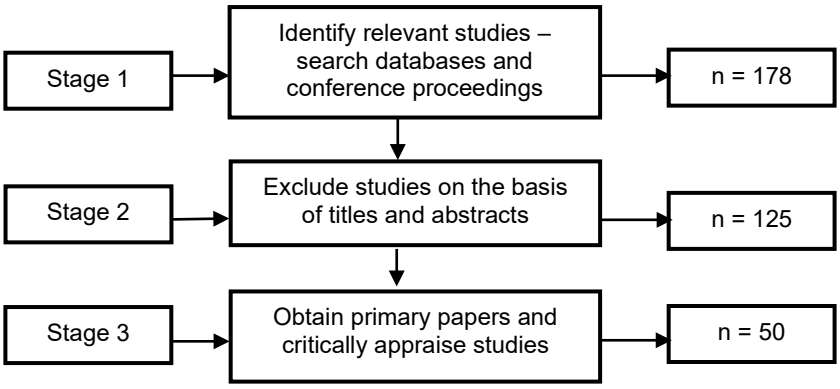

Figure 2.5: Stages of the study selection process

2.2.3 Screening and Selection Procedure

The search string applied on the four databases (i.e. ACM Digital Library, Scopus, IEEE Xplore, and AISeL) retrieved 178 papers. The summary of these screening results is illustrated in Table 2.7. Two researchers (the candidate and one supervisor) independently analysed the 178 publications and applied the inclusion and exclusion criteria. This included excluding studies based on titles and abstracts, and removing duplicates, non-English publications, and non-peer-reviewed scientific papers. This process resulted in 53 papers being excluded and 125 studies included. Next, three researchers (the candidate and two supervisors) analysed the 125 papers over a two-week period. Both during and at the end of this period, meetings were held to conduct in-depth reviews of each paper. The outcome of this produced 50 papers that were

assessed for quality using the criteria proposed by (Dybå & Dingsøyr, 2008) (see Figure 2.5).

Table 2.7: Summary of screening results

Database	Search Results	Relevant Paper
SCOPUS	323	72
IEEE	114	37
ACM	508	21
AISel	1061	48

Studies were eligible for inclusion in the review if they clearly addressed specific aspects of coordination in DASD and presented empirical data. Studies published between 2006 and 2016 were included. Only studies written in English were included. Studies were excluded if their main focus was not coordination in DASD. For example, studies that focused on large-scale agile development, but not in a distributed context were excluded. In cases where studies were published in more than one outlet, then the most recent version was included. For example, if there were two studies, one conference and a later journal publication, then the conference publication was excluded. Non-peer-reviewed scientific works (e.g. books, book chapters, experience reports) were excluded.

2.2.4 Data Extraction and Classification

The derivation of the classification scheme was based on Petersen's classification guidelines (Petersen et al., 2015). First, the abstracts were reviewed to identify the main concepts and themes related to coordination in DASD using the `keywording' process described in the guideline. Second, the set of keywords collected from different papers was combined together to develop a high-level understanding of the theme, context, and contribution of the research in order to form three classification categories: Theoretical Foundation and Application, Techniques and Tools, and Challenges (shown in Table 2.8). All the extracted data were mapped into these categories. To minimise the threat of researcher bias, the extraction and classification process was reviewed by the two supervisors.

2.2.5 Threats to validity

Construct validity relates to the alignment of what is investigated to what the researchers had in mind, as well as the completeness of the results. To reduce this threat, a data collection process was designed in advance that defined the research questions and the search strategy. Search terms were clearly defined that would enable us to identify the relevant literature. In addition, we used synonyms of the main terms to capture variations that may appear in the literature. To address the second aspect of construct validity (i.e. completeness of results), ensuring we found all the papers on the topic area of interest, we undertook the search in publication databases that are reputed to be well indexed. *Reliability* (repeatable with the same results) of the data collection was ensured by defining our search terms and procedures in sufficient detail so they can be replicated by other researchers. The inclusion/exclusion criteria are also described to a level of detail that can be replicated by others and have little room for misinterpretation. The categorization is a possible threat to validity and although the procedure is described, it is not certain that others would come up with the same classification schemes. To address this, the classifications were conducted by the first author and validated by the second and third authors. *Internal reliability* is a low threat in this study since only descriptive statistics were used in the analysis of the data. *External validity* is not in question for this study since we do not try to generalise our results to other review studies.

2.2.6 Results Summary

Our literature review showed that the majority of research (40%) had focused on the application of general coordination theories to understanding various aspects of coordination in DASD, e.g., activities, roles, mechanisms, and strategies. For example, CT is most commonly used as a theoretical lens to gain a better understanding of the process of formulating coordination strategy (Bick et al., 2014; Xu, 2009) and to understand the role of communication in supporting coordination (Li & Maedche, 2012;

Modi et al., 2013). Another theoretical framework of coordination proposed by Mintzberg consisting of three main mechanisms (i.e. direct supervision, standardization and mutual adjustment) has been applied to understand the relevance of agile organisational context (Hossain, 2008a; Hossain et al., 2009; Morken, 2014b).

Table 2.8: Classification Scheme developed from data

Classification Scheme	Key Reported Aspects
Theoretical Foundation and Application	Theories related to coordination in DASD: application of well-established theories to conceptualize activities, dependencies, dimensions, and effectiveness and coordination roles.
Techniques And Tools	Coordination practices and mechanisms, strategies, artefacts, conceptual frameworks and tools for coordination support
Challenges	Coordination challenges related to DASD

The second line of research that gained popularity in this field investigated different types of dependencies and their effective management. As Malone and Crowston stated, "*If coordination is defined as managing dependencies, then further progress should be possible by characterizing different kinds of dependencies and identifying the coordination processes that can be used to manage them*" (Malone & Crowston, 1994, p. 91).

While understanding coordination from various aspects, studies have investigated different types of coordination needs and mechanisms to manage them effectively as discussed in the previous sections. However, the review revealed that there is relatively limited research on the effective implementation and management of these aspects in DASD. For example, while a few papers highlight the significance of specific roles for effective coordination in DASD, there is no specification of the functions and obligations associated with these roles. Some studies have contributed to gaining a better understanding of coordination effectiveness and strategy formulation in general; however, there is an apparent lack of specific guidelines on the strategy formulation

process and measuring and monitoring coordination effectiveness in DASD. This presents a promising future area for more research to understand coordination in this dominant context.

The main type of research has been evaluation research, followed by solution proposals and philosophical papers. While there is some research on proposing and developing solutions to support coordination, there is a relative lack of research on their evaluation in real distributed agile settings. The outcomes of this review and classification scheme should help researchers position and plan their future research.

2.3 Limitations of Current Literature

Based on the above literature review, this section first summarises the current status of the knowledge and points out the limitations of the literature. At the same time, this section explains how this research seeks to fill the gap by answering the research questions of this study.

Coordination is a contemporary problem in both DSD and agile contexts and current literature has acknowledged multiple issues. Several kinds of literature have documented coordination problems and challenges related to DSD or agile, while the combination is not well examined in a real-world situation. Some of the literature advocated for agile approach to mitigate DSD issues, whilst others believe both DSD and agile approaches contradict each other in nature, therefore, imposing a coordination burden. Temitayo et al. (2017) highlighted the need for empirical in-depth studies to investigate the fundamental challenges in adopting Scrum in DSD projects and strategies to address those challenges. Their study also outlined the importance of studying the contextual factors and their impact on the effective use of Scrum practices in DSD projects. Hence, there is a need for a comprehensive study of challenges related to coordination while combining agile and DSD, which is answered by RQ1.4. While a taxonomy of dependency has been proposed in an ASD context, its applicability and validity in the DASD context have not yet been evaluated. The

research reported in this thesis seeks to evaluate the existing taxonomy by answering research question RQ1.1.

Most studies in the field of DASD described a number of agile practices as coordination mechanisms and consider ways of formulating them as strategies for effective coordination. While some studies suggest adopting a pure agile approach, others suggest a hybrid approach can be more beneficial. However, both approaches have a record of failure and there is no guideline for situations in which the agile or hybrid approach can better support coordination. Herbsleb (2007) articulated a similar problem by asking what practices are effective in DSD and when. Though his question was targeting DSD projects, this question applies to DASD projects as well. Moreover, coordination mechanisms in one context may not necessarily exhibit the same level of performance in other contexts, due to other antecedents such as task complexity, team configuration, the technology used and organizational structure, as well as dispersion factors. Therefore, there is a pressing need for research that investigates the effectiveness of coordination mechanisms in the DASD context and that at the same time looks for emerging technologies that can support coordination effectiveness. The current research addresses this knowledge gap by answering RQ1.2 and RQ1.3.

The review outcomes singled out that several studies have mentioned coordination effectiveness as a prerequisite for project and team performance. Some have made effort to define the construct of coordination effectiveness from an agile project context (D. E. Strode et al., 2011), while others defined it from a distributed project context (Espinosa, Lerch, et al., 2002). However, there is no literature that has covered both these contextual aspects and provided a comprehensive understanding of the necessary conditions for coordination effectiveness in globally distributed agile teams (Bick et al., 2017). This particular research seeks to redefine the concept of coordination effectiveness in DASD by answering RQ1.2.

Finally, most of the review studies conducted in DASD highlighted the lack of theoretical models or frameworks based on empirical works to provide a generalised

understanding of agile adoption in DSD projects (Jain & Suman, 2016; Vallon et al., 2018) and coordination in particular. Accumulating the knowledge gathered from the supporting questions, this research strives to develop a comprehensive understanding of coordination and, further proposed a theoretical model of coordination in DASD to answer the overarching research question (i.e. RQ1).

2.4 Preliminary Model Development

One of the key aims of this research is to develop a conceptual model of coordination in DASD based on prior theoretical understanding elicited from the existing literature and data analysis findings. A preliminary conceptual model has been developed by combining the understanding of the foundational concepts of coordination in similar contexts, which is further amended at a later stage to form the complete model. The benefit of this approach is that having a priori specification of the constructs is helpful to the design of any theory-building research (Eisenhardt, 1989). As discussed, the researcher systematically reviewed the extant literature to understand the requirements of coordination and what measures can lead to successful coordination outcomes. This understanding is combined into a preliminary model of coordination, which is illustrated in Figure 2.6.

Coordination in the DASD context can be well described adopting an 'input-process-outcome' (I-P-O) model (McGrath & Hollingshead, 1994) that has already been used in previous coordination research (Espinosa, Lerch, et al., 2002; Espinosa & Pickering, 2006; Li & Maedche, 2012). The input variables (I) alternatively referred to as situational factors, will include several contextual factors that affect the coordination process. The variables will include geographical distance, temporal separation, and other factors relating to task, team, organization, and technology. Process (P) will comprise a mix of prioritised coordination mechanisms termed a coordination strategy. The coordination strategy will include both explicit and implicit coordination mechanisms, to effectively satisfy coordination needs. And finally, the coordination outcome (O) can be portrayed by coordination effectiveness and coordination costs

which can lead to better team and project performance. This research initially adopted Strode's definition of coordination effectiveness (D. E. Strode et al., 2011) and further refined it based on the data analysis findings to match the context of this research. According to Espinosa & Pickering (2006), success of the coordination process should also consider the substantial cost of coordination incurred in managing distributed task dependencies, including extra effort for communicating, clarifying misunderstandings and costs of delay and rework involved due to misunderstanding. For this reason, this model includes both coordination effectiveness and coordination cost as measures of coordination outcome as shown in Figure 2.6.

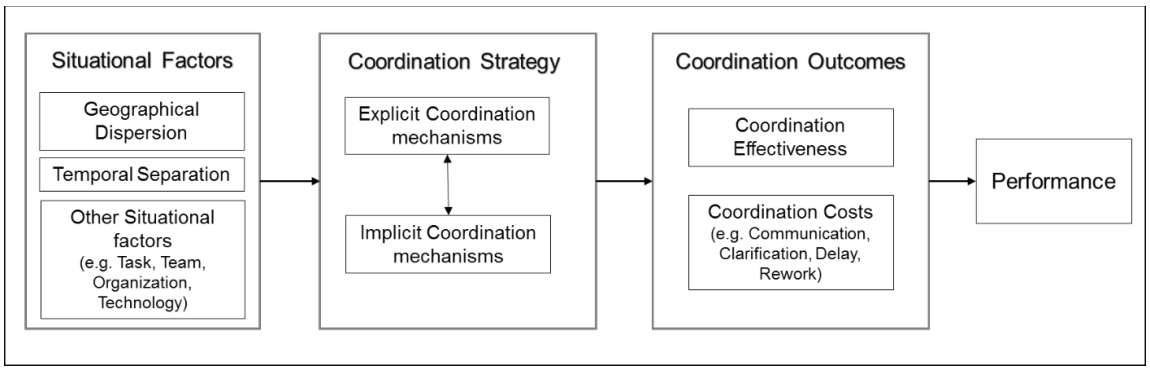

Figure 2.6: The preliminary conceptual model of coordination in DASD

As mentioned earlier, a definition of coordination effectiveness exists which is appropriate for the co-located agile context, but its applicability in distributed agile projects is not established yet. Following this as the key concern of this research, a preliminary model of coordination effectiveness has been formed drawing upon two empirical works by Espinosa et al. and Strode et al. (Espinosa, Lerch, et al., 2002; D. E. Strode et al., 2011).

Firstly, Espinosa et al. (Espinosa et al. 2002) presented an integrated framework of team coordination incorporating both explicit and implicit coordination mechanisms (see Figure 2.3). This framework is applicable for asynchronous and spatially dispersed environments. They emphasised a combination of explicit and implicit mechanisms to achieve a 'state of coordination' (i.e. coordination effectiveness). In distributed development settings, explicit activities (i.e. task programming, team communication) cannot effectively coordinate the needs of team members. The reason is that effective

coordination demands a shared cognition among the members about the task, process, people, expertise, and availability, which helps them to coordinate effectively. Additionally, explicit coordination has a significant impact on implicit coordination. Therefore, any theory related to distributed team coordination should consider both the explicit and implicit mechanisms for effective coordination.

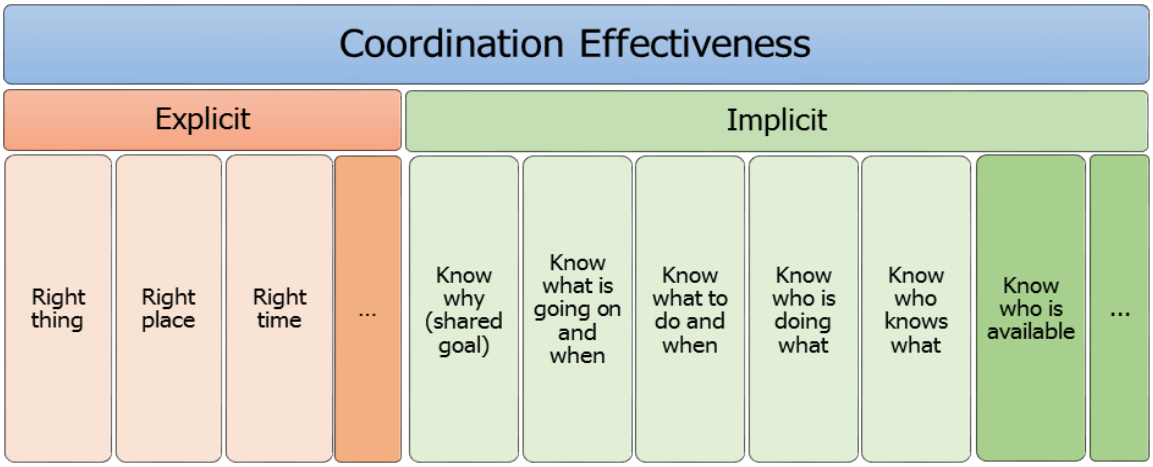

Figure 2.7: Initial Model of Coordination Effectiveness in DASD adapted from Strode et al. (2011)

While developing a coordination perspective on ASD, Strode et al. (2011) proposed a theory of coordination based on three independent agile projects and one non-agile project. They proposed a coordination effectiveness perspective which exploited Espinosa et al.'s work discussed in earlier paragraphs. While conceptualizing, Strode et al. considered an explicit part (i.e. 'right thing' should be present in the 'right place' at the 'right time') and an implicit part (i.e. Know why, Know what is going on and when, Know what to do and when, Know who is doing what, and Know who knows what) and several influencing factors (i.e. the size of the project, its complexity and level of uncertainty). Finally, they proposed a definition of coordination effectiveness that is applicable for co-located agile development projects. Due to the presumable loss of better understanding of the dependencies, shared goals (for particular sprint), location of expertise, and lack of overall knowledge of sprint planning, distributed agile development potentially differs from that undertaken in a co-located context. Moreover, this definition lacks in covering some of the coordination issues that are particular to DASD such as: knowing who is available in other sites (i.e. presence awareness)

(Espinosa, Lerch, et al., 2002), standardization of artefacts, tools, norms, work process and practices, building mutual trust, which motivates further research to validate and upgrade the definition for Distributed Agile Development context. For example, the model of coordination effectiveness is the refined form of Strode et al.'s model including a new implicit component "know who is available" based on Espinosa et al.'s suggestion (shown in Figure 2.7).

From the above discussion, it is worthwhile to plan and position research to develop a conceptual model of coordination effectiveness using Strode's theoretical model of coordination effectiveness as a platform. The preliminary models adopting the above studies are presented in Figure 2.6 and Figure 2.7, which are refined and finalised in a later stage based on further literature review and case study data analysis.

Chapter 3 : Research Design and Methodology

A research methodology offers ways to systematically address the research questions. Such a methodology will contain a research strategy and a research design that includes data collection and data analysis methods. In this study, the research strategy constitutes a mix of positivist and interpretivist paradigms applied in a multi-case study approach. This methodology is appropriate for developing theories by analysing real-world data coll(Eisenhardt & Graebner, 2007)s (Eisenhardt & Graebner, 2007).

The central aim of this research is to understand the ways Distributed Agile Software Development (DASD) teams can achieve effective coordination. To build the understanding, this work first focuses on identifying the key dependencies and mechanisms applied in each case. Further investigating these data, this study seeks to understand what constitutes an effective coordinated state and the factors that can influence the constituents of coordination effectiveness.

In the second phase, this work establishes the relationships between the techniques and tools adopted for coordination and the components of coordination effectiveness i.e. which techniques and tools contribute to which components. While doing so, the researcher also identifies the challenges that can hinder the effectiveness of coordination and possible opportunities to mitigate those challenges. Finally using those multi-phased outcomes, this work will fulfil the aim of the research by interpreting the outcomes to answer the research questions. The research questions are presented in the following to facilitate a better understanding of the relevance of the adopted research design and applied methods:

RQ1: *How are the key dependencies coordinated effectively in DASD?*

RQ 1.1: What are the key dependencies in DASD?

RQ 1.2: How are they coordinated?

RQ 1.3: Is this coordination effective?

RQ 1.4: What are the challenges and factors associated with the coordination mechanisms?

This chapter contains a discussion about the selection and justification about research strategy and approaches applied in this study. The discussion has the following structure:

- Research design
    - Methodology selection
    - Case study research
    - Case study protocol
- Research implementation
    - Unit of Analysis
    - Case selection
- Data collection
- Data Analysis
- Ethical considerations
- Validity, reliability, rigor and relevance
- Summary

3.1 Research Design

The selection of research methodology needs to be appropriate to address the study's research questions. The researcher needs to explore the available research methods to investigate the suitability and applicability of each in the research. The selection of the research methodology will further guide the research design that includes procedures to summarise the outcomes from a broad range of assumptions through the data collection and analysis (Creswell & Creswell, 2017). In any Information Systems (IS) research, the research topic and philosophical perspectives lead to the selection of research methods (Dubé & Paré, 2003). This research follows a multiple case study approach that helps to explore, understand and explain a contemporary social-technical phenomenon (Klein & Myers, 1999). In the following sections, we

discuss how the philosophical perspective of the researcher and the primary research questions influenced the selection of this particular research method.

3.1.1 Methodology selection

3.1.1.1 How the philosophical perspective influences the research approach

The selection of any research methodology is essential to address specific research questions that support the research paradigm. The research paradigm or philosophical stance comes from the belief system that originated from the researcher's epistemological and ontological beliefs (Guba et al., 1994). There are several epistemological beliefs already established in the research paradigm. Each epistemology defines 'what is true knowledge' and 'how to obtain it'. At the same time, ontological beliefs are based on the world's reality. Ontological believers argue that reality is fixed irrespective of how we attempt to understand it. Understanding the difference between these two beliefs is an integral part of any research because the stance we adopt will affect the choice of methodology, and that will lead us to acceptable evidence to answer the research questions. People possess different philosophical stances, Easterbrook et al. (2008) suggest that the researcher should be explicit in their philosophical stance as it will help to pose logical arguments for their methodology selection.

There are four predominant philosophical stances already established in software engineering research, and those are: *Positivism*, *Constructivism or Interpretivism*, *Critical Theory* and *Pragmatism* (Creswell & Creswell, 2017). Each of the philosophical stances is discussed in the following paragraphs to help the reader to understand the stance adopted in this research.

Positivism

This philosophical stance is based upon the belief that all gained knowledge is logically extracted from the observed phenomena. In simple words, any positivist will observe a set of prevailing phenomena in their contextual setting. They will try to understand them and relate them to other phenomena to create valid knowledge out of them. They also

believe that these relationships are repeatable and measurable in similar contexts. Positivists are also referred to as *reductionists* as they seek to break things into smaller components to better understand them. From the ontological perspective, positivists try to establish that there is only one reality, and it is knowable by observing the phenomenon. In the real world, these beliefs are questionable due to doubts about the reliability of the observation of the world (Easterbrook et al., 2008). Due to this doubt, most positivists focus on developing falsifiable hypotheses and try to test them to establish and validate their knowledge. A positivist also believes that any social phenomena and their meanings are independent of social actors and should satisfy expectations of falsifiability, logical consistency, relative explanatory power, and survival. Hence, case studies, surveys and controlled experiments are most closely associated with positivism.

Constructivism

Constructivists are concerned about people and their relationships in a particular social setting and what influences their behaviours and actions. Constructivism is also commonly referred to as *Interpretivism* (Klein & Myers, 1999) as the researcher tries to interpret the social interactions and the reasons behind those actions. Interpretivists believe that there are multiple realities possible in any social setting and each reality may represent different social situations. Each social situation influences the social interactions between the actors. A researcher following this paradigm argues their perceptions of the causality of each social interaction that creates valid knowledge. For example, in a Distributed Software Development (DSD) context, different teams adopt and perform the same agile practices in different ways. Each team has their own thinking and reasons for their choice. Constructivists study those occurrences and attempt to build theories from emerging knowledge. As this paradigm concentrates on the social settings and interactions and activities between the actors, rich qualitative data is necessary. Therefore, research methods such as ethnographies, exploratory case studies, surveys are most often used in Constructivism.

Critical Theory

Epistemologically, Critical theory researchers believe that the current view or thoughts about the system are strongly entrenched by power structures. Critical theorists attempt to relieve people from these restrictive systems of thought (Calhoun, 1995). Critical theory research challenges the existing knowledge and critically evaluates the reality of that knowledge. Finally, the outcome of this research makes an effort to improve the current understanding and practices. Therefore, researchers in this belief seek to actively engage the people whom it will help, and they perform the emancipatory role. From the ontological viewpoint, critical theory research is a holistic approach. It covers both historical and contextual elements in the study so that the research can include the historical actions and their perspectives. At the same time, it validates the present actions to find opportunities for constant improvement. Hence, critical theory research tends to follow longitudinal or ethnographical methods (Orlikowski & Baroudi, 1991). However, action research also closely relates to this philosophical stance.

Pragmatism

The Pragmatism paradigm acknowledges the interplay between knowledge and action. Pragmatic researchers believe that knowledge is not absolute for every single occurrence in the world. All the actions and their outcomes (i.e. changes) are relative. It means that no action is effective and applicable in every other scenario that are similar. It varies from person to person which makes the knowledge relative to the person who is observing. Pragmatists believe that knowledge can vary based on the method by which it is obtained (Menand, 1997). Therefore, researchers in this paradigm attempt to find what action works at the time can be entitled as truth. As a result, any method that gives the researcher an opportunity to intervene in the world instead of just observing, is more appropriate for this paradigm. Moreover, the pragmatic paradigm emphasises more on practical knowledge over abstract knowledge, so whatever method is fit for that purpose can be applied. A mix of multiple research methods can be used to

identify and solve the problems under study. As per Goldkuhl (2012), a combination of Action research and Design research is the best option in such cases. Action research enables the researcher to actively intervene in the problem domain and effect changes that are necessary. On the other hand, the design research approach supports building artefacts as an outcome of the research. Due to the inherent belief system, Pragmatic research usually faces criticism about the validity of the outcomes. Pragmatic researchers rely on the consensus of the participants to verify the truth uncovered during the process.

3.1.1.2 Paradigm for this research

Guba et al. recommended three factors to be considered while choosing the paradigm for any research. Those are: the basic belief system of the researcher, the research questions, and the nature of the phenomenon of interest (Guba et al., 1994; Orlikowski & Baroudi, 1991).

The definitions and examples of different paradigms suggest that both positivist and interpretivist paradigms help to develop a view of the world in a particular context and can be used to develop theories. Hence, both paradigms have merits to be accepted for this study. Critical theory is not appropriate for this study because the researcher is not performing any critical evaluation of any theory or existing knowledge. Rather, the researcher is seeking to develop a theory of coordination in a specific context, which could further lead to critical theory research. Similarly, pragmatic beliefs are also not appropriate for this study. The research participants will primarily be involved in the data collection and, possibly, in the evaluation phase, but they will not be involved in other research activities such as Action research. Rather, this study will investigate the coordination mechanisms for assessing coordination effectiveness, which can further be reported as a recommendation for practitioners. These recommendations can be used for solving coordination issues, which is the primary intention of Pragmatists. Though part of the study objective overlaps with the Pragmatic paradigm, the overall research objective is different.

The research questions of this study could reasonably be answered by either a positivist or interpretivist approach, given the focus of this research is to understand the coordination needs and their management techniques in DASD projects. Both positivist and interpretivist perspectives could be applied to understand the phenomenon and develop theories in this context.

The development of theories is considered to be a significant feature of any scientific study. Theories help to identify any phenomenon and then explain the reasons for the occurrence so that predictions can be made to contribute to changes. One of the key aims of this study is to develop theories for effective coordination in DASD context. All four philosophical stances have merits that contribute to the theory-building process. Easterbrook et al. (2008) explained how each stance can have a role in the theory-building process from a software engineering perspective. They mentioned that "*To the positivist, science is the process of verifying theories by testing hypotheses derived from them. To the constructivist, science is the process of seeking local theories that emerge from (and explain) the data. To the critical theorist, theories are assertions of knowledge (and therefore power), to be critiqued in terms of how they shape that power. To the pragmatist, theories are the products of a consensual process among a community of researchers, to be judged for their practical utility*."

This research will employ a mix of positivist and interpretivist approaches following a well-established framework (A. S. Lee, 1991). His integrated framework consists of three levels of understanding, Subjective, Interpretive, and Positivist (see Figure 3.1).

The first level understanding, i.e. *Subjective understanding*, is gained by understanding the coordination process. This understanding consists of daily activities and interactions that are purposefully performed to collaborate in a DSD process. While collecting data, the researcher asked questions to get individual participant's perspectives about their day-to-day needs for collaboration. Their perspectives are accumulated to form a common sense of coordination needs that give rise to collaborative behaviours in the form of coordination.

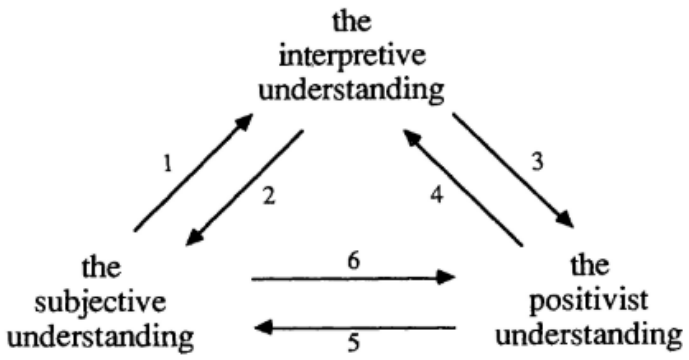

Figure 3.1: Positivist and Interpretivist approach taken in this research (A. S. Lee, 1991)

The second level understanding, i.e. Interpretive understanding, are devised from the researcher's interpretation of the first level of understanding. The researcher postulates a common-sense understanding of the dependencies, coordination mechanisms and coordination effectiveness. While reporting their interpretations, the researcher uses thick descriptions, pictorial presentations, and quotations from the interview transcripts to make it robust and meaningful.

Finally, the third level understanding, i.e. Positivist understanding, is created in the form of constructs, their definitions, and relationships between them. This understanding belongs to the researcher that is used to develop theories, and formal assumptions to explain reality. In this research, the key constructs relate to dependencies, coordination mechanisms and effectiveness of coordination that help to develop a theory of coordination in the DASD context. The theoretical assumptions also consider relative factors that might influence the key constructs.

3.1.1.3 How our study topic influence on research method

Yin (2018), One of the pioneers in case study research, suggested that the selection of a case study as the research method should depend on three primary factors. First, the primary research question uses 'how' or 'why' keywords that focus on the phenomena of interest. Second, the investigation will be performed in an environment under which the researcher has minimal or no control over the events. Finally, the intention of the research should focus on understanding a phenomenon that prevails in a real-life context.

This very research attempts to explore and understand the way software developers works are coordinated among distributed segments while using Agile development methodology. This understanding will be achieved by investigating the coordination needs that emerge during the software development process and *how* those needs are fulfilled using different mechanisms. The mechanisms involve people, their efforts and behaviours towards development of the software in a synchronised way. The role of the researcher would be collecting all these data without intervening the actions and behaviours of the participants. The data collection would focus on capturing all the coordination activities from the real environment. Further analysis of these real-world data would help us to develop a clear understanding about coordination in that context. Therefore, the case study research method is a suitable choice for this research.

Qualitative research methods are considered to be most suitable to study topics involving human subjects. Seaman (Seaman, 1999) argues that "*The principal advantage of using qualitative methods is that they force the researcher to delve into the complexity of the problem rather than abstract it away*". It is well established that coordination is a complex phenomenon in the DSD context. An extensive investigation of the scenario will help us to better understand the phenomena. While performing such investigation, exploratory questions are suitable as they can open new knowledge arenas which is otherwise difficult (Easterbrook et al., 2008). Easterbrook et al. suggested that a research method that offers rich data should be selected for answering exploratory questions. The data consider as rich are not limited to numbers, such as words and pictures. Qualitative data offers rich as they are collected from study participants' settings. This setting allows the researcher to understand the participant's decisions and actions, at the same time, understand the context which has influenced the actions performed (Myers, 1997). The key research questions and the context of this research requires an empirical method that supports rich data collection and qualitative data analysis techniques. An *Exploratory Case study* research design

fulfils all these requirements which makes it a suitable choice for conducting this research.

Another objective of this study is to build theories based on our analysis findings. Our developed theory would help us to explain the coordination activities in a DASD context and make predictions to improve coordination performance. Exploratory case study method is better suited to induct theories by investigating some phenomena in real context (Eisenhardt, 1989). From all the above discussion, it is clear that an exploratory case study research closely aligns with the research questions and study objectives, which is rational for our research method selection. In section 3.2.3, we have discussed the characteristics of the exploratory case study method along with the advantages and disadvantages that need to be considered.

3.1.2 Discussion about case study research

3.1.2.1 Definition of case study research

Case study research has been the most common research method in social science research. However, this method has received great attention in information systems research, especially in software engineering. The fundamental thinking behind adopting case studies in this field is to increase the understanding about different phenomena which will further improve the process and outcomes (Runeson et al., 2012). To identify the applicability of case study approach, the researcher needs to understand the definition and relevant criteria of this method which will guide further research activities.

Yin (2018) instituted the twofold definition of case study research covering the scope and feature. He defines case study as “an empirical method that investigates a contemporary phenomenon (the “case”) in depth and within its real-world context. Especially, when the boundaries between phenomenon and context may not be clearly evident”. The *feature* specific definition states, “*A case study copes with technically distinctive situation in which there will be many more variables of interest than data points and as one result benefits from the prior development of theoretical propositions to guide design, data collection, and analysis, and as another result relies on multiple*

sources of evidence, with data needing to converge in a triangulating fashion" (Yin, 2018).

Runeson et al. (2012) have translated popular case study definitions posed in other fields and defined case study research for software engineering as, "*Case study in software engineering is an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified*".

From the above definitions of case study research, it is clear that five essential aspects are there to distinguish case study from other similar research methods. We call them similar methods since each method might comply with some of the aspects. The five aspects are: *empirical investigation*, *contemporary phenomenon*, *rea-life context*, *multiple data sources* and *unclear boundary between phenomenon and context* (Runeson et al., 2012).

Besides case study, there are other empirical methods available that depends on rich data sources to study a contemporary phenomenon. Those are: *Ethnography, Action research* and *Grounded theory* (Easterbrook et al., 2008; Myers, 1997). Each method has its own protocol and preference of data collection or data analysis techniques. And each one comes with different kinds of benefits and drawbacks. It is recommended to know the distinctions between case study and other methods to get most out of this method (Yin, 2018).

There are no definite criteria to differentiate each of them which makes it difficult for the researcher to select the appropriate one. However, it is better to present how these methods are not the best fit for this research. In the following, we will discuss each of them in brief with the definition, key features and how they differ from the research objective in terms the context, research question, researcher's involvement.

3.1.2.2 Discussion about other empirical methods

Ethnography

Ethnography is a research method that focuses on studying any community and their attitude towards a particular phenomenon. In a software engineering domain, the community may represent the technical communities who are involved in the development process. An ethnographer attempts to investigate their collaborative actions and strategies while participating in collective work. This method can be useful in studying Computer Supported Collaborative Work (CSCW) domain and develop collaboration tools. However, this method lack attention in current empirical software engineering research (Sharp et al., 2016).

Ethnography notably differs from case study in the sense that it avoids using any pre-existing theories and tries to capture the perceptions of the community members towards their social and cultural setting (Easterbrook et al., 2008). Whereas in a case study, researcher depends on theoretical understanding which guides their course of actions. For instance, explanatory case studies are well-known method to confirm existing theories. Ethnography concentrates on the community; their social and cultural settings and the researcher observes the community to develop a rich and detailed understanding of their culture. Since, the aim of the research does not focus on the cultural or social understanding, ethnography is not a suitable method for this study.

Action Research

Action research is a popular research method where the research attempts to influence or change some aspects of the phenomena of interest to solve any contemporary problem (Robson, 2002). The key feature of this method facilitates the researcher to actively participate in the study and purposefully intervene the undergoing actions to improve the situation. Action research is gaining popularity in IS and SE research since it is more practitioner-oriented approach to study and solve any problem.

From the philosophical perspective, action research closely resembles Critical Theory. An action researcher actively engages in the study with a *Problem Owner* to identify

the problem and iteratively attempt to solve the definite problem. The researcher captures the experiences throughout the process and investigates them to generate a critical understanding of the situation which can be supportive to other practitioners.

Action research is similar to a case study as both attempt to solve a contemporary phenomenon. However, they strongly contradict each other in terms of the researcher's involvement in the study. While conducting a case study, the researcher participates in the case to observe and collect data rather than being involved in the study that is done in action research. The researcher of this study will observe and collect data about the coordination needs and their management for the case setting without interfering with the existing process. This objective fits well with the case study method which makes it suitable for this study.

Grounded theory

As the name suggests, the Grounded Theory (GT) is an empirical research method that intends to develop a theory that is grounded on the data. The GT research follows a rigorous process where data goes through continuous comparison until a set of conceptual hypotheses is generated which further produce a theory about the investigating phenomena (Glaser & Strauss, 1967). In GT, the research needs to gather data and then systematically generate concepts and categories from the data which indicates a pattern. These patterns are relevant and grounded on data which further helps the researcher to generate hypothesis.

Glaser & Strauss (Glaser & Strauss, 1967) restrictred the researchers to use their pre-exisitng values and knowledge to interpret the data, rather encouraged them to use them as another source of data which will support the categorisation and theory generation process. Because of this restrictions, GT is not sutiable for this reseach. This research uses the researcher's pre-existing knowledge and experience to develop a preliminary conceptual model which guides the data collection and data analysis process. The preliminary model would be reviewed and updated though the analysis to develop a theory of coordination in DASD context. Case study method is the best well-

known alternative for generating and testing theories which is adopted in this research (Eisenhardt, 1989).

3.1.2.3 Case Study Research

Case study research methods can be of four types depending on the purpose or motive of the research aim. These are *Exploratory*, *Descriptive*, *Explanatory* and *Improving* (Robson, 2002). As per Robson's classification:

Exploratory or Evaluatory case study seeks new insights while investigating what is happening and generates new ideas in the form of hypotheses and build theories.

Descriptive case study investigates the ongoing events to illustrate a particular situation or phenomenon.

Explanatory case study seeks an explanation for the current status or situation of a phenomenon.

Improving case study attempts to improve aspect(s) of a certain phenomenon.

Exploratory case studies are the most popular in software engineering research. The primary reason for this popularity is the belief that case study is more suitable for exploratory purpose. But there are evidence that other types of case studies are also helpful and been used in this field (Runeson et al., 2012). The application of different types of case studies profoundly depends on the philosophical perspective. Explanatory case study closely relates to the *Positivist* approach where the researcher seeks evidence to conform or generalise formal proposition and test existing theories. On the contrary, *Constructivist* or *Interpretivist* approach are similar to exploratory or descriptive type as they rely on researcher's interpretation of the study phenomena and develop new theories. Since this study aims to develop a conceptualisation of coordination, an Exploratory case study method is adopted.

Single vs Multiple case study

Single and Multiple-case studies are two primary types of case study research design. Yin (2018) has proposed a total of four variations of case study design as shown in Figure 3.2, those are:

- Type 1: single-case (holistic) design,
- Type 2: single-case (embedded) design,
- Type 3: multiple-case (holistic) design, and
- Type 4: multiple-case (embedded) design.

It is recommended to select one of the four variations that is more suitable given the nature of the research. Likewise, the data collection process relies on the decision of the design. In a single case, the researcher wants to study one single entity (e.g. a person from a specific group) or a single group (e.g. a group of people) or a single event or incidents (Yin, 2018). A multiple-case study design consists of two or more cases that are being studied under a single case study research. One of the substantial motivation behind a multiple-case study design is that the researcher wants to get a broader view of the phenomena of interest (Easterbrook et al., 2008). This broader view can provide a generalised understanding by performing cross-case comparison of the cases. The choice of the design entirely depends upon the context and the type of case that are being studied. For example, if a single case can provide sufficient data that can be investigated and draw empirical outcomes. On the other hands, multiple-case study can give a similar (literal replication) or a contrasting (theoretical replication) outcome that can better validate the research contributions (Yin, 2018).

As mentioned earlier, this study will follow a multiple-case study design. In this research, The phenomena that is under focus of this research is coordination and the context is DASD. The understanding this study wants to explore is that how distributed teams achieve coordination in the DASD context. Additionally, we are interested in exploring how agile methods can support effective coordination in this context. One of the ways to achieve this broader understanding is doing within-case and across-case

data analysis, which is possible in multiple-case design (Yin, 2018). Another benefit of using multiple-case study design is that it gives the opportunity to replicate findings across case which can lead to discover new and contrasting outcomes (Baxter & Jack, 2008). This opportunity is not possible if the researcher selects a single-case study design. In a multiple-case holistic case study, each case represents different contextual settings, and hence, researcher will be able to explore the impacts of the contextual factors. Such kind of research outcomes are considered to be robust and reliable (Baxter & Jack, 2008).

While selecting a multiple-case study design, the researcher needs to consider its relative strengths and limitations. Multiple-case study can deduce compelling evidence which can provide reliable outcomes. However, this type of study is considered to be expensive as it requires extensive time and resource to conduct. Moreover, case selection is a crucial step in multiple-case study. Cases need to be selected based on the intended replication logic of the outcomes, either similar results (i.e. literal replication) or contrasting results but for reasonable cause (i.e. theoretical replication). Yin (2018) suggested that for achieving literal replication, two or three similar types of cases need to be selected. In contrast, for theoretical replication, more numbers of cases should be selected that are believed to provide extreme outcomes. This research intends to achieve a literal replication of outcomes; therefore, two cases are selected that are believed to produce similar outcomes. Moreover, it is a as realistic number of cases to study within the limited timeframe of this doctoral study. But we will be open for any contrasting outcomes that can guide future research directions.

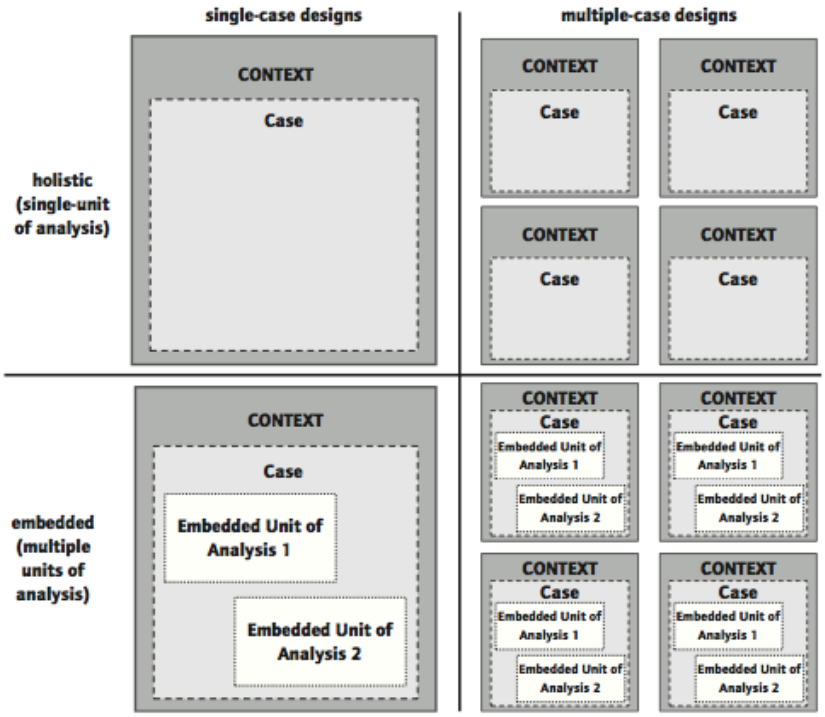

Figure 3.2: Types of case study design (Source: COSMOS corporation)

3.1.2.4 Strenghts and Limitation of case study

It is also important to acknowledge the strengths and limitations of the selected method. It is beneficial to know the strengths and avail them while doing the study. On the other hands, it is also neccesary to acknowedge the limitations and take measures to mitigate them.

Like any other, case studies have openly acknowledged strenghts and weaknesses. Case study research can give deeper insight about the occurance of certain phenomena and its related contextual factors. Case study research desing allows flexibility in selecting the case whether single or multiple. Case studies are applicable for both generating and testing theories in the form of logical assumptions (aka. hypothesis) or conceptual frameworks. In general, theories generated from case studies are novel, testable and empirically valid (Eisenhardt, 1989).

Although case study method has the above strengths, there are some major limitations of this method. Case study research design supports high degree of realism which comes at the cost of level of control. Case study allows very minimal level of control on the study subjects so that the study can be conducted in real-life settings which may

result in major deviations in the expected results. Another drawback of case study is the interpretation and researcher bias (Easterbrook et al., 2008). While conducting a case study, there is a possibility that the reseracher may form bias for the subject, or in the data collection, or interpreting the data. Since the reseacher is also a human being, they can be subjective in several occasions leading to researcher's bias which makes the design vulnerable. Moreover, due to the design flexibility, it is difficult for the reseracher to achieve high level of rigor in case study. Since there is no strict guidelines how and from whom the data will be collected and analysed, it is more open for the reseacher to choose from a list of options which may raise validity threats. It is also challenging to report the finding to make it plausible for the readers. The research, being a human, tends to prejudice about what is important and may ignore other important aspects. On the other hand, the interpretations and explanations are subjective to the researcher's personal knowledge which may not be easy for the readers to understand.

Each method has some advantages and disadvantages which the researcher needs to acknowledge. Since the selection of the research method depends on several factors and the same study can be conducted using multiple methods (Easterbrook et al., 2008), the researcher should take the responsibility to choose the most suitable one. As per Yin (2018), the criteria to validate the choice depends on the philosophical perspective taken by the researcher. We have already presented our philosophical stance and the case study research is best fit for this exploratory study as discussed. To compensate the issues due the design limitations, we will follow the established case study protocol (Brereton et al., 2008; Maimbo & Pervan, 2005) to minimise validity threats. We will also use our preliminary conceptual framework to analyse and interpret the findings as suggested by Yin (2018).

3.1.3 Case Study protocol

The research activities follow a case study protocol that has been crafted from the well-established protocol templates in this field (Brereton et al., 2008; Eisenhardt, 1989;

Maimbo & Pervan, 2005). Following an established protocol helps researchers to achieve repeatability that improves the validity of the study. A Case study protocol defines a set of process that need to be executed in a structured way to complete the research. The protocol will guide the researcher to know what to do, when and, especially, what not to do whenever applicable. Therefore, it provides a greater validity and, if intended, generalisability of the research outcomes.

The construction of the case study protocol is influenced by the philosophical stance taken by the researcher. However, most of the approaches are quite similar to the high-level design. Eisenhardt (1989), taking a positivist approach to developing theories, suggests eight key steps for case study research as shown in Figure 3.3:

1. Getting started- knowing the problem, define research questions
2. Selecting cases- identify the target population and use theoretical aspects in case selection
3. Crafting instruments and protocols- define data collection methods, prepare the research instruments to contact participants and collect data
4. Entering the field- apply multiple data collection methods from multiple sources
5. Analysing the data- follow the data analysis method(s), perform within-case and cross-case analysis
6. Shaping hypothesis- identify constructs and existing relationships evident in the data and attempt to explain them
7. Enfolding literature- search for similar literatures either to support or contrasting the findings
8. Reaching closure- attempt to reach *theoretical saturation* to confirm that no more interesting findings are achievable

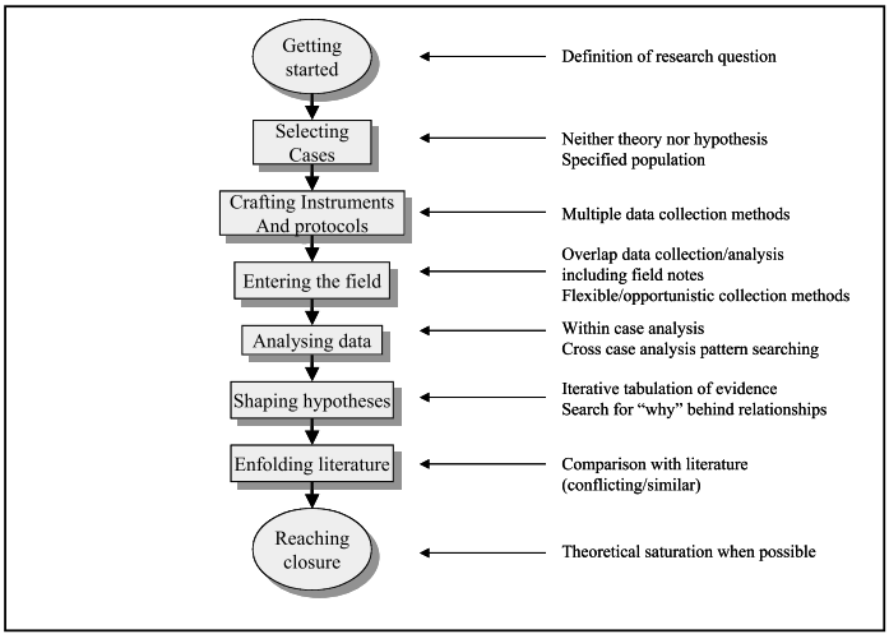

Figure 3.3: Eisenhardt's Case Study Research Framework (Eisenhardt, 1989)

Based on her work, Maimbo & Pervan (2005) developed a unique case study protocol suitable for IS research presented in Table 3.1. This case study protocol outlines the procedures and rules that will administer the research activities. They have adapted Eisenhardt's framework and divided the research activities into three phases: *Model development*, *Model testing*, and *Model refinement*. The model development phase consists of literature review, research model development, research instrument and protocol crafting, and case selection activities. The next phase focus on implementing the model in the form of data collection and data analysis. Data collection and analysis activities can be performed sequentially or in parallel to each other. While collecting data, the authors recommended that the primary focus should be on *Triangulating* both the data and the perspectives. They have outlined a structured guideline for data analysis using an exemplar that uses priori coding of themes and patterns. This priori coding influences the data collection process and is further used in analysing collected data. However, the priori coding technique is not necessary for data analysis, for instance, this research will use the Thematic content analysis technique where the codes are created during the analysis process and further grouped together to develop themes and patterns. In addition, this data analysis process includes 'Within-case' and 'Cross-case' analysis methods to facilitate overarching research findings.

Table 3.1: Case Study Protocol developed by (Maimbo & Pervan, 2005)

Section	Contents	Purpose
Preamble	• Confidentiality and data storage • Publication • Documentation • Layout of protocol	Contains information about the purpose of the protocol, guidelines for data and document storage, publication
General	• Overview of research project • The case research method	Provides a brief overview of the research project and the case research method.
Procedures	• Initial approach to organisations o Selection of cases o Number of cases o Establishing contact • Scheduling of field visits • Length of sessions • Equipment and stationery	Detailed description of the procedures for conducting each case. These procedures should be utilised to ensure uniformity in the data collection process and consequently, facilitate both within case and cross case analyses
Research Instrument(s)	• Research instrument(s) that may either be: a) Qualitative- interview guides utilising either open-ended or close-ended questions b) Quantitative – survey questionnaire applied in face-to-face interviews	It is recommended that these research instruments be highly structured to facilitate the data collection process and uniformity in the collection of said data.
Data analysis guidelines	• Overview of data analysis processes • Details regarding a) How convergence of data from multiple sources will be achieved b) How triangulation of perspectives from multiple participants will be achieved • Description of 'Within case' analysis process a) Descriptive Data b) Explanatory Data c) Individual Case report • Description of 'Cross case' analysis process • Description of 'Cross Sectional' analysis process (where necessary) • Data schema a) Summary of primary data types, sources and purpose b) Summary of secondary data types, sources and purpose • Description of data displays that will be used in analysis • A priori list of codes that will be used during qualitative analysis	Provide detailed guidelines for data analysis.
Appendix	• Participation request letter	Template letter sent to potential participants inviting them to participate.

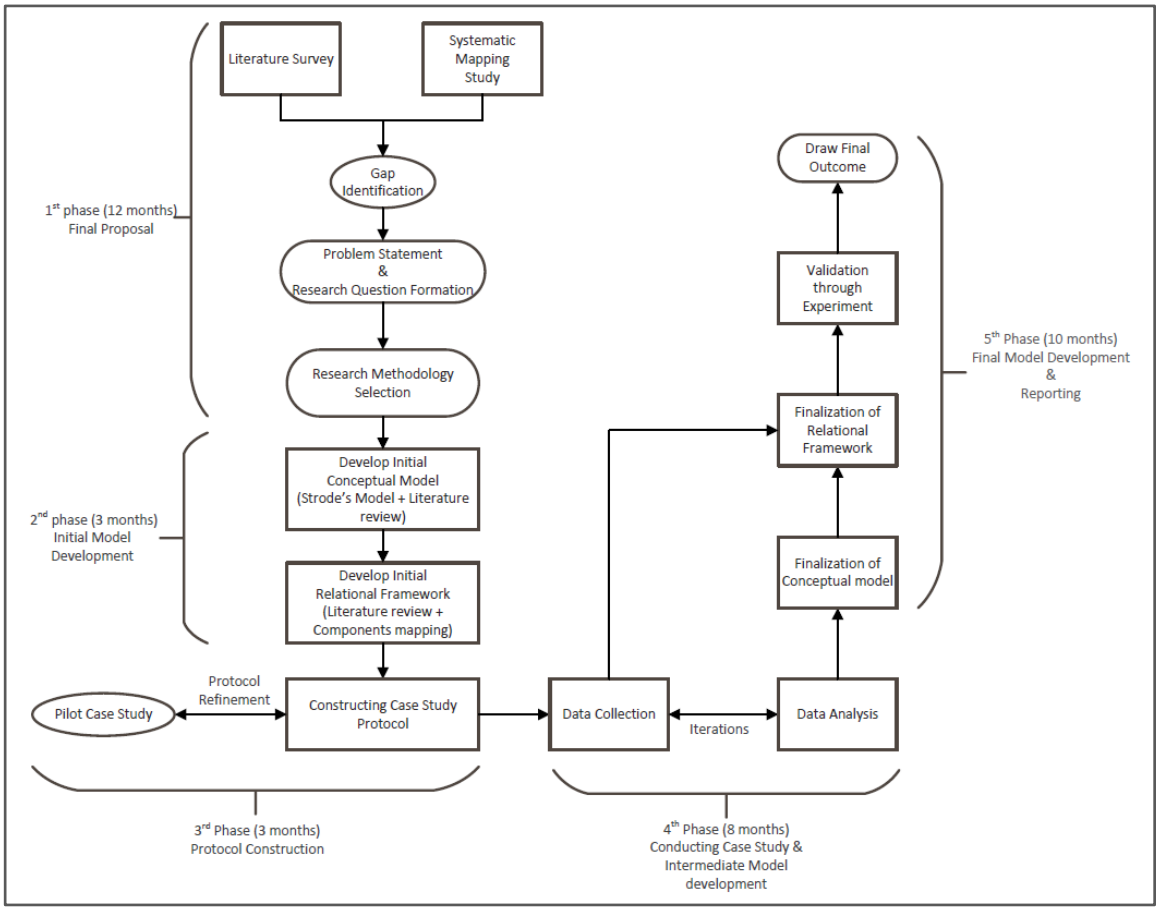

Figure 3.4: The activities flow of this research

In this research, a case study protocol was crafted by adapting the case study protocol templates from (Brereton et al., 2008; Eisenhardt, 1989; Maimbo & Pervan, 2005; Yin, 2018). In this protocol, we have included five phases- model development, designing the study, model testing, model refinement, and finally, validation and closing.

At the same time, we have included detailed activities similar to those (Stake, 1995) which involve the researcher and one of the supervisors separately performing the coding and analysis to support the interpretive approach taken in this research. The case study protocol is presented in Table 3.2, outlining the activities performed under each phase.

Table 3.2: The Case Study Protocol used in this research

	Phase	Stage	Activities
1	Model Development	Getting started	• Literature Survey & Systematic Mapping study • Problem Statement and Research Question formation • Research methodology selection
		Conceptual Model development	• Analyse secondary data • Develop initial conceptual framework
2	Designing the study	Case Selection	• Define Unit of Analysis • Selection of cases • Number of cases • Establishing contact
		Crafting Case Study Protocols	• Data collection guideline • Data analysis guideline • Plan validity • Limitations • Reporting • Schedule
3	Model Testing	Data Collection	• Identify the data to be collected • Identify the data collection plan • Ensure Triangulation
		Data Analysis	• Transcribing the transcripts • Thematic content analysis • Within-case analysis • Cross-case analysis
4	Model Refinement	Shaping hypothesis	• Iterative tabulation of evidence for each construct • Replication logic across cases • Search for the cause, i.e. the 'why' behind relationships
		Enfolding the literature	• Comparison with conflicting literature • Comparison with similar literature
5	Validation and Closing	Reaching Closure	• Refining and finalising the conceptual model • Validate the findings • Develop the Conceptual model

The activities outlined in this protocol are sequentially executed to maintain the rigor of this case study research. However, phase 3 (model testing) and phase 4 (model refinement) activities required to be iterated a number of times for each case. The flow of the research activities indicating their aims to be achieved during that phase are illustrated in Figure 3.4. The model development process has been thoroughly discussed in previous chapters. The rest of the research activities are discussed in the following sections.

3.2 Designing the study

In this phase, the unit of analysis for this case study is finalised, which is followed by the development of the case section criteria and the case selection process as discussed in the following subsections.

3.2.1 Unit of Analysis

Defining a unit of analysis is one of the five components of case study research design (Yin, 2018). A 'unit of analysis' of a case study defines a clear boundary within the case. In more general terms, the unit of analysis of the study helps to specify the unit (e.g. who or what) that will be analysed. For example, the unit might be a particular aspect of the case or a group of aspects or the whole case. Therefore, it is important to define the 'case' before determining the unit of analysis of any case study. Miles and Huberman (1994a) defined the case as, "*a phenomenon of some sort occurring in a bounded context."* The case is*,* "*in effect, your unit of analysis*" (p. 25). Once the case is identified, then a unit of analysis needs to be defined. The unit of analysis can be defined as an individual, or group of individuals, a specific project, process, activity, a particular event, or a series of events, an organisation, a nation, or a geographical location (Miles & Huberman, 1994a). The unit of analysis helps to design the study and determine what data to be collected and analysed under the study.

For this research, the case would be the *Distributed Agile Software Development process*. Under this process, we could investigate the whole software development process, or a specific aspect of the software development process such as the

requirement engineering process, ASD process, and coordination process. To define the unit of analysis from the above options, we have used Yin's (2002) guideline that states, "the definition of the unit of analysis (and therefore of the case) are related to the way the initial research questions have been defined." (p. 22). The key research question of this study focuses on the coordination process to manage the dependencies arising in the DASD process. therefore, the '*coordination process*' is the suitable unit of analysis for this research.

While investigating collective work in IS development, Korpela et al. (2002) recommended that the unit of analysis should be the 'activity', rather than any constituent part of it. They have adopted the definition of the work 'activity' from (Engeström, 1987) which defines development 'activity' as a systematic entity that consists of a number of elements which must fit together to achieve any shared goal. In this research, the definition relates to *coordination process* that can be defined as an activity that constitutes a number of actions as elements that must fit together to achieve an overall outcome, i.e. well-coordinated state. Due to the similarities in definition, this research has selected '*coordination process*' as the unit of analysis following the recommendation of Korpela et al. (2002).

3.2.2 Case Selection

The case selection process is crucial while conducting a case study research. while conducting a multiple-case study, the primary motive is to explore the targeted phenomena through replication strategy. This replication logic is considered to be the strength of multiple-case study design. While designing a multiple-case study, Yin (2002) suggested that the research should carefully select cases that will either provide similar results (literal replication) or contrasting results for known reasons (theoretical replication). The purpose of following such replication strategy is that it will help the research to achieve expected outcomes. Therefore, it depends on the researcher which type of case will be selected based on their choice of replication. For instance, if the researcher wants to achieve both kinds of replication, they need to select some

cases that will aim for literal replication and some for theoretical replication. Another researcher might aim for single type of replication, and hence, will select cases that will support desired replication.

In this research, the case selection process followed Yin's guidelines to choose case that are contextually similar that could support the 'Literal replication' of the cases. The reason for targeting literal replication is that it will enable us to support our preliminary theory. Eisenhardt (1989) regarded that by achieving literal replication, the researcher will get substantial support for the preliminary theory that describes the phenomena. The following criteria are used to select the cases to support the literal replication:

- Each case should have distributed segments involved in their software development process.
- The DSD process should use any of the established ASD methods, e.g. Scrum or XP.
- The adoption of the agile method should be mature enough. Majority of the team members should have a minimum of 1 year of experience.
- Majority of the team members are working together for a reasonable time so that they know how the DSD works.
- The DSD process had a clear business value, e.g. product, that is developed for external clients or for other parts of the organisation.
- A minimum of five and maximum ten members involved in the software development team, so that we can collect sufficient data to ensure data triangulation.
- The Software development process should not be involved in developing highly complex products that might represent extreme or critical case which will not be suitable for literal replication.

There is no established rule to determine the suitable number of cases for a multiple-case research. While achieving replication, the single most important criteria is that the number of cases should be sufficient enough to achieve the replication (Yin, 2018). He suggested that it is possible to attain literal replication by conducting two or three case. But the condition is that the theoretical framework should be straightforward enough to support the replication logics. The term *straightforward* is further clarified by the statement that "*the framework needs to state the conditions under which a particular*

phenomenon is likely to be found" (pg. 46). The preliminary conceptual model established in this study is considered to be straightforward since it is grounded upon well-established definitions and taxonomies (Espinosa, Lerch, et al., 2002; D. E. Strode, 2016) to identify the dependencies and their mechanisms to achieve 'State of coordination' (Espinosa, Lerch, et al., 2002) that are applicable in both distributed and ASD context. Moreover, there is a strict time constraint in the completion of this research work which also influenced the selection of the case numbers. Considering all these factors, this research has selected two literal replicating cases for this case study research.

The selection of a potential case in case study research is crucial as it can illuminate the research questions by providing sufficient access to the appropriate data for this research (Yin, 2002). In this research, the researcher used his personal as well as his supervisor's professional network to identify potential cases. The researcher advertised on the Agile Auckland LinkedIn page (https://www.linkedin.com/groups/56202/) and also presented in the Agile Auckland meetup (https://www.meetup.com/Agile-Auckland/) group to locate candidate cases. There were several candidate cases identified that met the section criteria, but the final selection of the cases was made upon several criteria, such as willingness to participate in this research, agreement to provide access to required data, meet the required distributed structure, uses agile in full or partially for supporting coordination process. Finally, two cases were selected that met all the selection criteria. Details of each case are discussed while reporting the case findings in the next chapters (see Chapters 4 & 5).

3.3 Data Collection Approach

Case study research design offers different data collection strategies from different kinds of sources. While doing a case study, it is important to choose the data sources that can provide sufficient coverage that supports triangulation.

3.3.1 Triangulation

Achieving *Triangulation* is an important aspect of case study research since it enhances the reliability and construct validity of the research findings. Construct Validity is one of the four different kinds of validity as mentioned in Yin's (2018) case study guideline (discussed in detail at the end of this chapter). As mentioned in Yin's book, there are four basic types of triangulation: *data triangulation* (i.e. multiple data sources), *investigator triangulation* (i.e. more than one evaluator), *theory triangulation* (i.e. different perspectives of the same data) and *methodological triangulation* (i.e. multiple methods). Due to the time and single researcher constraint, only data triangulation has been used in this study.

Data triangulation refers to collecting data from multiple sources, for example, participants' interviews, observation of actions or events, document analysis, and physical artifact collection. This study achieved data triangulation in two ways, by collecting data on the same topic from different sources of the same type and by collecting data from different types of the data source. For example, for collecting the same data from multiple sources the researcher asked the same question to different persons performing different roles in the project. And for the second approach, data were collected by using multiple data types, such as participant interviews, observing project activities, analyse different types of documents, such as project documents, system documents, and photographs. We have applied semi-structured interviews as the primary data collection method supported by observations and physical artifacts collections.

3.3.2 Participant Selection

The initial contact was made with one of the representatives of the case organisation and a 'kick-off' meeting is conducted to demonstrate the purpose of this research and the probable benefits of the organisation. After the meeting, a consent form has been sent via email to attain consent for contacting candidate participants. The researcher has sent email invitations including the participants' information sheets (attached in

Appendix B) to the candidate participants. An appropriate candidate would be a member of a software development team that is involved in different phases of the DSD process using agile methodologies, such as scrum master, product owner, developers, testers, and QA. Furthermore, this research has used several other criteria for participant selection from the selected cases, such as participants should have a good amount of experience (e.g. one or more years of experience) as a part of distributed agile project so that they can provide meaningful information on various coordination related aspects of the project, they should be available and accepted the invitation by signing the consent form.

Finally, the participants are recruited based on the above eligibility criteria for the current research. For example, team members from both local (NZ) and global locations will be chosen based on their experience working in DASD. A variety of participants from different roles have been selected to maintain data triangulation. All the participants signed a participant consent form and sent it to the researcher prior to the interview. It is mentionable that the researcher has received ethics approval from the Auckland University of Technology Ethics Committee (AUTEC) prior to commencement of the research (see Appendix A)

3.3.3 Interview and protocols

Interviewing the participants is considered to be a powerful method of data collection in case study research. Interviews are conducted with carefully selected participants from various roles and situations related to the case study topic. Asking specific questions focusing on the key aspects of the phenomena of interest can provide valuable insights reflecting participant's relativist perspectives. It can also provide explanations (e.g. hows and whys) about the key events (Yin, 2018). Interviews can be conducted in three different ways: *structured*, *semi-structured* and *unstructured*. In a structured interview, the researcher will prepare a set of pre-formulated questions. Both the interviewer and interviewee have to follow the regulations of the interview, such as the order of the question, and the time limit for each question. Structured interviews are

helpful to describe and explain the relations between the constructs of the research (Runeson & Höst, 2009). Semi-structured interviews are more relaxed than the earlier format. In this type of interview, the researcher also prepares a set of pre-formulated questions mixing both open and close-ended questions, but there is no necessity to adhere to those questions. The researcher can include any new questions or improvise the existing ones following the conversation. The purpose of the pre-formulated questions helps the researcher to guide the interview process and stay focused on the topic. There is no restriction in terms of ordering and timing allotted for each question. However, there should be an approximate time limit for each interview to provide control. Semi-structured interviews are helpful when the researcher wants to explore any phenomena and explain the way it is present in the current context (Runeson & Höst, 2009). In an unstructured interview, there are no or very few pre-formulated questions prepared beforehand. The researcher has full freedom to ask any sort of open-ended questions related to the topic. In most cases, there is no time limit or other restrictions imposed to provide full flexibility to the researcher. This type of interview is the mostly exploratory purpose (Runeson & Höst, 2009), and the interview sessions are not typically time bounded.

Considering the benefits and limitations, we have chosen semi-structured interviews to support both the exploratory and explanatory purposes of the study. The interviews are conducted involving various stakeholders of the project. The variety of the stakeholders involved in the DASD process can represent diverse views that improve reliability. Like any other data collection method, interviews have weaknesses too. One of the major weaknesses is the *Bias* in articulating interview questions and response bias. Using a semi-structured interview method, helps to avoid the bias issues. Moreover, we have developed a set of questionnaires focused on the key research questions and submitted to AUTEC to get verified. Another problem of interview is the *Reflexivity*, e.g. forcing the interviewee to answer the way the interviewer wants. Though, Reflexivity is considered to be the important feature of the interview, it might be creating problems

while exploring key insights. Following a semi-structured interview protocol, the researcher kept the discussion open for new insights and perspectives to emerge. At the same time, tried to keep each interview session within a time limit (45-60 mins) to reduce the opportunities for the researcher's bias toward the participants or presumptions that are likely to pose a reflexive threat.

The interview protocols of this research are developed as part of the ethics approval requirement following the sample provided in Strode's doctoral thesis (D. Strode, 2012). In her thesis, she has investigated dependencies, and coordination mechanisms in a similar context, i.e. ASD, therefore, it is reasonable to use that as a guideline. We have designed two different versions of the interview protocol, one for the managerial roles (e.g. Product or project managers, Product Owners) and another for the team-level roles (e.g. developers, testers, scrum masters, analysts, team leads). Each version of the protocol is submitted and verified by AUTEC before conducting the interviews (attached in Appendix B).

To prepare for data collection and, at the same time, verify the protocol, the researcher started with an informal meeting with the Product Manager from the first case. The researcher and one of the supervisors conducted the interview following the developed interview protocol. Based on the initial meeting, some changes are made in the protocol before continuing with actual participants. The pilot interview data are only considered for refinement of the data collection plans and protocol refinement. That is why the interview was not recorded and not used in the later analysis process. Besides, this pilot interview data has given a preliminary insight into the organisation's software development process, its products, its purpose, and an overview of the coordination process. This elementary data confirmed the contemporary issues in coordination present in their DASD process. This insight confirmed the efficiency of the protocol to collect necessary data for detailed analysis.

3.3.4 Direct Observation

Direct observation is considered to be another data collection method while investigating concurrent events. In case study research, the researcher aims at understanding a phenomenon in real-world context. Direct observation, without interrupting the events occurrences, can provide such opportunity by closely observing the actions, its participants, and the environmental settings. Such data can provide evidence and explanation of ‘how’ or ‘why’ the events are occurring that directly relates to the investigating phenomena. Direct observation data collection activities can be performed formally or casually (Yin, 2018). While doing formally, the researcher might need to develop an observation instrument that will indicate what types of events or actions will be observed, how it will be conducted and frequencies or schedules if applicable. On the contrary, casual observations are kinds of data collected about the environments or surroundings while taking interviews or field visits.

This research used direct observation as the second data collection method. Since, observational data can provide additional information about the study topic (Yin, 2018), it will be useful to support interview data. We have developed an observation protocol and received approval from the AUTEC as part of the case study protocol. We have used both formal and casual observations in both cases. While doing the formal observation, we have asked for consent to attend some of the group sessions. Our target was to observe the group meetings that include distributed parts to see what types of dependencies are coordinated, how they coordinate, what tools they use and gather experience as a participant in those sessions. To enhance the reliability, this research followed Yin’s suggestion and two people (the researcher and one supervisor) participated in most of the observation sessions. Each observer has taken individual notes that have been used to verify the collected data. At the same time, details of all the physical and electronic materials used during each observed session are also collected and further analysed to identify their contributions to the coordination process. However, no actual physical documents are being collected or analysed in this study.

Both the investigators have observed a number of coordination sessions for both cases and the list of the sessions observed is presented in Table 3.3. These meetings provided a good amount of evidence related to coordination involving distributed members for each case.

3.3.5 Photographs

Photographs are the final source of data used in this research. Photographs are used to collect evidence of physical artefacts such as Squad boards, Dependency boards, Definition of done, Definition of ready, retro wallboard, scrum-of-scrums board. These are publicly visible artefacts that are obtained while taking interviews and observational sessions. Due to access concerns, almost all the photographs are taken from the second case. We have taken permission from the project manager before capturing the photographs. In this research, photographs are used as evidence of the physical artefacts used in each case, for example, Dependency board is an important artefact used in the second case to radiate the dependencies between teams.

3.3.6 Website

The researcher collected additional information about the case organisations and their products through their official website. Both the organisations have their own public-facing website, that provides information about their business goals, functions, structure and size. For the first case, the researcher gathered important information about their product, its features and a list of clients through their website. This information is used while discussing the case organisations and their detailed backgrounds.

3.3.7 Data collection process

This research followed Yin's four data collection principles to address design issues. Those are, a) using multiple sources of evidence, b) create and maintain a case study database, c) maintain a chain of evidence, and d) exercise care when using data from social media sources. All these principles direct the data collection activities for all kinds of data sources used in any case study research.

Table 3.3: List of coordination meetings observed in both cases

SL#	Case	What	Participants	Duration	Method & Tools
1	Case 01	Coordination meeting for Team-C	1 from NZ 1 from USA	30 minutes	Phone conference, Screenshare (occasionally)
2	Case 01	Coordination of integration across squads	4 from NZ (office) 2 from USA 1 from NZ (home)	30 minutes	Phone conference, (occasionally) screenshare, and Confluence
3	Case 01	Daily Standup of Team-R	5 from NZ (office) 1 from NZ (home)	10-15 minutes	Phone conference & Slack
4	Case 01	Testing review and Triage issues for Team-R	5 from NZ (office) 1 from NZ (home)	1 hour (but sometimes takes 15 minutes)	Phone conference, Skype for screenshare, Jira for item review
5	Case 01	QA coordination	1 from Japan 1 from India 4 from NZ (office) 1 from NZ (home)	30 minutes	Phone conference, Skype for screenshare, Jira for item review
6	Case 01	Support – Triage production issues	5 from India 1 from India (home) 3 from NZ (office) 1 from NZ (home)	1 hour	Phone conference, WebEx for screenshare, Jira for item review
7	Case 01	Prioritisation with Product Mgmt.	2 from NZ (office) 1 from NZ (home)	1 hour	Phone conference, WebEx for screenshare, Confluence & Jira for item review
8	Case 01	Tech Sync meeting (most complex coordination meeting)- Censored	6 from NZ (office) 2 from NZ (home) 5 from USA (multiple locations)	1 hour	Phone conference, WebEx for screenshare, Confluence & Jira for item review
9	Case 02	Daily Standup	9 from NZ 1 from Hawaii	10-15 mins	Video conference using Skype in phone, WebEx for screenshare
10	Case 02	Sprint Planning	8 from NZ 1 from Hawaii	30-45 minutes	Video conference using Skype in phone, Jira for Backlog sharing

These steps make the data collection process transparent and insightful for later process, i.e. data analysis and reporting outcomes. We have followed all the principles except the fourth one, since this research did not collect any data from social media (e.g. Facebook, twitter).

The first principle guides us to use more than one data sources while collecting data Which is being achieved by the triangulation of the data sources. To follow the second principle, the researcher has created and maintained a case study database. The purpose of this principle is to organise the data collection and documentation process. We have electronically created a folder for each case and all the data collected from each source for that case are stored under that folder. We have organised the data into subfolders based on their source, e.g. interview folder to store interview data, observation folder to store observation notes, photos folder for storing all the photographs. All the raw data (interview recordings, observation notes) and transcribed data are kept in a separate folder to avoid data corruption. We have created a data analysis subfolder to store all sorts of analysis findings for each case, such as excel files, different types of diagrams, and tables. The case study database has been stored in the University's Cloud storage (OneDrive) and only the research team (the researcher, and two supervisors) has permission to view and edit the files to ensure data protection. Hard copies of the interview and observation notes are scanned and stored in respective folders in online storage. All these data will be stored in the cloud as per University's data storage policy.

Maintaining a chain of evidence is essential to increase the construct validity in case study research (Yin, 2018). This principle guides the transformation of any raw data into valuable information by maintaining links (aka chains). If presented properly, this chain of evidence validates that the study findings are logically linked to the data collected for this research. Failing to maintain or establish the chain will eventually question the study findings and lose the quality of the case study. The following

diagram from Yin's (2018) book shows the steps of maintaining a chain of evidence (see Figure 3.5).

From the top, all the research findings are directly linked to the stipulated objects from the case study databases. These findings are the resultant of the analysis performed on the data collected from both cases. We have followed a structured data analysis process using a framework of analysis. The details of the data analysis process are discussed in the next section. Whenever required, excerpts from actual data are included in the findings to show their evidentiary source from the case study database. Sometimes the excerpts are exactly quoted, some other times, they are rephrased to make it meaningful for the readers. For example, we have used the following direct quotation from one of the interview transcripts to present a key dependency that has potential to raise dependency issues.

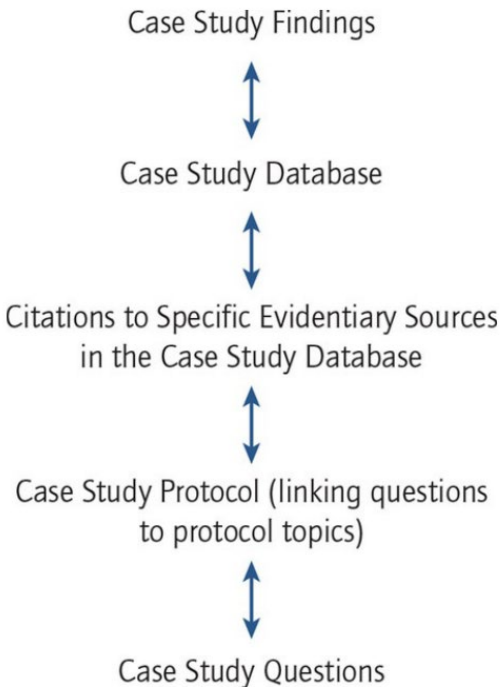

Figure 3.5: Steps to maintain chain of evidence (Yin, 2018)

[PSM1] says, "…*you're trying to do some feature that you haven't talked to us about, its 2 or 3 weeks and you need our help. so it is not going to deliver because of us*".

This piece of information is linked to the question asked to the participant about '*what sorts of information they need to exchange to coordinate their work process'*. The response to that question indicated that one team depends on another team's work (i.e. work dependency), so they need to share their requirements with other teams. But the problem arises, when the dependency is not identified earlier, or not notified or exchanged until it is too late. Such kind of situations has the potential to block work deliveries and create delays.

This example shows how one piece of information is collected (through interview questions), stored in a case study database (folders and subfolders for case data) and, finally, cited in the report to present a key dependency. This key dependency is identified to answer RQ1.1, i.e. what are the key dependencies in the DASD context? This chain of evidence has been maintained throughout the case study process to ensure the quality of the data collection and its findings.

3.4 Data Analysis Approach

Data analysis is an iterative process where the researcher attempts to analyse a large amount of data to make sense of them. The sense-making could be in any form, such as patterns, themes, causal relationships, hypotheses, and theories. In a positivist approach, the researcher wants to evaluate their hypothesis using the data to validate their knowledge. Whereas, in a more constructivist approach, the researcher efforts to interpret the data by analysing their causalities to formulate a hypothesis. Following the philosophical beliefs, this research applied a mix of data analysis techniques in two phases. In the first phase, the researcher analysed each independent case separately. In the second phase, cross-case analysis is performed to compare and contrast the findings between cases and finally, develop a theoretical framework of coordination in DASD context. However, before performing actual data analysis, the data are prepared for subsequent analysis process.

3.4.1 Data preparation

As mentioned, semi-structured interviews were the primary data source for this study. Though our initial plan was to record all the interviews, it was not possible for the second case. All the interviews in the second case were done in the team's workspace which was very noisy because of the open office plan. Both the interviewers (the researcher and one supervisor) took notes during the interview and verified them afterwards. The data preparation process started soon after the interview recordings and notes are available. All the audio recordings have been transcribed by the researcher and reviewed a couple of times by the other interviewer (one of the supervisors) to confirm the accuracy. After completing the process, all the transcripts were loaded into NVivo™ (2022) tool for the later analysis process.

After each interview, the researcher gathered all the field notes (interview and observation) and checked them for any confusion and consulted with the second interviewer to sort them out. All the field notes were scanned as pdf and saved in the case study database. The hardcopies were kept in a secure location as per the case study protocol. The photographs taken from each case were also uploaded under respective folders in the study database. Since the data collection and analysis were performed in parallel, the researcher started the first iteration of case data analysis soon after the completion of the data transcription process.

3.4.2 Within-case analysis

Each case data are analysed using Content Analysis techniques (Braun & Clarke, 2006; Fereday & Muir-Cochrane, 2006a), followed by the activity analysis and coordination analysis. Content analysis techniques are applied to locate the patterns and categorisation of themes from the data which supports the concept-building process. The software development activities are analysed using the activity-based framework (Korpela et al., 2002) for describing the elements involved in the software development process which is useful for cross-case comparison. To get a deeper

insight into the coordination, this study has performed the analysis of the coordination activities using a priori- developed framework presented in the following sub-section.

The content analysis technique is a popular qualitative data analysis technique. The benefit of using content analysis is to identify patterns and themes inherent in the collected data. These patterns and themes are used to answer the research questions and develop high-level concepts. This research has employed both inductive and deductive content analysis at the latent level (Braun & Clarke, 2006) to understand the underlying ideas, assumptions and conceptualization of the key dependencies in DASD. Table 3.4 shows the steps followed while doing the content analysis indicating the inductive and deductive approaches used in that step.

The content analysis activities are performed in multiple rounds. In the first iteration, the researcher performed the coding of all the interview transcripts to gather preliminary ideas. In the next iteration, the primary supervisor followed the same steps on a subset of transcripts and coded the scripts to identify dependencies and related concepts. And in final iteration, the researcher and the two supervisors discussed the coding to reach a consensus about the coding process. Based on the consensus, the primary researcher continued the rest of the analysis process to generate the high-level themes which is again reviewed by the supervisors and modified to report the findings. We have followed the same process for both cases to address validity and reliability concerns. Further data analysis is performed using our '*framework of analysis*' to examine each dependency, its impact and coordination effectiveness.

Table 3.4: Content Analysis Steps (Braun & Clarke, 2006)

Steps	Process
1. Familiarise with the data	Transcription of data and reading the data (iterations) and note or underline initial ideas related to dependencies, related issues, mechanisms, challenges (deductive approach)
2. Generating initial codes	Coding interesting features of the data (inductive) Collating data relevant to each code (inductive & deductive)
3. Categorising and creating themes	Categorise relevant data under potential theme (inductive & deductive) Gathering all data relevant to each potential theme

4. Review themes	Checking if the themes work in relation to the coded extracts (Level 1) and the entire data set (Level 2), Generating a thematic 'map' of the analysis.
5. Creating high level themes	Ongoing analysis to refine the specifics of each theme, Generating clear definitions and names for each theme (inductive & deductive)
6. Producing report	Finalising the analysis, Creating relations to the research questions and literature, Produce scholarly report of the analysis

3.4.2.1 Activity-based framework

The Activity-based analysis process (termed as activity-based framework) is founded upon the popular 'Activity theory' (Engeström, 1987) to understand the 'real-life work activity in context' (Korpela et al., 2002). According to this theory, an activity is a collective approach of a number of elements intending to transform a shared object into an outcome. While achieving the desired outcome, all the elements that are part of the joint work should, to some extent, fit together. It means that all the elements need to engage in a certain way to achieve a jointly produced outcome. Since the coordination in software development bears similar characteristics, therefore, it is plausible to use this framework to analyse the activity and its constituent elements to understand their coordination process.

The activity-based framework is useful to analyse the IS software development as presented by (Korpela et al., 2002). Nonetheless, it has been applied to understand software development activities on different scales, from small to large to globally distributed contexts (De Souza & Redmiles, 2003; Tell & Babar, 2012). This framework depicts the actors and their organisation, and how they work, interact, and coordinate themselves within the organisational boundary to accomplish a shared goal which makes it useful to apply in this case analysis.

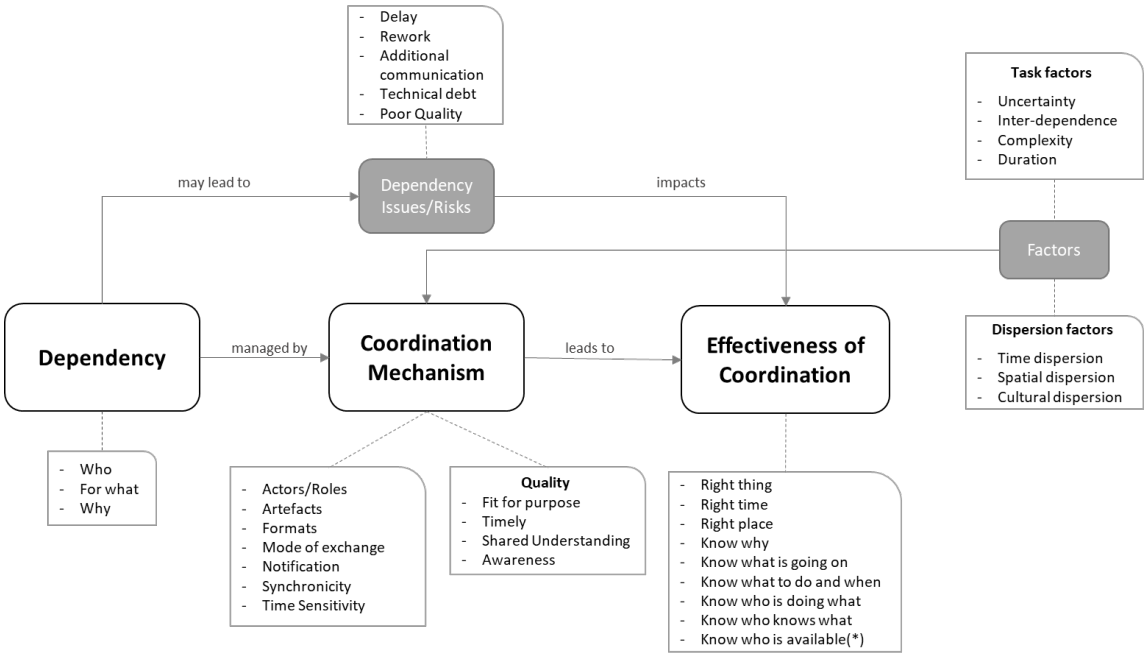

Figure 3.6: The Framework to analyse coordination in the DASD context

3.4.2.2 Framework of analysis

For analysing the coordination process of each case, we have developed a framework based on the preliminary model of coordination presented in Figure 2.6 under Chapter 2. The purpose of this framework is to analyse the data in a structured way to increase the repeatability of the analysis process. Likewise, this framework will assist in presenting the findings associated with the research questions and synthesize the concepts of the final model. This framework, presented in Figure 3.6, specifies the individual components that will be explored while analysing the coordination process in each case.

Dependency

The Coordination Theory (Malone & Crowston, 1994) defines coordination as "*managing dependencies between activities*". According to this definition, there are some components (*who*) that depends on another component (*for what*) to accomplish any activity and there is a cause (*why*) that urges the components to coordinate themselves. To analyse the coordination, we need to understand the dependency structure by answering the following questions:

- ***Who*** are dependent? (e.g. developers, testers, parts of the software etc.)
- ***For What*** they are dependent? (e.g. Knowledge, process or resource etc.)
- ***Why*** they are dependent? (e.g. to solve a bug, task prioritisation, develop shared understanding of the epics or user stories etc.)

The answer to these questions helps us to identify the dependencies in the investigating process. This list of dependencies will be examined to identify the '*key*' ones that are potential to create *dependency issues*. Dependency issues can be described by *delay, misunderstanding, rework, additional communication, technical debt,* and *poor-quality* output. All these dependency issues can be grouped as a form of '*Delay*' because they somehow lead to some sort of delay. For example, when a task-in-action depends on relevant information (dependency), the required information should be communicated (mechanism) in a way that creates a shared understanding. If the information is not properly communicated, it may create misunderstanding resulting a wrong output from the task. Whenever such situation occurs, the task needs to be reworked which will create delay in delivering the output.

Coordination Mechanism

Likewise, what techniques or activities are followed to address those dependencies also need to be investigated in a structured way. Coordination activities or techniques referred as mechanism have different types and patterns. A single mechanism can be applicable to manage multiple types of dependencies, but it is not guaranteed that the same mechanism can effectively manage all of them (Espinosa, Lerch, et al., 2002). Besides, the applicability and efficiency of coordination mechanisms are not unique and straight-forward, it is important to explore the characteristics of different coordination mechanisms that has been applied in any scenario. Amalgamating the researcher's theoretical understanding and professional experience, the following list of characteristics are identified that can better describe the coordination mechanisms:

- Actors/Roles involved
- Artefacts shared

- Format (Written, Verbal, Cognitive)
- Mode of exchange (Personal either group or individual, Impersonal)
- Notification Mode
- Synchronicity (Synchronous, Asynchronous)
- Time Sensitivity

It is worth to be mentioned that not all characteristics are appropriate for each mechanism, rather it is a generic form to cover the majority of them. While analysing the mechanisms, it is important to measure the quality of the mechanism. Based on our preliminary analysis findings, it was identified that quality of any mechanism can be described by several distinctive features, such as *Fit for purpose*, *Timely*, *Shared Understanding*, and *Awareness*. For instance, if any task-in-action depends on the output of another task (i.e. producer/consumer dependency (Malone & Crowston, 1994), the mechanism needs to ensure that the output is fit for purpose. Otherwise, the dependency might be managed but not well-managed. Similarly, the output should be available at the right time to be used by the consumer task. Shared understanding is another quality requirement for mechanism used for knowledge dependencies. A qualified mechanism should not only ensure the timely communication, but also need to ensure a shared understanding. Otherwise, it may create dependency issues such as rework or poor-quality work output. Finally, any mechanism applied to manage a certain type of dependency should foster awareness of the activities and its effects on the other side, particularly in distributed software development. For example, Team A depends on work output from Team B, Team A needs to communicate the work requirements so that Team B can identify ‘what needs to be done’ and ‘when’. The mechanism applied for this dependency is inter-team communication by sharing the work requirements in a certain format. The mechanism would be considered as high-quality if Team B can develop the awareness of ‘what needs to be done’ and ‘when’. We have applied these requirements to measure the quality of the identified coordination mechanisms.

The choice and application of coordination mechanisms could be affected by two factors: *Dispersion* and *Task*. Dispersion factors contains time, spatial and cultural factor. Task factors may relate to Uncertainty, degree of inter-dependence, complexity and duration. While analysing the coordination mechanisms, these factors have been considered to investigate their impacts on the quality of the mechanisms.

Effectiveness of Coordination

As discussed in the literature review chapter, coordination effectiveness characterises the extent to which key dependencies are effectively managed (Espinosa et al., 2007b). To evaluate coordination effectiveness, Strode et al.'s (D. E. Strode et al., 2011) framework have been adapted for the DASD context as presented in the Figure 2.8 (Chapter 2). The framework contains explicit and implicit components. Explicit components are 'right thing', 'right place' and 'right time', whereas implicit components are 'Know why', 'know what is going on and when', 'know what to do and when', 'know who is doing what', 'know who knows what' and 'know who is available'. While analysing the key dependencies for each case, we have used these components to identify how effectively the applied mechanisms managed those dependencies. Since the poorly managed dependencies leads to *dependency issues*, it has direct impact the *coordination effectiveness*. We have used this relationship to identify how dependency issues impact the components of coordination effectiveness.

3.4.3 Cross-case analysis

The cross-case analysis is a popular technique applied in case study research to achieve robustness, generalisability and applicability of the study findings (Yin, 2002). In this approach, multiple similar cases are selected based on the particular phenomenon under investigation and the findings of the analysis are compared and contrasted for essential elements and components across cases (Khan & VanWynsberghe, 2008). The cross-case analysis outcomes are theorised to build a conceptualisation of the phenomenon that would offer a greater understanding of the phenomena. In this research, we have performed a cross-case analysis of the two

selected cases from two interrelated perspectives. Firstly, we have compared the cases based on their contextual elements following Korpela et al.'s (2002) activity framework and other elements. Secondly, we have compared the coordination aspects including dependency coordination mechanisms, coordination effectiveness and associated issues.

The cross-case analysis findings have ratified each of the steps of the theory development process discussed in Chapter 7. During the cross-case analysis, we have focused on the types of dependencies and their associated issues that possess high risks. The results are used to develop a generalised view of the key dependencies and their associated risks in the DASD context. Then, a list of coordination mechanisms has been populated that can support the key dependencies in a similar context to reduce their impacts. The final list of components of coordination effectiveness has been portrayed based on the case evidence. Finally, the relationships between the key concepts of coordination, *Dependency*, *Coordination mechanisms* and *Coordination Effectiveness* are drawn to complete the theory.

3.5 Ethical Considerations

This research findings are dependent on the interview, observations, and documents relating to personal and organizational knowledge which require human participation. As per the AUTEC's instruction, all research work involving human subjects requires to receive approval for ensuring the privacy, safety, health, social sensitivities, and welfare of human participants. Therefore, this research followed the formal procedures to get Ethics approval from AUTEC before making any formal contact with any of the case organisations.

The research team has ensured that all data related to this research and their participants are considered 'Confidential' and a confidentiality agreement has been signed beforehand of the collection of data. All the data collected from each case has been stored in a secure location as committed to AUTEC. Both the interviewees signed a Non-Disclosure Agreement (NDA) form for each case organisation and are

responsible to maintain the confidentiality of sensitive organisational data and the anonymity of each participant. The research team acquired signed consent from the management representative of each case to collect data from participants and other sources. Each participant is provided with a participant information sheet in advance and briefed about the purpose of the research at the time of the interview. Each participant signed the participant consent form prior to the interview. All the study participants are well informed about their rights and protocols to assure their privacy and alleviate discomfort during and after the interview. A draft containing the case description and findings has been shared with case representatives to confirm that confidentiality and anonymity are maintained accurately.

Table 3.5: Aspects of Quality Assessment applied in the study

Aspects of Quality Assessment	Definition
Construct Validity	Establishing correct operational measures for the concepts being studied
Internal Validity	Establishing a causal relationship, whereby certain conditions are shown to lead to other conditions. as distinguished from spurious relationships
External Validity	Establishing the domain to which a study's findings can be generalized
Reliability	Demonstrating that the operations of a study can be repeated with the same results

3.6 Validity, Reliability, Rigor & Relevance

The quality of any research depends on how well the research design, data collection and data analysis process meet the validity and reliability standards. Though there are no definite standards established for studies using qualitative methods (Miles & Huberman, 1994a; Wilson & Creswell, 1996; Yin, 2002), Yin (2002) has suggested a framework for case study research. The framework includes four aspects of quality assessment, namely, *Construct Validity*, *Internal Validity*, *External validity,* and *Reliability*. He further guided case study researchers by presenting some tactics to deal with each assessment. Following his guidelines, this exploratory case study research addressed all these aspects to enhance the validity and reliability of the research outcomes. The definition of these four aspects are outlined in Table 3.5 and the

application of these aspects in this study is discussed in chapter 8 (section 8.4) while evaluating the research.

Rigor and Relevance

Dubé & Paré (2003) established a list of attributes to assess the rigor in any positivist case study research. Since this case study follows a mix of positivist and interpretivist approaches, these attributes are valuable while evaluating methodological rigor in activities performed in the *research design*, *data collection* and *data analysis* phase. The following Table 3.5 presents the list of attributes that are being applied in this research and is further discussed in Chapter 8 (section 8.3). Since this study follows a multiple-case exploratory research design, only relevant attributes to this approach are included in this table. The table contains three divisions to cover each specified area and each division contains a discussion about how rigor has been achieved in that area. While evaluating from an interpretivist perspective, (Fossey et al., 2002) proposed five distinctive criteria: *authenticity*, *coherence*, *reciprocity*, *typicality,* and *permeability*, that conform to the well-accepted principles for evaluating interpretive studies in IS research (Klein & Myers, 1999). The details of each criterion and their application are presented in the discussion chapter (section 8.3 in Chapter 8) while evaluating the quality of the interpretations to produce a convincing explanation of the phenomena of investigation

Research relevance is an important factor to improve the acceptability of the research to the practitioner audience. Relevance refers to the fact that the research outcome should be relevant to practice and can be directly applied to professional practice (Robey & Markus, 1998). In IS research, there can be three dimensions of relevance: *importance*, *accessibility*, and *applicability* (Rosemann & Vessey, 2008). A research is considered to be relevant if the research can cover these three dimensions. From these perspectives, this research is *important* as it aims to understand the coordination process in a trending DASD context and provide plausible suggestions to improve the state of coordination. This research can be considered *accessible* since we have tried

simple stylistic attributes to make it appealing to non-academic readers (Robey & Markus, 1998). For example, we have used diagrams to portray delay scenarios that are easy to understand. We have used tables to summarise the key case findings that will provide a snapshot of the overall findings. Additionally, this researcher follows a clear and simple writing pattern that is readable for non-academic readers.

Finally, the research meets *applicability* concerns by suggesting ways to improve the coordination states. We have thoroughly discussed the key dependencies, and their associated issues and then presented recommendations (whenever needed) to improve the condition by using alternate mechanisms. These recommendations can be applied in similar scenarios (suitability) that deemed this research applicable to practice.

Table 3.5: Attributes used for Assessing Rigor in Exploratory IS research (Dube & Pare, 2003)

Attribute	Purpose
Research Design	
Clear research question	• Links the study's theoretical and practical contributions • Addresses reliability concerns (Miles & Huberman, 1994a)
A priori specification of constructs	• Shapes the initial research design showing constructs of interest
Clean theoretical slate	• Reduce research bias while analysis • Limits researcher's interference on limiting the findings
Multiple-case design	• Produce robust and more generalised results than single case • Enable cross-examination of case results to maximise theoretical coverage
Replication logic in multiple-case design	• Case selection based on substantive significance or theoretical relevance • Addresses external validity concerns (Yin, 2002)
Unit of analysis	• Define a clear boundary within the case • Guide the data collection and applicability of theoretical contributions
Context of the case study	• Describe the contextual elements in which the research was conducted • Enhance the credibility
Data Collection	
Elucidation of the data collection process	• Elucidate the data collection process including o what data has been collected o how those data are collected o what sources are used and why o how collected data contributes to the findings • Enhance reliability and validity (Benbasat et al., 1987)
Multiple data collection methods	• Collect evidence from multiple sources to provide rich insight of the investigating phenomenon • Ensure construct validity (Yin, 2002)

Data triangulation	• Converge evidence from multiple data sources to strengthen research outcomes • Enhance construct validity (Yin, 2002)
Case study protocol	• Documenting details of the research procedures • Ensure the research process can be repeated by other investigators and reach to similar conclusion • Enhance research validity and reliability (Yin, 2002)
Case study database	• Organise and manage study documents • Enhance research reliability (Yin, 2002)
Data Analysis	
Elucidation of data analysis process	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability and reduce researcher bias
Field notes	• Supporting evidence for other data collection methods
Coding and reliability check	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability and reduce researcher bias
Data displays	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability
Logical chain of evidence	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability and construct validity (Yin, 2002)
Explanation building	• Demonstrate how data are interpreted to textually explain the phenomenon
Searching for cross-case patterns	• Look for similarities and differences between cases using categories and dimensions
Quotes (evidence)	• Supporting evidence to develop theoretical concepts and show their relationships
Comparison with extant literature	• Strengthen study findings by showing supportive or contradictory literatures

3.7 Summary

This study has selected an exploratory case study research method. This chapter has presented the influence of the philosophical perspective of the researcher followed by the contextual influence on the methodology selection. This research has adopted a mix of positivist and interpretivist stances to answer the key research questions. The research design process is presented with a case study protocol, followed by implementation details including the selection of the case and participants, data collection and data analysis. All the ethical considerations and how this research has addressed them are discussed. The validity and reliability concerns are first described, followed by a discussion of how this study addresses those concerns. A detailed discussion about the process followed to ensure the rigor and relevance of this study are presented. Finally, the elements of theory for this research are briefed to highlight the research goals, and the outcomes are presented in Chapter 7. The next chapter will

present the findings from the within-case analysis, followed by a cross-case comparison with a related discussion.

Chapter 4 : Case 1 Findings

This chapter presents the analysis and findings of the first case, a mature product development project involving globally distributed teams and work streams. The product was being used by a large number of clients spread across the countries, which was supported and enhanced by teams working from five different time zones. Due to the nature of the product and the software development process, a large number of critical dependencies were identified involving local and global teams in which coordination anomalies and challenges were analysed.

An overview of the project background and the DASD process are presented in sections 4.1 and 4.2. This discussion provides a better understanding of the project structure and how the software development activities are conducted. A discussion of the project activity analysis process and its findings are presented in section 4.3. The coordination analysis process, mainly focusing on the dependencies and their management, is discussed in section 4.4. Then follows the findings of the analysis in section 4.5, with a detailed discussion about the results for better interpretation for the readers presented in section 4.6. The chapter concludes with a summary presented in section 4.7, providing the foundation for the cross-case analysis presented in Chapter 6.

4.1 Project Context

The first case, code-named ‘Pigeon’, is the key project for a US-based multinational company that focuses on developing software systems for managing health informatics. This project is considered a DASD project since there are globally distributed teams and stakeholders involved in two work streams and adopted agile practices in their software development process; thus deemed suitable for this research. The case organisation is a prominent healthcare service provider that regularly collects and stores complex healthcare-related data into the cloud database

and provides advanced analytics through innovative technology solutions to support better decision-making and customer service. The key product developed in Pigeon delivers unique and valuable insights into diseases, treatments, costs, and outcomes to their clients. The organisation is globally spread in more than 100 countries involving 50,000 employees to help clients run their operations more efficiently.

The case Pigeon involves the ongoing development of a mature product that has been used by its clients on a regular basis to get critical, real-world disease and treatment insights. The product has a front-end service that utilises the strong backend infrastructure to provide critical insights for healthcare-related services. This product can be classified as critical from both the organisational and client perspectives, as it drives the organisation's business and enables it to provide end-to-end service to its clients. Likewise, the clients depend on this solution for their day-to-day services to access sensitive healthcare data and analytical insights to support their customers. An overview of the Pigeon case is presented in Table 4.1.

The product development activities are primarily divided into two main streams of work, existing features enhancement, and bug fixing and support. Both work streams involve six teams in total that are globally distributed in NZ, Europe and the US. Four development teams are located in NZ, code-named as Team R, Team FT, Team RT, and Team M. Another two teams are USA-based, code-named Team FW and Team C. Quality Assurance (QA) and Client Support teams are external to the organisation and distributed in different parts of Europe and Asia. The development teams are module-based and specialising in specific parts of the product, as described in the following paragraphs. Several of the product's core modules are strongly inter-connected with each other creating high interdependencies between the globally distributed development teams. The teams' organisation and their inter-relationships due to the product architecture are portrayed in Figure 4.1, which helps to provide a clear understanding of the teams and their contributions to the product.

Table 4.1: Overview of Pigeon case Project Context

Case Name	**Pigeon**	
Organisation	Organisation Type	Commercial
	Organisation Size	Large (>250 employees)
	Business activity	Healthcare data analytics and client support
	Market	Global (distributed over 100 countries)
Project	Project Status	Ongoing
	Project Purpose	Enhance the product features, Bug fixing & Client support
	Project Criticality	Medium
	Project Stakeholders	Global POs, Proxy PO, Development teams, QA teams, Support teams
Product	Product Size	Medium
	Product Status	Mature
Team	Team Size	7 (Team R) 6 (Team RT)
	Interviews	Seven from Team R One from another Team RT
Development Method	Agile Approach	Scrum (primarily) XP (occasionally)

All but one of the participants of this study are members of Team R, that is specialised in data replication services. A list of all the interview participants from Team R and their roles is presented in Table 4.2. The replication service is one of the core back-end services of the product that assesses which bit of data needs to go to which remote machines like tablets or iPads. This service ensures the optimised data replication facility so that only the necessary amount of data is replicated or copied to the user's machine to avoid copying the whole database simultaneously to save device space and bandwidth.

Table 4.2: List of study participants, their roles, and representative codes

Participant No	Participants Role	Code	Team
1	Development Manager	PDM	Team R
2	Developer 01	PDV1	Team R
3	Developer 02 (TL)	PDV2	Team R
4	Developer 03	PDV3	Team R
5	Developer 04	PDV4	Team R
6	Tester	PTS	Team R
7	Scrum Master 01	PSM1	Team R
8	Scrum Master 02	PSM2	Team RT
9	Product Manager	PPM	Team R

Team R is located in NZ and has seven members: one Development Manager (DM), four developers, one tester and a Scrum Master (SM). The tester is a regular team member who performs the acceptance tests for all the development works before submitting them to the Product or integration tester. The Product/Integration testers and Quality Assurance personnel are referred to as QA. There are two groups of QA, one for integration testing and the other for QA. All the QA team members are *global* and located in USA, India and Japan. Initially, QAs were shared by all the teams, but later this was changed to each team having their own QA to enhance coordination and performance.

Team C has three members who are located in different parts of the USA: one team lead, one developer and one integration tester. Team C is responsible for ensuring the connection establishment and data transactions within the product features. Team C and Team R have high inter-functional overlapping in which multiple module features are calling services from each other. Due to this multiplex relationship, both teams need to maintain intense collaboration with each other. For example, Team R needs to coordinate with Team C to establish a connection before replicating any data from the database to any target device. Since Team R and Team C are located in different time zones around the globe, all their dependencies are coordinated via a formal channel facilitated by the Scrum Master, who is communal for these two feature teams. The

Scrum Master corresponds with each team separately for its internal activities and conducts joint coordination sessions weekly with both teams. Due to the time differences, there are few opportunities for these teams for informal communication to coordinate their dependencies.

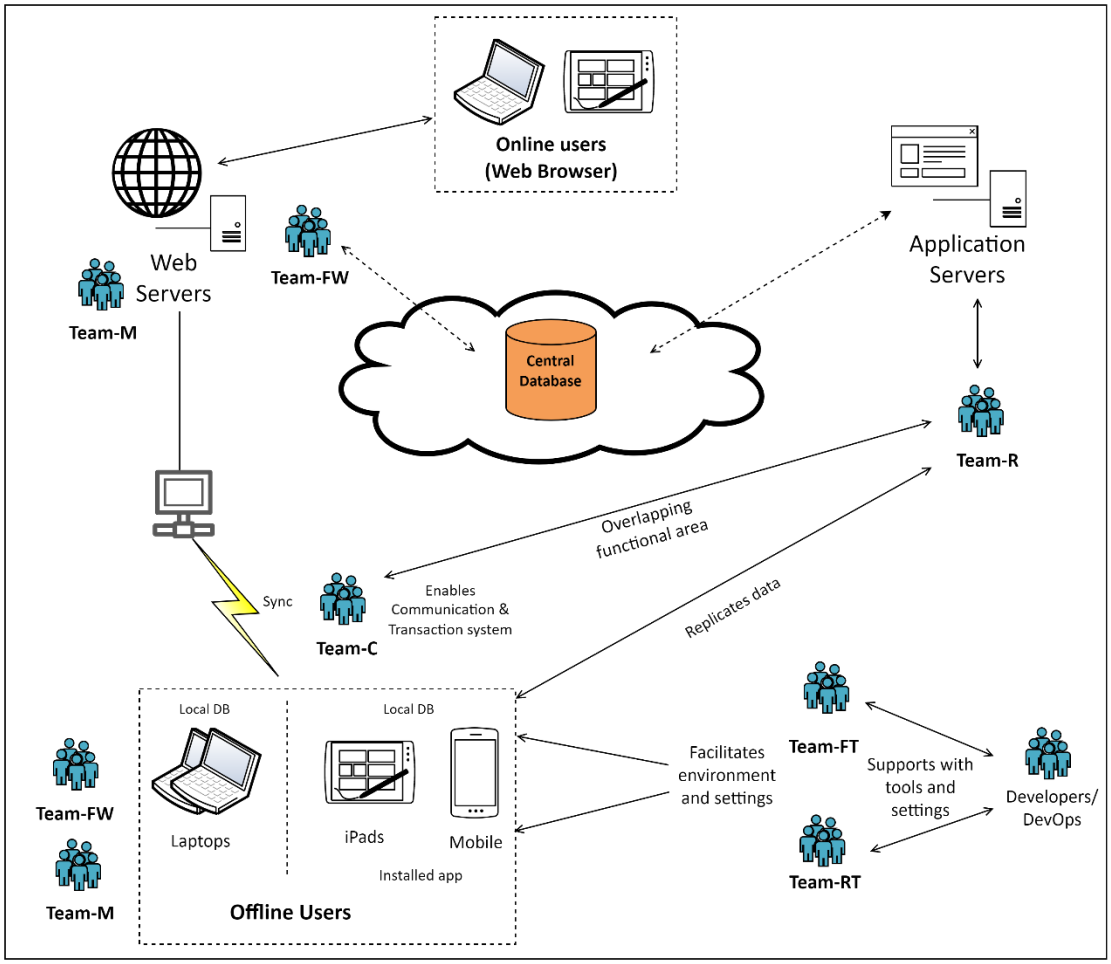

Figure 4.1: Product architecture and different team's involvement in the product

Team FT, an NZ-based team, is responsible for managing the admin tools and provides all kinds of environmental settings (i.e. services, clusters, the database). This team also acts as a representative for the client teams to set up for their tenants, including their configuration settings for their apps, e.g. adding users, configuring how the users can interact and security settings etc. They also develop the UI for the front-end services to initiate the back-end jobs.

Team RT is also an NZ-based team that works with all sorts of build tools, development and DevOps tools, and configuration management (i.e. how does install, how to

authenticate etc.). Team M, based in NZ, specialising in integrating the web servers for offline and online services. Finally, Team FW is a US-based team ensuring data management for the front-end services. For understanding the relationships between teams and their functional areas in the product, a diagram of the system architecture is presented in Figure 4.1.

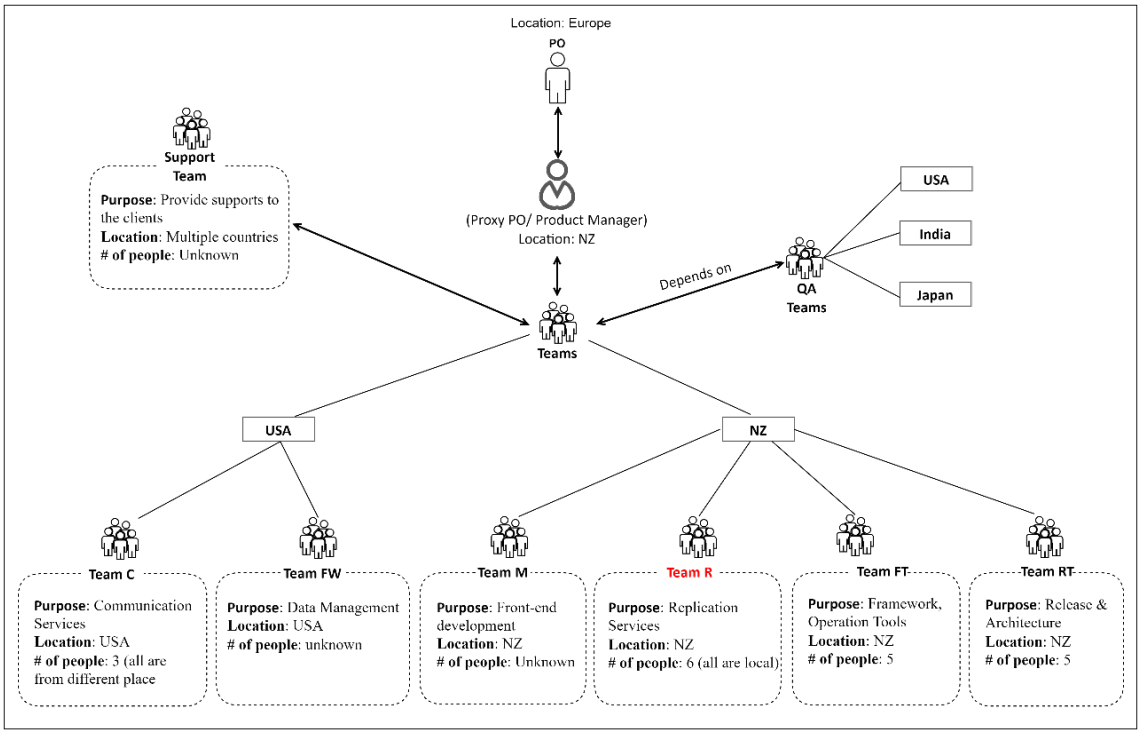

Figure 4.2:Teams organisation indicating their locations

All Product Owners (PO) for this product are located in different parts of Europe, and there is a liaison person in NZ who acts as Proxy PO. All the team's organisations and their structure, including the product owners**,** are presented in Figure 4.2. Regarding stakeholders, Global POs, clients and Proxy PO (local) are primary stakeholders for product requirements. and other teams (e.g. replication, deployment, client support) and QAs are the stakeholders for feature development and bug fixing.

This case represents a widely distributed vendor development workforce with highly inter-connected product features, and clients and development teams distributed globally. For successful product development and client support, all the global touchpoints need to collaborate in their day-to-day work. Successful coordination of this complex and distributed product development is necessary. An overview of the DASD

process of this project is presented in the next section, which highlights the coordination touchpoints and their relationships to better understand the coordination needs.

4.2 Overview of the DASD Process

Product feature enhancement and bug fixing are the two primary work streams consuming most of the work and coordination efforts. Each workstream consists of several interdependent activities involving co-located and distributed teams. All the teams involved in the process need to synchronise their work so that outputs can be integrated and released to the clients without any issues. While working in these activities, a significant amount of knowledge and expertise needs to be shared, which demands frequent communication among the parties. The teams' physical and temporal differences are being considered while managing this work and knowledge coordination. A coherent understanding of the process and work requirements would be crucial to identify the critical dependencies, and how they are currently managed, thus, helping to evaluate their effectiveness and associated challenges. The subsequent sections separately discuss the work processes under each stream that supplements the analysis process of this study.

Since the analysis process followed in this study focused on Team R's coordination efforts, a diagram showing all the top-level activities performed by Team R in the two main work streams is presented in Figure 4.3. The diagram is segmented into five main parts based on the generic software development activities, such as requirements gathering, requirement analysis, development and testing, and release. Part A & B relates to product feature enhancement activities discussed in section 4.2.1. Part D & E are specific to bug fixing, and support activities discussed in section 4.2.2. Part C relates to the core development and testing activities, which successively lead to a new release. While discussing each part of the development process, coordination touchpoints are identified, which provides the basis for the coordination analysis and discussion presented in the later part of this chapter.

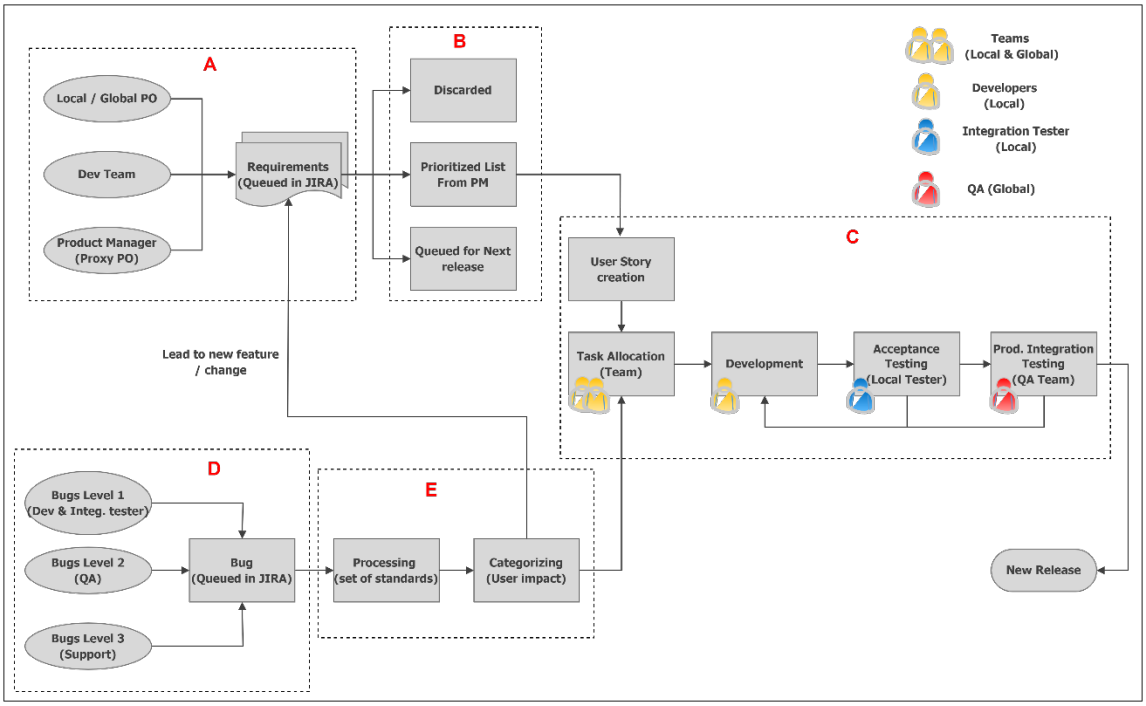

Figure 4.3: Overview of the Software Development Process

4.2.1 Product Enhancement (PE) Process

4.2.1.1 Requirement Gathering for PE workstream (Part-A)

In the product or feature enhancement process, global POs are the primary sources of new ideas and requirements. Since they are from different parts of Europe, the Proxy PO coordinates with them and manages the work requirements and prioritisations. While the proxy PO in NZ is the key contributor to the release requirements, other teams also contribute to the requirement gathering, and on approval, their ideas are also included in the release backlog. Therefore, the requirement gathering and prioritisation activities rely on the POs, Proxy PO and the development team representatives. An Issue tracking system, i.e. Jira is used to record and manage all the release requirements.

Additionally, the reported bugs that are complex and have high user impacts and may require multiple teams' involvement are also an important source of requirements, as indicated in part E. In Team R, the DM takes the decision for such bugs and pushes them to the requirements list in Jira for collaborative decision-making. As illustrated in diagram 4.3, all the preliminary requirement ideas from all these sources are recorded

in Jira as indicated in Part A, which is further discussed and approved to be the release goals as discussed in part B.

4.2.1.2 Requirement Analysis for PE workstream (Part-B)

In this segment, all the requirements listed in Jira go through the analysis and decision-making process. The POs, both local and global, and development teams are responsible to finalise the ideas that will go through the development process. All selected enhancement ideas are analysed according to the feature and expertise needed. Based on this analysis, teams with matching expertise are assigned to release requirements. Requirements confirmed for release are prioritised in this phase which might be changed at a later stage of the development. The prioritisation decision includes what features to be included in the current release or queued for future releases. Coordination of this decision making, and work distribution activities are performed in a group meeting named '*Release Planning*' chaired by the PM. The meeting is pre-scheduled and conducted over the conference call to connect all the distributed parties from different time zones.

Product feature requirements related to a specific module are managed by the team specialised in that module. The Team leads from each specialised development team participate in the release planning meeting. They are involved throughout the analysis and decision-making process to have a mutual understanding of the release requirements. The team leads of assigned teams are now responsible for continuing the further development process along with other team members to meet the release deadlines.

4.2.1.3 Development and Testing for both streams (Part-C)

All sorts of coding and testing activities for the two streams are performed in this segment of the process. The development teams follow different sprint cycle duration for their convenience, for example, Team R initially followed three weeks sprint which later on switched to two weeks sprint to synchronise with the release cycles better. The

new product feature releases are scheduled around every four months, which includes all the sprint releases and major bugs fixed within that time.

The breakdown of activities performed in the core development process is shown in Figure 4.4. It starts with the '*Sprint planning'* activity, which is a group session where all the Team R members participate in setting up the sprint actions. Team members collectively decide the sprint duration, sprint goals and the way to achieve the goals. The planning starts with a list of user stories that are created based on the release goals. Team R depends on the Senior Development Manager to convert the release requirements into user stories. Team members then participate in the '*planning poker*' session in which the tasks are broken down to estimate the time required to complete the story. Based on the estimation, the team decides on the stories to be included in the next sprint. Sprint planning sessions are separate meetings, but occasionally they are combined with the '*backlog refinement'* session. Usually, backlog refinement sessions are used to review the user stories and clarify any misunderstandings. Team R records and maintains the sprint stories in Jira termed as '*Sprint Backlog*'. SM coordinates the tasks related to sprint backlog items and the '*self-task assignment*' during the sprint planning.

The sprint planning session is another coordination touchpoint that helps to surface dependencies and risks that might impact the sprint activities. The dependencies might be between the tasks or might be between teams. Team R performs a stakeholder analysis at this stage to identify the potential involvement of other development teams. For Team R, this analysis is a way of identifying potential work or knowledge dependency on other specialised teams that might be involved in the upcoming sprint. For example, if Team R identifies that any sprint work item requires a connection establishment, then Team C needs to be involved as they are responsible for that kind of task. However, the data indicates that not all the development teams follow the same process, and as a result, several coordination issues occurred as discussed in the findings section.

Team R maintains an electronic '*Squad board*' using Jira for task allocations and tracking work progress, which serves as an 'information radiator' for the team members (Buchan et al., 2019; Fröling, 2018). Each work item is recorded as Jira tickets and their status is visible to all team members. In terms of coordination, Team R conducts iterative sessions to reinforce the shared understanding of the sprint goals and the team's progress. the *'Daily standup'* meeting is a coordination point to share work updates and manage blockers. Team members use the *'Slack'* tool to report their daily work updates during the daily standup. Development-related issues and impediments are discussed further in a separate meeting to examine and respond, and if required, escalated to other development teams for their support. Team R members also conduct regular code review sessions during the sprint which helps them refine their understanding of '*what needs to be done*'. At the end of the development, completed Jira tickets are queued in the testing backlog for acceptance tests.

In terms of the testing, Team R has a tester in the team to perform all the acceptance tests for the completed backlog items. The tester is an essential part of the team and participates in all the meetings and ceremonies like other team members. Based on discussion and feedback about the user stories, the tester can start thinking about the possible test cases and prepare a '*test plan'*. The test plan is a graphical presentation that specifies the test coverage and action items are reported as bullet points.

Team R conducts regular '*test case review*' sessions to review and give feedback about the test cases and test plans. The tester also attends the developer's code review session to know '*what has been done*' in terms of coding and testing, so that, if possible, he can skip some test cases to save time. Tester maintains a separate '*Testing Squad board*' for testing activities which shows the progress of testing activities and their status. The team structure and software development practices indicate the tester is an integral part of the development team. However, the testing cycle (i.e. sprint) differs from the coding cycle which starts at the end of the coding sprint.

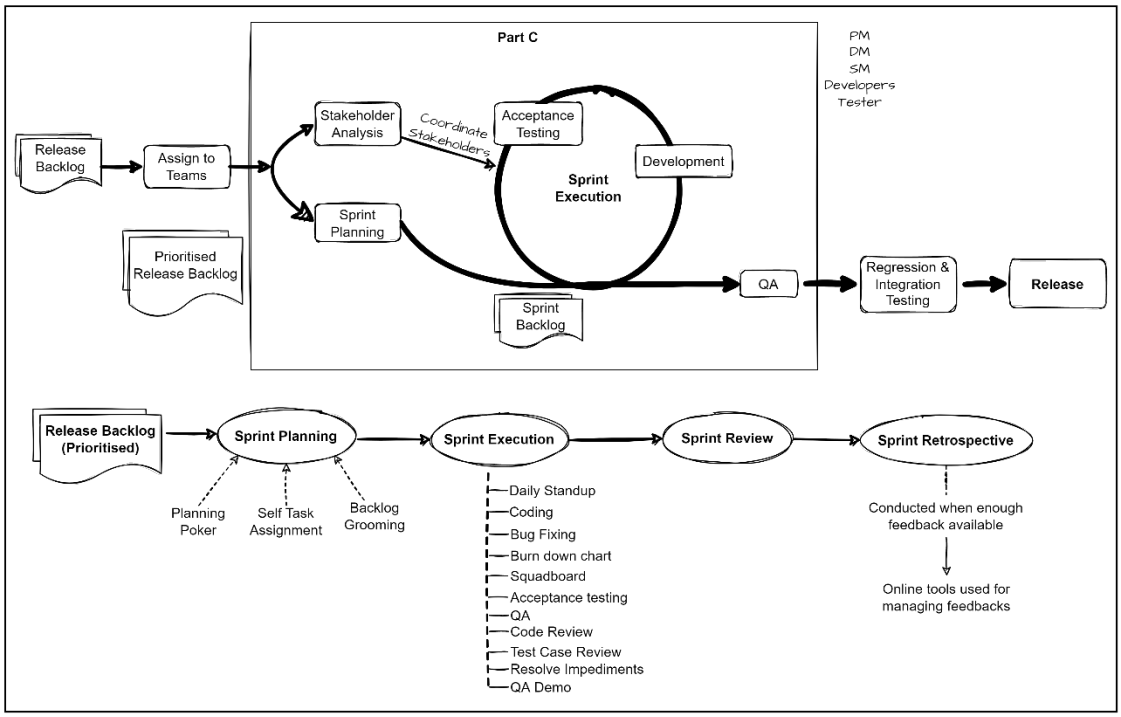

Figure 4.4: Agile SD Activities performed in Part-C

After completing the acceptance tests, the tickets are forwarded to the QA team for approval before releasing their work. Both the developers and tester need to exchange information with the QA regarding the tickets or any problem they are facing. Since the QA team members are located in India and Japan, email and comments on Jira tickets are the primary communication channel used between these parties. The QA members can access the tester's squad board and see the ticket details and comments specifically done for the QA. Besides, Team R attends a regular '*QA meeting*' every fortnight with the QA leads to answer their queries and resolve work issues. Other development team members also attend this meeting. Team R used to conduct '*Sprint review*' sessions after each sprint which is now altered by '*QA Demo*' for sharing the recent development outputs and how things work. All the failed tickets are registered as '*Bug*' in the system which needs to go through the bug fixing process.

Team R performs '*Retrospective*' sessions to evaluate and improve the team's sprint performance. The Scrum Master uses an 'online tool' to collect and share improvement ideas between team members. This meeting is an 'Ad-hoc' kind of meeting conducted

after every couple of sprints. Based on the discussion and feedback, team members work together to make plans to implement them in upcoming sprints.

4.2.2 Bug fixing & Support (B&S) Process

The Bug fixing and support process is another work stream that adds workloads for the team throughout the year. It is a continuous process that not only focuses on the reported bug resolution but also solves customer support issues. Bugs related to specific features or functionalities are handled by the respective teams who developed them. For Team R, the bug fixing activities require 50% of their work efforts. The current process for handling the bug-fixing workloads differs across teams. For example, while Team C integrates all the bug fixing requests with the new enhancement workflow, Team R handles bug and support activities separately from product enhancement. In Team R, the majority of the firefighting works are handled by the Development Manager (DM) without interrupting the development team's workflow. With his product knowledge expertise and development experience, the DM manages all the bug and support requests along with one or two developers who are assigned intermittently. The overall bug fixing process goes through several key phases (shown in Figure 4.5), those are *Bug identification or creation*, *reporting* (Part D)*,* and *reviewing and Categorising* (part E), which are discussed separately in the following subsections.

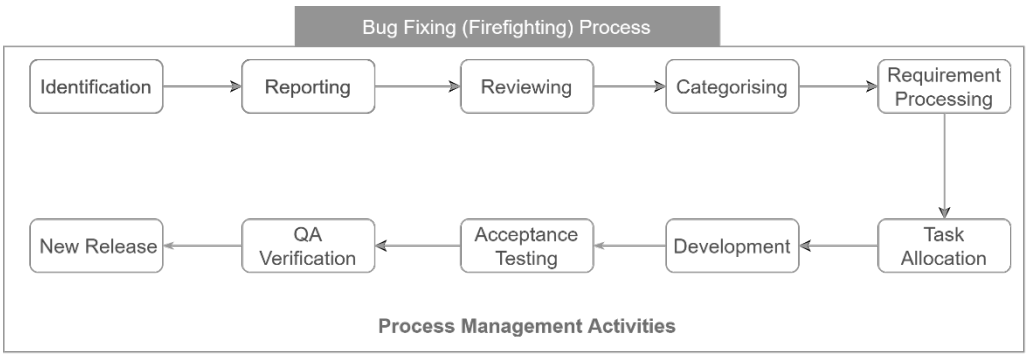

Figure 4.5: The activity flow of the bug fixing process

4.2.2.1 Requirement gathering for B&S workstream (Part-D)

All the incoming bugs are categorised under three levels (Level 1, 2 & 3) depending on their reporting sources and customer impacts. The bugs reported by the developers

and integration testers are categorised under Level 1, QA reported bugs under Level 2, and customer support members reported bugs under Level 3. The benefit of using separate levels for the bugs is to support the prioritisation of the bugs based on their reported source. For instance, Level 3 bugs get the highest priority because of the potential customer impacts. Current sprint-related bugs, i.e. Level 1 bug gets the next priority and then the Level 2 bugs since in most cases this needs to be included in the next sprint.

The bug fixing process starts by *identifying* any potential bug by any of the teams involved. In most cases, the bugs are identified by the local tester or the QA team who is working offsite. Other parties, such as the support team or customers can also report any bug through different channels. Once the bug is identified, they need to be *reported* to the development team using the designated channel. If the bug is identified by the tester or the QA team, they need to report that by creating a new Jira ticket with details of the problem they are facing. In the case of the support team or clients, then they have to use an online tool called '*SMART'* that is specifically developed in-house for this purpose. Another communication channel commonly used for this purpose is *Email*. Any of the parties can directly report any bug by sending an email to any predefined or known point of contact, such as DM or any specific developer. Based on the level and priority, the DM reviews all the requests and either works or allocates them to the supporting team members which are discussed in the next section.

4.2.2.2 Requirement Analysis for B&S work stream (Part- E)

The next activities include bug processing and categorisation, which are handled by the DM. Once the reported bugs are reviewed and categorised, they are either recorded as potential improvement ideas (from Part E to Part A) or go through the development process as illustrated in Figure 4.3.

Whenever any bug is reported through any of the channels, the Development Manager *reviews* the bugs before it goes to the next phase. As mentioned earlier, the DM is the

central decision-making authority and technical resource to handle most of the bug review and development activities. He possesses sound technical and product-specific knowledge and experience to perform these activities which creates a central knowledge and work dependency structure. Therefore, other team members heavily rely on him for his expertise knowledge and decisions throughout the bug fixing process.

Next, the request goes through the categorising process to measure the importance of the issue based on the user impacts, i.e. impact analysis, which is primarily done by DM, but also involves POs (both local and global), PO Proxy (local), SM and development teams if the bug leads to product enhancement. In such cases, the rest of the process follows the steps mentioned earlier in terms of product enhancement.

Any Ad-hoc support cases are resolved by the firefighting team led by the DM. The subsequent development activities such as coding, testing, and QA are performed along with other sprints. The bug-fixing activities need to elicit the knowledge and expertise of other teams (some global) to debug and solve the issue. Scrum Master organises phone conferences using WebEx with respected teams to coordinate similar situations.

Currently, Team R responds within 24 hours of a ticket being raised and dedicates three days for issue resolution. This schedule does not align well with the timing of the new enhancements (two weeks sprint) and the release cycle (three times per year). The overall bug fixing process goes through several process management activities such as decision making, impact analysis, task synchronization between local and global teams (both local and global), work progress tracking and sharing status between the team and management authorities. Therefore, proper management of the development and managerial activities play a vital role in coordinating the entire bug fixing process effectively.

Taken together, both the work streams consist of a number of activities and knowledge sharing that involve people working from local and distributed sites. While performing these activities, they need to synchronise their work because of the input-output dependency (Malone & Crowston, 1994), and exchange knowledge (Razzak & Ahmed, 2014) and expertise (Faraj & Sproull, 2000). To manage effectively, appropriate mechanisms need to be applied accordingly to avoid coordination breakdowns (Cataldo & Herbsleb, 2013). The next sections illustrate the two-phased data analysis process and then report the findings of the analysis.

4.3 Activity Analysis & Findings

The software development activities of this case are analysed in two phases. At first, the activities are analysed following the guidelines suggested by (Korpela et al., 2002). This guideline provides a structured way of describing the elements of the software development activity that establishes a common ground to compare and contrast the activity elements of the two cases of this research. Secondly, the coordination process is analysed using the prior developed analysis framework presented in Chapter 3. Since 'coordination process' is the central unit of analysis of this research, this later analysis helps to identify and describe the dependencies, and their coordinating strategies and associated challenges with the SD process.

As discussed in the methodology chapter (section 3.4.2), this activity analysis is conducted using Korpela et al.'s (Korpela et al., 2002) framework which is grounded on the activity theory. This activity analysis is useful to understand the Globally Distributed Agile Software Development (GDASD) project activities by articulating the actors, the work structures, ways of interaction, and coordination within the organisational boundary to achieve a shared goal. The following sections first describe the adaptation process of the framework for this case and then present the outcome of the analysis by mapping the activity elements following guidelines provided by (Korpela et al., 2002).

4.3.1 Adaptation of Activity-based Framework for Pigeon Project

One of the important usages of the Activity-based framework is to analyse software development as an Activity to identify elements that are “misfit”, and hence, can improve those areas for a better outcome. Coordination in DSD is also a ‘collective work activity’ that involves multiple actors performing their activities in a way that fits with each other to achieve a common goal. If the coordination activities are not appropriate, it will create conflicts, tensions, and challenges in achieving the goal. Since this framework lacks visual notations, this research adapted Laurent et al.’s visual presentation guidelines to present the activity elements (Laurent et al., 2010).

According to the checklist provided in this framework, the primary elements that are observed in an activity are *Outcome*, *Object and process*, *Actors*, *Means of work*, *Means of coordination and communication*, *Group or team*, and *Mode of operation*. Figure 4.6 illustrates the abstracted model used in the activity-based framework depicting all the elements. This model is being used while analysing the overall activity and its elements observed in the DASD process.

In the following section, each of these elements are briefly discussed and mapped with this case data. It is mentionable that the framework is adapted to include the distributed segments of the project and portray the software development process as the way of working. The visual notations proposed by (Laurent et al., 2010) are also adapted accordingly to match this study’s context.

4.3.2 Mapping Pigeon’s Work Activity Elements using Activity-based Framework

In Pigeon, a production release involves activities to convert the release requirements into the working product. A single production cycle consists of several work elements that are mapped using the activity-based framework. In this mapping process, the software development activities discussed in section 4.2 are analysed and mapped to the elements suggested in the activity-based framework. Finally, an overview of the entire activity is presented amalgamating the mapped elements in a diagram.

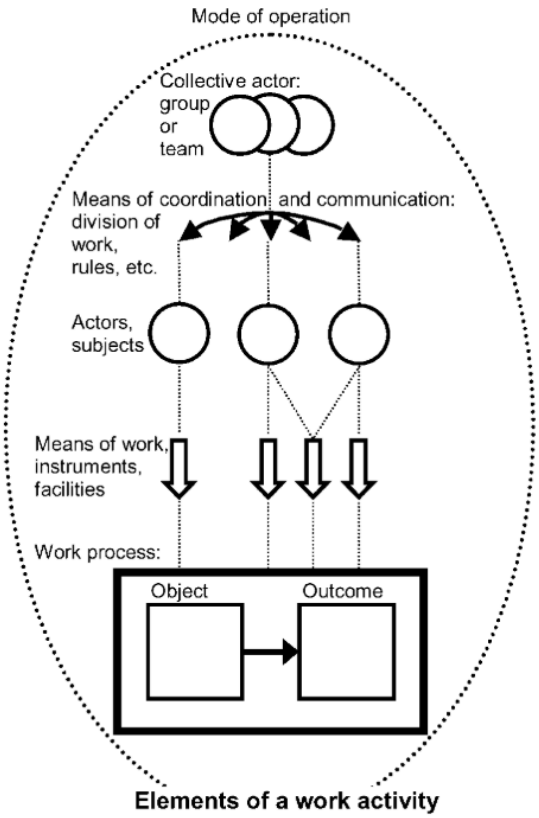

Figure 4.6: Elements of Activity based Framework (Korpela et al., 2002)

The visual elements presentation has three primary types: *roles*, *sites* and *artefacts*. Roles represent different kinds of stakeholders, Customers, Developers, Testers, Managers, Subject Matter Experts, Requirement Analysts, Local Spokes persons, and Users. Roles are depicted using Human shapes. There are three variants of stakeholders based on Multiplicity- One, Few and Many. Sites are represented by square boxes and have two variants, Single site and Multiple sites. The communication and relations between the elements are separated based on *Co-location*, *Distribution* and *Access*. Each relation is represented using a bi-directional arrow with different types of lines in between. The communication mediums are represented using common and well-known symbols and icons. The artefacts of the projects are depicted using the standard software engineering symbols. A summary of visual notations proposed in Laurent et al.'s work is shown in Figure 4.7.

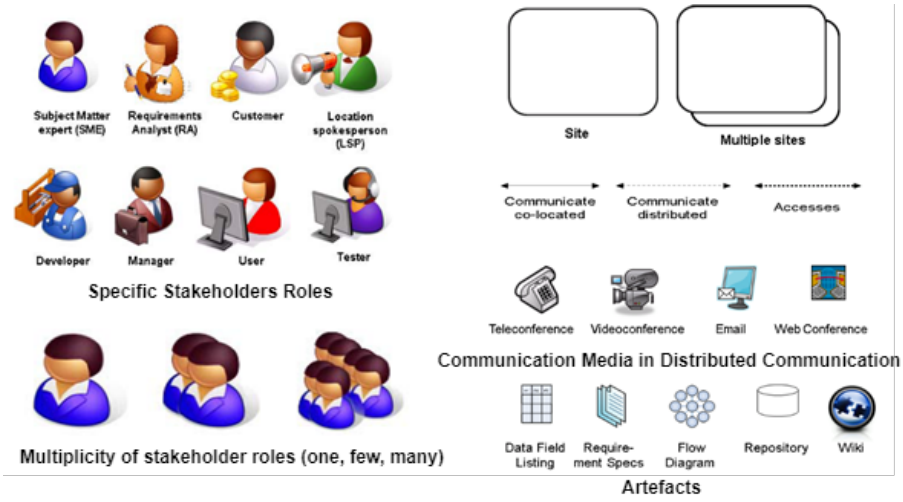

Figure 4.7: Visual Modelling notations proposed by (Laurent et al., 2010)

While presenting the actors and roles, Laurent et al.'s notations are enhanced by adding icons for Product Owners, Product Proxy, QA and other related stakeholders as shown in Figure 4.3. We have created notations for multiple teams and indicated their locations with their time differences from New Zealand Daylight Time (NZDT). The collocated and distributed elements are grouped and placed separately for clarity. These enhancements are reasonably performed to represent all the contextual elements and improve the readability of the notations. The visual notations of all the contextual elements will help to identify and highlight the coordination touchpoints within the work process.

4.3.2.1 Object and Outcome

The *Object* of the software development activity is the shared release goals (i.e. list of release requirements) upon which the actors perform their activities. The SD activity starts after release goals are finalised and allocated to the teams. The motive of these teams is to transform the shared object into an *Outcome*. In the case of software development, the requirements are the shared object based on which multiple development teams collaborate with each other to transform the requirements into a software release, i.e. the intended outcome of the activity.

4.3.2.2 Mode of Operations

Mode of operation characterizes the way of operating the work as a whole. The mode might be collaborative, technology-driven or can be combined. In DSD, the activities are performed in a collaborative manner involving people from distributed locations which are apparent in this case. This distributed collaboration is facilitated by various tools and technologies; therefore, the mode of operation is both collaborative and technology driven.

4.3.2.3 Actors and sites

An *actor* can be an individual who perform their activities individually or a group to transform the requirements into a usable outcome. For example, in ASD, the product owners identify and share the product requirements, the developers write codes, the testers test the features and finally, the production/operations team releases the software for the customers. Each of the individuals, from the Product Owner to the Production team member, is an actor of the software development work. Each actor in the joint work might be part of a collocated team closely working together, might be part of higher management, or might be from part of a globally distributed group that occasionally works together. All of them are part of the actor group since their collective intention is to achieve a common goal.

In Pigeon, the product development work is performed by multiple teams specialised in specific parts of the overall product; therefore these teams are the primary actors. There is a sixteen-hour time difference between the locations (i.e. NZ and USA) of the actors. Since English is the primary language in both sites, there are no language differences involved. The teams are working together for several years as the product is a mature one. Team C has three members who are working from different states of the USA. Other development teams have collocated team members who occasionally work remotely from home.

The feature enhancement requirements and prioritisation decisions are made by Product Owners (POs) who are located in different parts of Europe. The time difference with the POs is twelve-hour and six-hour with NZ and US teams, respectively. There is an NZ-based Product Manager (PM) who acts as the Proxy Product Owner for all the development teams. Much of the product enhancement ideas come from the Proxy PO and make the final calls for the priority conflicts between the teams. Once completed, the development works must be checked and approved by the QA teams before release. The QA Teams are distributed in three different sites in India and Japan, so the time difference ranges from four to eight hours with NZ teams.

In summary, the actors of this development activities include Product Owners (POs), Proxy PO, Development teams and the QA teams. All the actors and their locations with their time zones are indicated in Figure 4.8. All the actors have specialised knowledge and expertise that are required in the development activity. They collaborate and coordinate their actions using mediating instruments to transform the requirements into a software release.

4.3.2.4 Means of communication and coordination

Since the actors are part of the collective work, they need to collaborate with others to perform their activities and finally, integrate them to form the outcome. The *means of coordination and communication* are used to coordinate each one's work in the joint work process. Meetings, phone calls, division of labour, organisational rules, and schedules are some of the examples of means of coordination and communication.

Both synchronous and asynchronous mediums of communication have been applied to coordinate between multiple sites. Synchronous communications are performed over meetings using Voice Conference calls. Teams also use other synchronous communication tools for internal communication. For instance, Team R uses Slack, an Instant chat messenger (IM), to frequently communicate and share their daily work

status that is being used in daily standup. Additionally, Email, WebEx, Jira and SMART tools are being used for communication and coordination between sites.

4.3.2.5 Means of work or mediating instruments

While working on this transformation process, actors use different kinds of *instruments or means* for their work. These means can be any material, e.g. tools, technology, or can be immaterial, e.g. language, skills, or theories.

Examples of means of work and mediating instruments found in this case were an Issue-tracking system (e.g. Jira) and an electronic storyboard. Jira was used to share the requirements based on which teams were allocated specific requirements in the form of product enhancement user stories which were put back into Jira as Product Backlog items. The user stories were implemented in two-weekly sprints which were preceded by sprint planning activities. Daily standup meetings were used for regular coordination which was followed by sprint reviews and retrospectives. Along with Scrum artefacts such as product/release backlog, sprint backlog, and burndown charts, an electronic storyboard using Jira was also used to share and maintain the artefacts.

Most of the project-related documents are shared using *Confluence (wiki)* where each team has its own workspace to create and share its knowledge resources. For instance, when Team R creates *'How to' documents* to share details about new features, solutions of previous critical issues, and other important technical information, they can be updated by other team members. The Product Code is shared, and version controlled using *version control software*. Teams have their own branches along with a final branch for integration and production purpose. Besides, teams use other mediating tools for their software development activities, such as Docker for containerised work environments, and online tools for Retrospectives.

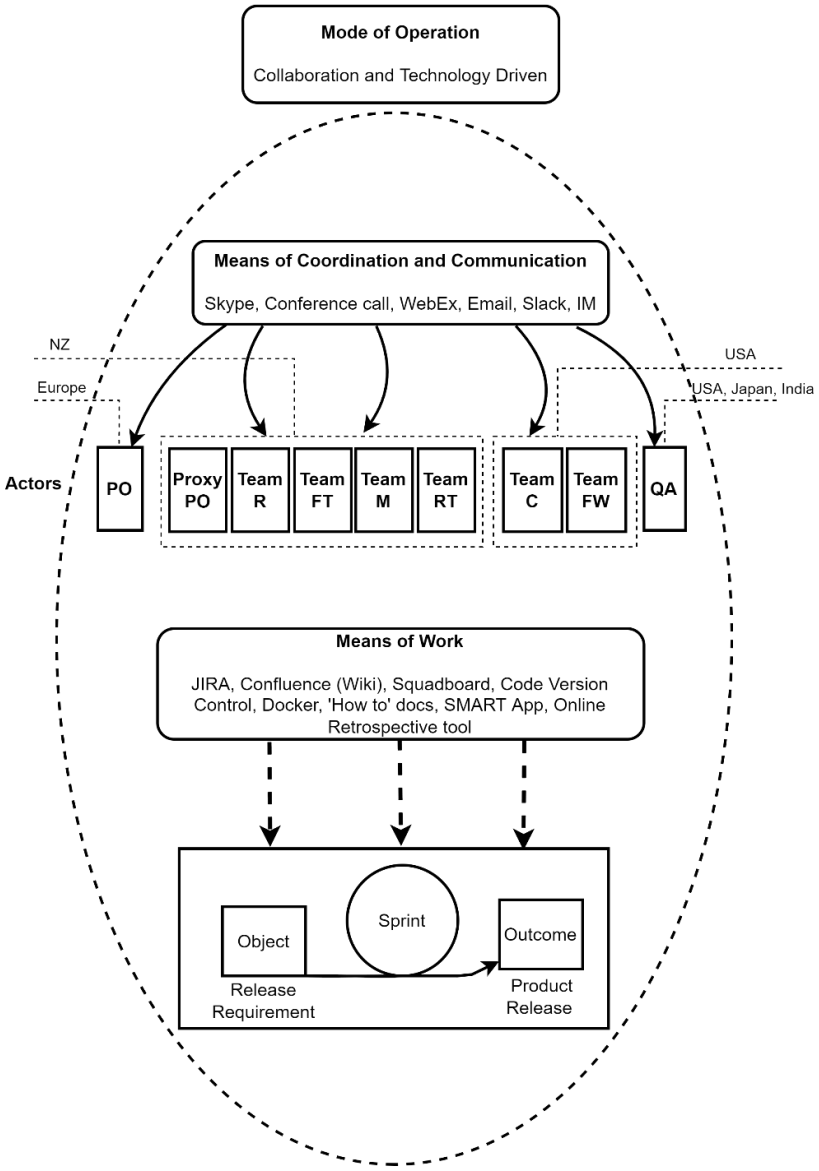

Figure 4.8: Elements of Distributed SD work activity in Pigeon

Finally, we have accumulated and portrayed each of the elements of the Distributed Agile Software Development activity in Figure 4.8. All these elements need to fit together to complete the conversion of the release goals into a production release. The actors are represented using icons and notation of their roles or group names along with their geographical locations. The roles and groups of the actors are created by the organisation based on their specialisation. They perform their activities using means of work based on their assigned roles and division of work. Their collaborative work activities are harmonised by means of coordination and communication, which are shown using curved lines and arrows. The iterative development cycles depicted at the

bottom of the figure show the software development process used to transform the release goals into product releases which are elaborated on in the previous section. This high-level view of the activity supplements the coordination analysis presented in the next section.

4.4 Overview of Coordination Process Analysis

The analysis of the coordination process is crucial for this study since it focuses on understanding the dependencies and their coordination mechanisms; thus could evaluate their effectiveness in the DASD context. The framework consists of three main components: Dependency, Coordination mechanisms, and Coordination effectiveness.

Before presenting the analysis of the case, the framework of analysis (first presented in Chapter 3, Figure 3.6) is summarised below:

1) A dependency occurs when the progress of an action is dependent on the timely completion of another action/task, or the presence of a thing, where a thing can be an artifact or a person or a piece of information. Dependencies are analysed based on purpose e.g. *who*, *for what,* and *why*. There are probable issues associated with the dependencies that could impact the outcomes, such as any forms of delay, misunderstanding, rework, technical debt, increased communication and poor-quality work output. The following tools are used to conduct the dependency analysis:

    a. Coordination theory (Malone & Crowston, 1994) to distinguish different types of dependencies in the SD process,
    b. Delay diagrams to analyse the delays caused due to dependency issues.

2) Coordination mechanisms are used to address different types of dependencies that are identified in the process. A coordination mechanism could be an entity (person or artifact) or activity (practice or technique) used to manage a dependency. Additionally, identified mechanisms are further analysed based on the roles, format of the resources shared or involved, mode of exchange,

notification process, synchronicity and time sensitivity. While considering the quality of the mechanisms, four key aspects are applied: applicability (i.e. fit for purpose), timeliness, level of shared understanding, and shared awareness. The analysis of mechanisms includes investigating the activities either agile or non-agile practices, tools or artefacts or any other strategy that is evident in the interview data.

3) The suitability and effectiveness of the coordination mechanisms are assessed by analysing whether the required resources (i.e. the right thing) are available at right time and at right place, and whether proper understanding is achieved about the desired outcomes (*know why*), current work process (*know what is going on*, *who is doing what*), individual responsibilities (*what to do and when*), the availability and location of the resources (*who knows what* and *who is available*). The decision rule applied to measure the effectiveness is that "*Does the current coordination mechanism reduce or eliminate the dependency risks?*". We have presented an example of this evaluation process while discussing the case findings in Section 4.6. The purpose of these examinations is to get insight about the appropriateness of those mechanisms and, at the same time, identify factors involved in choosing the mechanisms.

4) The selection of the coordination mechanisms depends on two types of factors: dispersion factors (time, spatial and culture) and task factors (uncertainty, complexity and duration) that have impacts on the selection and quality of the coordination mechanisms.

Using this framework, the dependencies, mechanisms, and challenges are analysed in relation to the overall SD process and factors that impact coordination performance are identified.

The analysis helped in locating the critical coordination areas within the SD process. In this research, the criticality of coordination is measured using three factors: the number

of dependencies per development cycle, the number of teams and globally distributed sites involved, and probable risks. It is obvious that a high number of interdependencies involving a higher number of resources would require a high level of coordination efforts (Dietrich et al., 2013) which creates difficulty in effective management of all the dependencies. Besides, global distribution reduces overlapping work hours and opportunities for informal communication between the parties, which complicates dependency management; hence considered a cause of criticality in distributed coordination. Another criterion for criticality is the number of risks involved in the dependency. If a dependency possesses a number of vulnerabilities that have the potential to cause high impacts, e.g. delay in the downstream work or release or feature delivery, then it is considered a potential risk; hence categorised as a critical dependency. The key dependencies are identified using these criticality factors, which are followed by the coordination mechanism analysis. An overview of the analysis process is illustrated in Figure 4.9 and the findings of this analysis are presented in the following section.

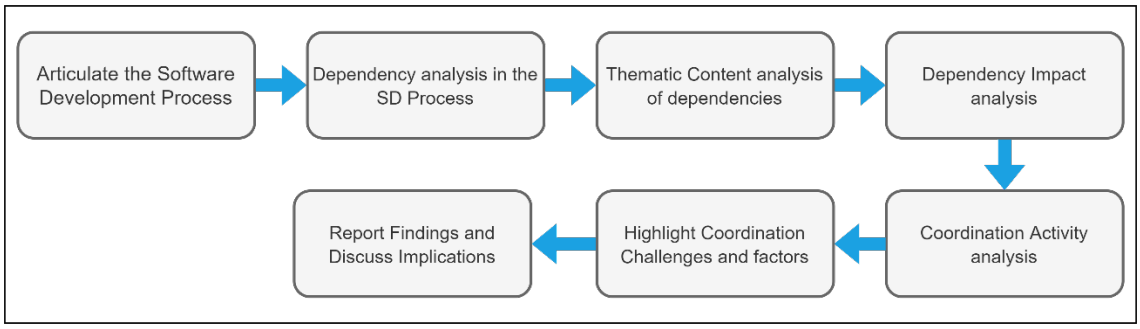

Figure 4.9: Steps of the coordination analysis process

4.5 Findings of the Coordination process analysis

The dependencies are classified into two main categories: *Critical* and *Non-critical*. As discussed at the end of the previous section, critical dependencies are associated with probable high impacts, if they are not managed with effective coordination mechanisms, whereas non-critical dependencies do not.

The critical dependencies in the SD process are illustrated in Figure 4.10. Each dependency is presented with its description, justification of categorisation, and possible impacts with examples in Table 4.3.

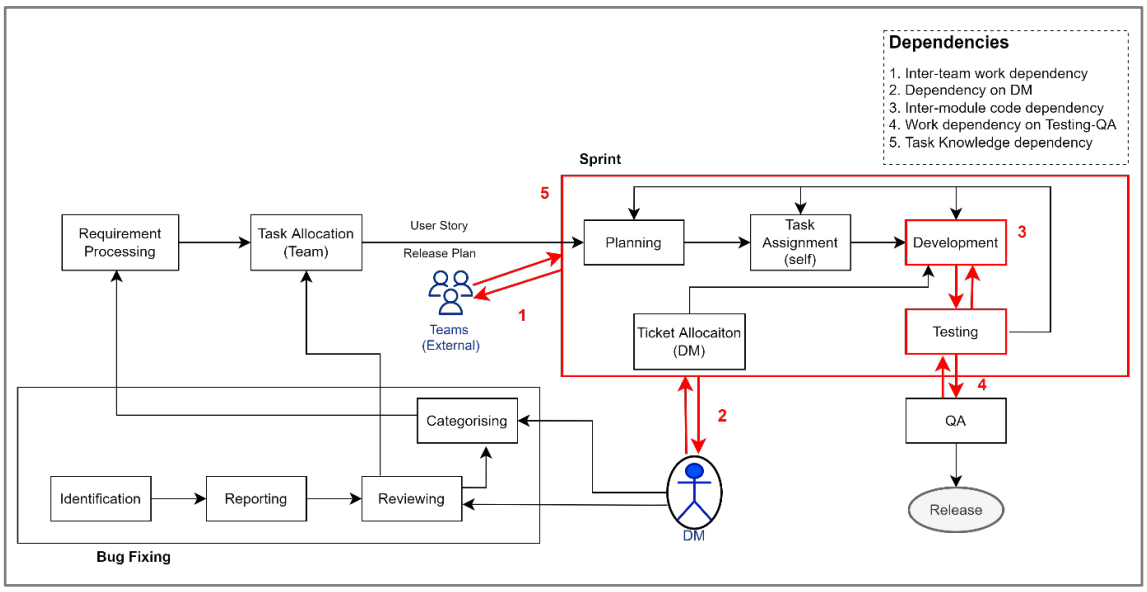

Figure 4.10: Critical Dependencies within the DASD process

4.5.1 Dependency 1: Work output dependency between Development Teams

Several issues related to lack of knowledge about other development teams' activities led to unrealistic expectations between the teams. The following sections present a deeper analysis of the dependencies and coordinating activities related to the interdependency between development teams.

4.5.1.1 Dependency Description:

In this dependency, the work progress of one team relies upon the timely completion of work done by another team. For example, whenever a development team needs access to data related to any feature or product enhancement, they cannot proceed with their work until Team R complete its work is completed. This dependency is well-illustrated in the PSM1's statement

"*On the other end is we delivered software and then another team relies on, and they have issues how do we interact with that team.*" (PSM1)

This was categorised as a ‘Pre-requisite’ work dependency since other teams rely on Team R’s work completion to be used for other tasks. It was later categorised as a sub-type of “work-output dependency” based on the coordination needs between the work progress of teams’ development activities, i.e. the work progress of other development teams relies on the timely completion of Team R’s development work.

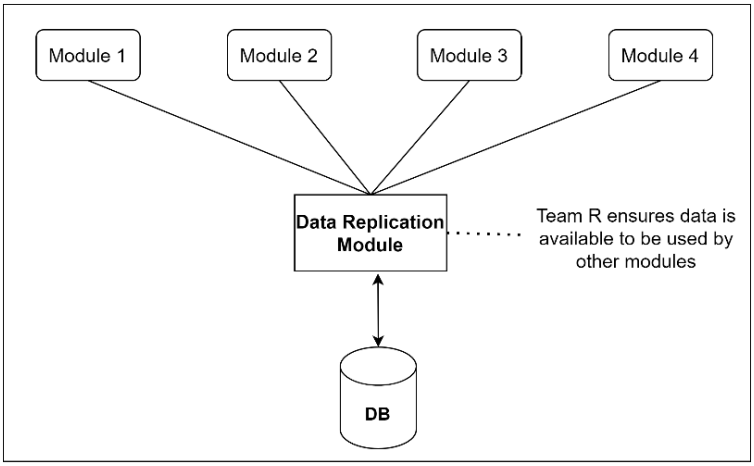

Figure 4.11: Dependency on Data Replication Service

This work-output dependencies can exist in any phase of the SD process, and they have potential high impacts the downstream work progress and release delivery activities in globally distributed teams; therefore, it is categorised as a high-risk dependency. As previously described in the system architecture, Team R is responsible for facilitating all types of data required by different modules for their functionality (See Figure 4.11). When data is requested, Team R must first complete its data replication work before other teams can access that data through the replication service. Until Team R finishes their coding and testing, other teams cannot complete their development and testing activities. The following scenario illustrates the complexity of this dependency and its potential impacts of this dependency.

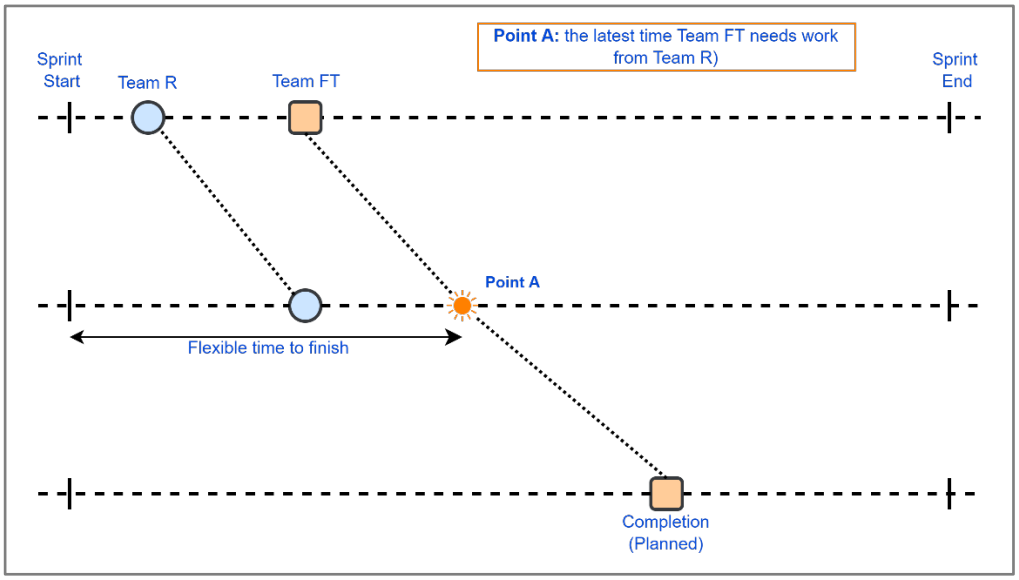

Figure 4.12: Work-output dependency between Team R and Team FT

For example, while working in the front-end services, Team FT needs to call a backend service that collects specialised data from the database and loads it in different parts of the systems which are then available for presenting in the front-end module. Since the existing service does not provide that specialised data, Team FT needs some changes to be made in the replication service side that is managed by Team R. In this situation, the dependency requirements are:

- Team FT's work progress depends on Team R's timely work completion
- Team R relies on Team FT to inform about the need of work
- The nature of Team R's data replication work depends on information from Team FT

Team FT needs to communicate the work requests through the proper channel. Prior to starting the work and during their work, Team R requires additional information from Team FT, such as what features they are working on, what kind of data they require, what format the data needs to be in, by when and how it wants access to the data, and so on.

In Figure 4.12, **Point A** represents the latest time for Team FT to be able to use the work output from Team R that should be completed before that point, in order to prevent blocking the progress of Team FT's work. Teams need to share work updates

regularly and coordinate the information exchange required for the timely completion of work by both sides. The prescribed scenario represents the work-output dependency on Team R, which was the most common type of dependency observed in the SD process. The next section presents further analysis of the persistent issues and impacts related to this dependency.

Table 4.4 presents examples with evidence from data illustrating different types of work-output dependencies on Team R. This table briefly describes each of these dependencies and explains why it is categorised as a work-output dependency.

Table 4.3: Summary of Key Dependencies in Pigeon

SL#	Dependency	Dependency Description	Impact	Explanation/justification
1.	Work-output dependency between globally distributed teams	The work completed by Team R is needed for another team to complete their work. For example, Team FT cannot proceed with their front-end development until the data replication work in completed by Team	Interruption of work will create delay as Team R does not has any plan for the work need. Possible work delay for: Team R, if they accept requests during the sprint and delay their work, Other teams if Team R continues their work. Both teams, if Team R accepts the work but will take longer to finish due to work complexity	Another team needs the work from Team R to complete their feature development and the delivery of the feature is blocked for the work from Team R. (PSM1) says, "…*you're trying to do some feature that you haven't talked to us about, its 2 or 3 weeks and you need our help. so it is not going to deliver because of us*" (dependency issue)
2.	Dependency on the Dev. Manager	Team R members extensively depend on Dev. Manager for expertise knowledge that is required to perform activities and decision-making. A high volume of debugging work relies on Dev. Manager. Availability and responsiveness of the Dev. Manager is crucial for decision-making and task allocation.	Possible Knowledge Acquisition bottleneck at Dev. Manager. Local knowledge and debugging capability get siloed to Dev Manager. Dev manager being overwhelmed with firefighting to the detriment of team Nurturing.	Being the central source of domain and expertise knowledge, the DM is creating high risk of single point of failure that has possible impacts in decision making and task allocation.
3.	Inter-module code dependency	Replication and Communication modules have high overlapped functional areas that create a critical technical dependency between Team R and Team C	Work output from either side has a roll-on effect on other modules that can cause bugs. Delay in downstream work due to bugs raised from other side code	Code changes made in other module is creating bugs that are blocking the deployment of features or critical customer bug fix. (PDV1) says, "*Sometimes other teams' bug which causes our stuff to break and then we get a lot of bugs, but it's all one cause of other team's issue. Sometimes you can't deploy your fix*'

4.	Work-output Dependency on Testing and QA	Testing and QA activities have strong work-input and work-output dependency that has implications for the overall work output of Team R	Inconsistent work cycles have a high potential to delay work completion that can lead to under-delivery and poor-quality work-output Possible activity bottleneck at the Solo tester	The work output of the Tester ensures the work quality of the developers (i.e. work-input for local tester) is good enough to go to the next level (i.e. work-input for QA) and the QA ensures the developed features are releasable. *(PDV4) says, "The local tester is testing thoroughly all the technical works, a bit low level. After it's passed, we are confident about what we did is correct. then we are giving it to the other testers (offshore QA & integration testers)"*

Table 4.4: Work-output Dependency categorisation

SL#	Secondary level coding	Primary Coding	Dependency Description	Example Evidence from transcripts
1	Work-output	Review work	The work completed by Team R needs to be approved by another team; otherwise the work progress is blocked (i.e. work cannot be submitted to QA)	"*we delivered software and then another team relies on, and they have issues how do we interact with that team. and then when it needs to be documented and we have to talk to someone of them to get it approved and final step is actually the QA*"
2	Work-output	Ad-hoc work	Another team relies on Team R to do some work that is required for their work	"*ohh.. you're trying to do some feature that you haven't talked to us about, its 2 or 3 weeks and you need our help. so it's not going to deliver because of us. That's not really fair, because you didn't tell us about this upfront.*" "*whenever anyone's got any problem or they want something done, you suddenly get something someone contact you or Reuben would say*" "*When someone comes into something and I say, how long have you been doing this? This impacts us and you should've including us for the past six months. If you not being successful, is because you are not talking to us.*"
3	Work-output	Support work	Other teams require support from Team R to resolve their production issues	"*if someone has a production issue they've come to for help.*"
4	Work-output	Data Replication	Development teams rely on Team R to facilitate data that is being consumed by the feature they are working on	"*The Replication Service is core and like if other product or software need some data, so we provide that. Definitely some enhancement is going on some other product and they need some data. we have to*

				make sure that by doing some coding."
5	Work-output	Bug Fixing	Team R and Team C depends on each other for their development works because of the multiplexing effects of their work	"*I guess there' a sort of connection between the online-offline but we do it on top of framework, and Team C I guess take the other half... of the backend service*" "*Generally, we get their bugs and they get our bugs.* "
6	Work-output	Pre-requisite work		"*there's parts that you don't touch, and you've got to feed into another team. Like You guys need to fix this part, and then we can do that*" "*On the other end is we delivered software and then another team relies on, and they have issues how do we interact with that team*."

4.5.1.2 Dependency Issue & Impact

One of the key issues associated with this dependency is the coordination of teams' unexpected and uncertain work-specific expectations, i.e. lack of knowledge about the work that needs to be completed by a team that is required by another team to proceed with their work. For example, in the above scenario, Team R was not aware of the need to do work for Team FT until they were stuck and urgently needed the work. Hence, Team R faced difficulties in managing unanticipated requests during the sprints from other teams. In order to address this, Team R either has to interrupt its own scheduled work at the cost of its release goals; or create a delay in the progress of Team FT's work. This highlights the potential risks of delay in either situation due to a lack of knowledge about the work requirements.

On the contrary, Team FT did not inform Team R in advance about the work requirements because of the lack of awareness of any coordination needs until the very end when they were blocked. "*You need our help. so, it's not going to deliver because of us*" (PSM1), i.e. Team FT needed some work output from Team R before the features under development could be released. The reason for this issue is that Team FT did not perform the stakeholder analysis to identify the work needed from other teams. If work requirements were identified well ahead, Team FT could have communicated with Team R to give them enough lead time to plan and coordinate the work requests from Team FT.

Three possible scenarios are used to illustrate the impacts of this unanticipated dependency between multiple development teams.

i. Team R accepts the delay
ii. Team R transfers the delay
iii. Team R delays to complete their work

i. *Team R accepts the delay*

If Team R decides to temporarily stop their work by accepting a work request from Team FT, the impact of delay on Team FT is minimal. However, this will create a delay in Team R's own scheduled work that might result in a delay in delivering customer feature/bug fixing (as shown in Figure 4.13). This situation is termed as '*Delay Accept*'. In most cases, Team R accepted such requests and took extra load on top of their own sprint tasks, which implied poor quality in their output leading to rework and further delays.

"And the technical people, who've been here, if someone has a production issue they've come to for help. If [DM] isn't taking a lot on his shoulder, who else is gonna do it, is what he says. Which is fine up until the point when product management go over up that [DM] and his team hasn't delivered this thing, why not?" [PSM1]

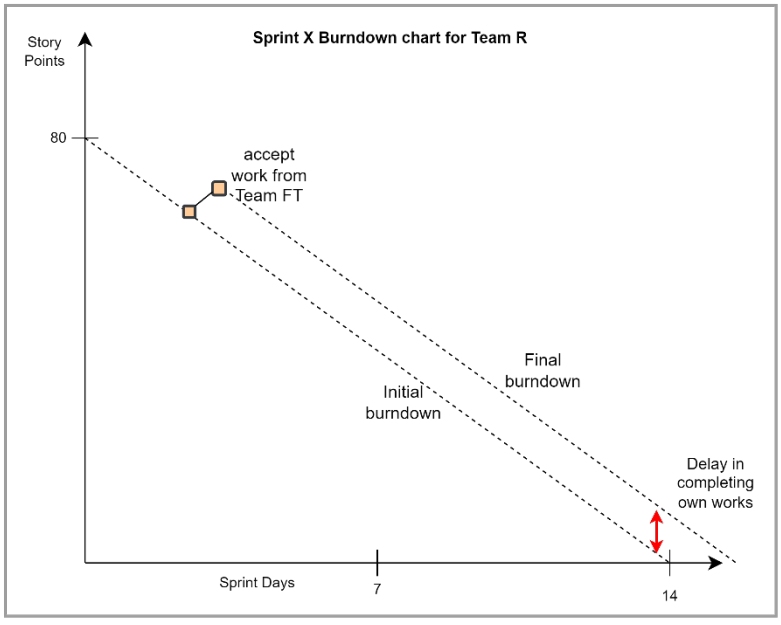

Figure 4.13: Example scenario of 'Delay Accept' for Team R

ii. *Team R transfers the delay*

If Team R decides to continue with their own work first, it will delay Team FT's work. In this case, Team R is transferring the delay to the other side and Team FT has to

wait until Team R is ready to accept their work. This is termed as '*Delay transfer*' as illustrated in Figure 4.14.

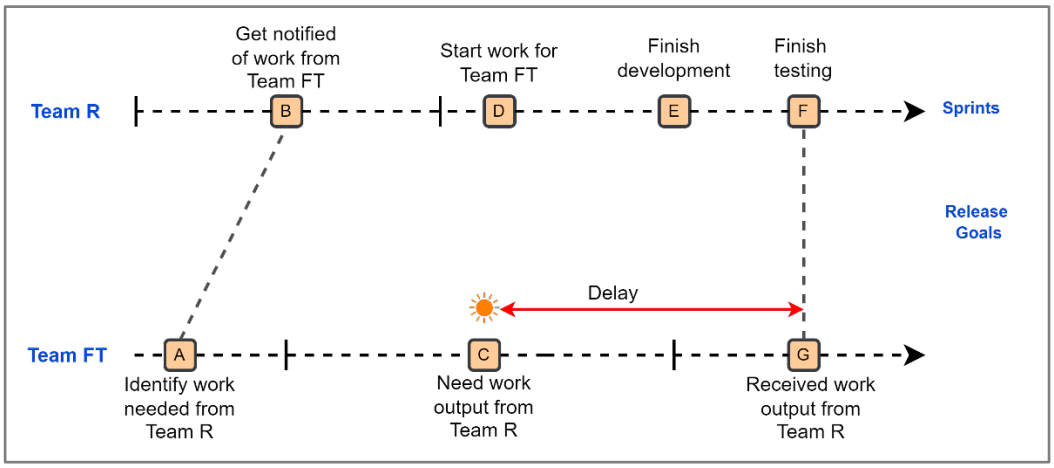

Figure 4.14: Example scenario of 'Delay Transfer'

This situation was described in frustration by Team R's Scrum master when another team requested some tasks from Team R which they were not aware of, "…*ohh you're trying to do some feature that you haven't talked to us about, its 2 or 3 weeks and you need our help. so it's not going to deliver because of us*" (PSM). She indicated that it was a difficult choice for Team R to either accept in current sprint or defer for future sprint. Deferring the request meant delaying the delivery of other team's features whereas accepting the request would interrupt their own delivery goals and schedules.

iii. Team R delays to complete their work

At times, Team R could not complete the work that was needed by another team on time which complicates the situation. For example, while Team FT is waiting for Team R to finish their development activities, there might be unexpected delay for several reasons, For instance, on several occasions works that were initially categorised as small, often turned out to be much bigger because they had misjudged the complexity of the work.

"*Sometimes it easy and we can do it easily, but sometimes, it comes back as a bigger piece of work, and we've got to prioritisation choice, or you have to decide. We've got three more weeks to release, we can do everything we're in the middle of or we can drop it off*". (PDV3)

In such situations, Team R needs additional time to complete their work which will keep the other team waiting. This could lead to two types of potential impact,

a) In Figure 4.15, Team R finished their work and delivers the output at **Point B** which is past **Point A**. From point B, Team FT cannot complete their work within the deadline. Team FT must drop the work of this feature and plan to include it in a future release.

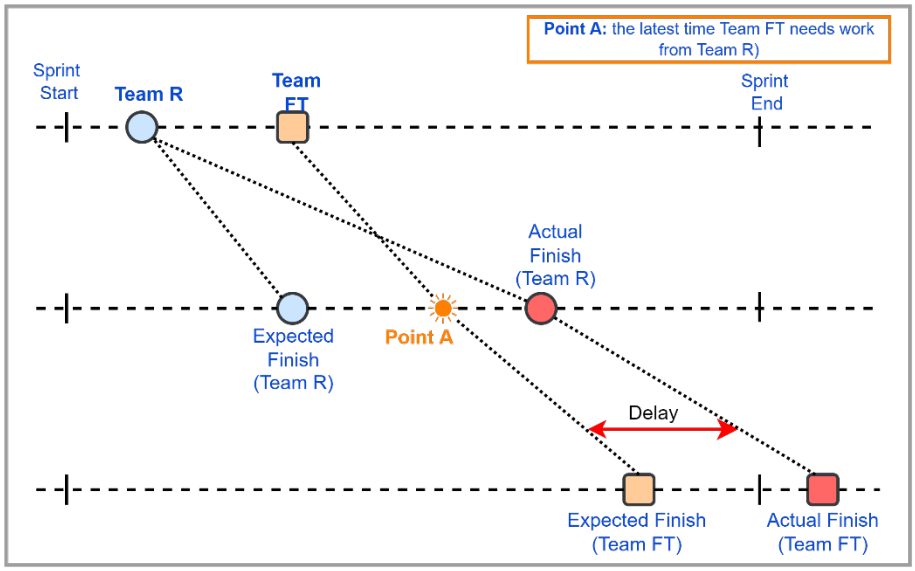

Figure 4.15: Possible Delay scenario for work-output dependency between Team R & Team FT

b) In Figure 4.16, where Team FT receives the work output from Team R at **Point B**, and they have a very short period to complete the remaining work. If Team FT decides to continue their work, they must speed up their development to meet the sprint deadline. In such a case, the team has to trade off the quality of the work with on-time delivery of sprint items. Poor quality works generally results in unstable system performance and often generates bugs and potentially creates technical debt (Salaou et al., 2020).

The root cause of the issues discussed above highlighted the lack of an effective mechanism for early identification of work required from Team R, i.e. Team R does not have any prior knowledge about the work needs of other teams, sometimes until very late when the delays become unavoidable.

"*Before we didn't know what's other team were doing, until they have been blocked by us. it takes a while to figure out the problem, to then introduced the effects and then to get buy in.*" (PSM1)

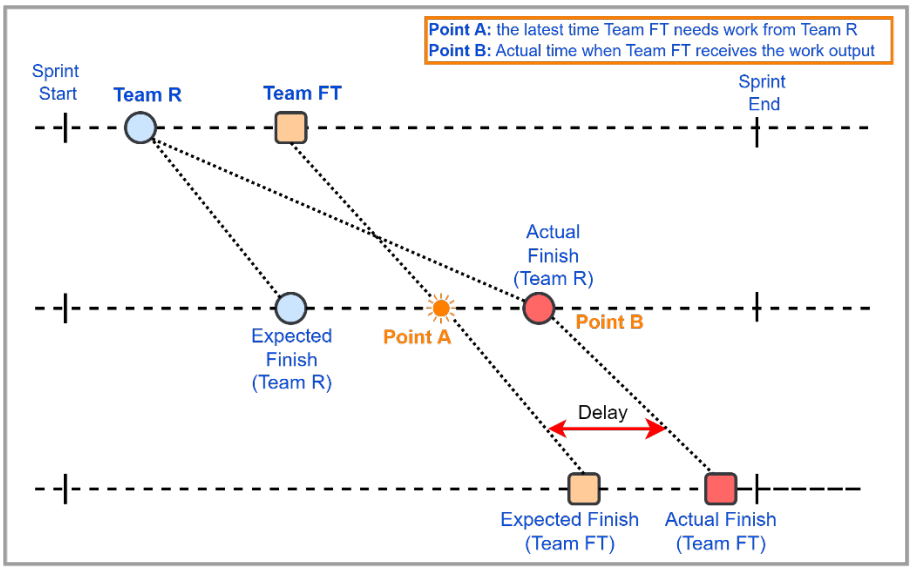

Figure 4.16: Possible scenario for Technical Debt resulting from this dependency

To avoid the impacts of a situation when there is a delay in Team R's work completion, there should be a mechanism through which Team FT can get regular updates about Team R's work progress. Team FT will also get early notification of any possible delay due to any blocker in Team R side work. Accordingly, Team FT can take necessary measures to reduce the delay, such as communicating with Team R to know the issues they are facing so that they can work together to address the issue.

Using the above scenario, four kinds of delays resulting from this dependency can be anticipated:

- *Delay 1:* Possible delay between Team FT being aware of its need for Team R to do work and the time Team R is informed is 1 to 7 days.
- *Delay 2:* Possible delay time from Team R being informed and starting the work is 1 to 2 weeks.
- *Delay 3:* The time for Team R to start and finish the work (including testing) is 2 to 3 weeks.
- *Delay 4:* The time between Team R finishing work and Team FT releasing is 1 to 2 weeks.

The root cause for the *delay 1* and possibly for *delay 2* is the interval between the weekly coordination meetings (i.e. Tech sync meeting). Since Team R does not have any early awareness of Team FT's work, it is challenging to promptly fulfil their request. Other factors such as complexity of the work, time differences, responsiveness from Team FT side, availability of the DM, and decision making can delay the commencement of the required work till the next sprint (i.e. 2-3 weeks considering two weeks sprint) as portrayed in Figure 4.17.

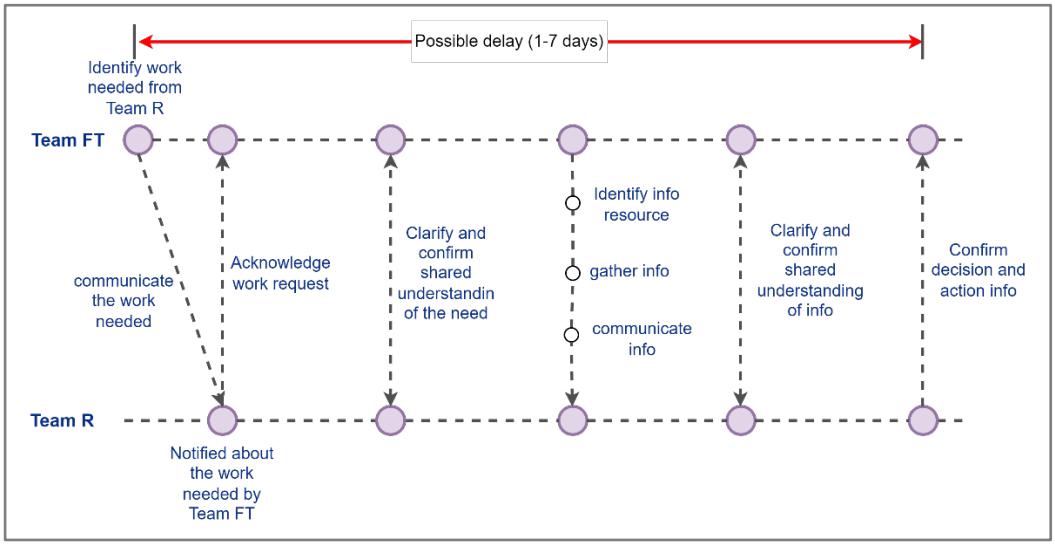

Figure 4.17: Example scenario for Delay 1 & 2

4.5.1.3 Analysis of coordination mechanisms

The weekly group meeting, termed as 'Tech Sync' meeting was the primary mechanism used coordinate the inter-team work-output dependency. This meeting was conducted via a conference call where both local and distributed development teams participate and share their work needs or issues that requires work from other teams. It

serves as an mechanism for coordinating the needs and issues of distributed teams, negotiating their work requirements and priorities, exchanging required information and progress updates. However, there is no mechanism available for early detection of dependent works and in most cases, teams communicate with Team R only when their work is blocked. As a result, Team R struggles to accommodate the Ad-hoc request in the ongoing sprint leading to delays as discussed above.

Email and Jira are other forms of mechanisms used to coordinate with Team R for any work request. The DM receives most emails and notifications of Jira tickets and decides whether to 'accept delay' or 'transfer delay'. However, this decision making was quite challenging due to lack of participation of other teams. The DM disclosed that the teams never communicated their needs or responded to queries until it was too late to meet their deadline. Each team has their work priorities for the release goal that creates impediments for any ad-hoc request to get prioritised. Due to the information exchange complexity and competing work priorities, there is high possibility of frustrations between teams and delay in work progress that can impact the planned release. In some extreme cases, the PM was involved to cut through the tensions by deciding the work priorities between the teams.

4.5.1.4 Challenges

The primary challenge identified for this work-output dependency is the communication between development teams. Due to the time difference, it is difficult to initiate and maintain continuous communication between distributed teams. For example, there is a range of 12-16 hours' time difference with Europe and US based teams which provides a small amount of overlapping work hours. The SM of Team R shared the challenges associated with e coordinating with a team in Poland when the Poland team had to wait till 8 pm to participate in a meeting with Team R (7 am NZT), and the meeting lasted for two hours call to solve the issue.

Another articulated challenge in communication was the rigid mindset of the development teams towards thinking that how their work affects other teams. While talking about the coordination challenges, the Scrum Master brought up this issue and said, "*we had a list at the start of the release, we think it will impact them and contact them. 'ohh no no, we don't need you that's fine'. Are you sure?... that's cool okay."* However, Team R's anticipation was correct, and other team realised the impact at a later stage, and said, *"ohh you did…*". Due to this poor mindset, other teams were not aware of any potential work impact from Team R, and even though Team R proactively notified, they did not collaborate until they need it. It was difficult to force teams to think of any potential work dependency at the start of any release or sprint which created pressure on Team R and produced a high risk of coordination breakdown.

Miscommunication is another perceived challenge resulting from the communication channels and language barriers that was creating frustrations among the teams. The conference calls and emails are the primary communication channel used in this case that lacks the physical expressions. Due to the lack of face-to-face contact between the teams, participants do not know how other parties look like which makes them feel isolated, and hence, do not want to engage in the conversation.

"*If you pursued someone to come to a meeting that they don't know people, they don't know what's gonna happen and you spend 50 mins, definitely on my toes, then you don't get to conversation, or they go, that was a waste of my time.*" (PSM1)

While observing the distributed team coordination meetings (listed in Table 3.3 in Chapter 3), we felt the same as it was hard to identify the person talking from the other side. The audio quality and peoples' accents were making it difficult to understand the importance or priority of the work and they end up not giving enough time to solve any conflict. Participants also reported that the difference in terminology used in email is also contributing to creating misunderstandings that can delay the work progress. For example, the DM mentioned an email he received by saying,

“I've got these jobs that I'm trying to run in parallel, right? Okay, what do you mean by jobs?... Because it wasn't quite clear, it didn't make sense what they were saying’ (PDM)

Since the intention or the problem was not clearly mentioned, the DM had to continue the email conversation further until he realised that “*there was a misunderstanding of how the system works. And I was able to clarify it*” (PDM). Besides, this slow and continuous email communication was interrupting the flow of work which was another challenge faced in the distributed coordination.

Table 4.5: Summary of Dependencies on Dev. Manager

Process Activities	Dependency					
	Knowledge			Resource		Activity
	Requirement	Expertise	Task	Decision-Making Authority	Entity	Work
Requirements gathering	√				√	
Requirement processing	√	√			√	
Assign to team	√		√			
User Story creation	√	√				√
Stakeholder analysis	√	√	√			
Sprint Planning	√	√	√		√ (High)	√
Task Allocation to members			√ (High)	√ (High)		√ (high)
Development		√ (High)	√ (High)			√ (High)
Daily Standup	√	√	√	√	√	
Acceptance Testing	√		√	√		
QA	√		√		√	
Regression and Integration Testing	√		√		√	
Deployment	√		√	√	√	
Bug Identification						
Bug Reporting						
Bug Reviewing & Processing		√ (High)	√ (High)	√ (High)		√ (High)
Bug Categorising		√ (High)	√	√ (High)		√ (High)

4.5.2 Dependency 2: Dependency on the Development Manager

Throughout the interviews, a recurring theme brought up was the central importance of the DM for several key process activities in the development and bug fixing process. A deeper analysis of the dependencies and associated coordinating activities revealed a complex system of knowledge, work and decision activities that relied heavily on the availability of the DM at the time of need. While it was not identified as an immediate issue because the DM *was* highly available at the time of the interviews, the sheer volume of dependency on this one person that has been identified signals a clear risk

with a potential high impact on work. The following sections present a deeper analysis of the situation to understand the breadth and depth of the dependency risks and suggest some ways to improve it. A summary of the dependencies identified on DM is presented in Table 4.5.

4.5.2.1 Dependency Description

This was identified as a key dependency since the progress of the development and bug fixing activities strongly relies on the timely decision and work output from the DM. This dependency can be further categorised according to purpose as knowledge, resource and activity.

The DM is the sole person who has knowledge of the various tasks and the skills and availability of team members. Therefore, a large amount of task allocation decisions related to bug fixing, work reviews, prioritisation, and assignment of team members are made by him. All the work requests from both local and global teams are also solely handled by the DM. Because of his strong product and task-specific knowledge, DM is the sole authority who makes decisions based on the triage of the problems to determine the criticality of the tasks and potential impacts on the clients. Having the high degree of authority and essential knowledge, the DM turned out to be the key resource person for the team.

The DM has gained strong product knowledge and expertise over a very long period on product features, internal architecture, functionalities, and inter-module dependencies required to do core coding and testing activities. Therefore, almost all team members rely solely on the DM for several development related activities (e.g. coding, testing, bug fixing) – see Table 4.5. Though some are familiar with the issue, they still depend on and wait for the DM to make decision as stated by one of the developers, "*sometimes I am not familiar and sometimes I know what the issue is but don't know how to solve it… and he might say that doesn't matter or you can shortcut*" (PDV3).

Most team members felt that the DM can make fast and effective decisions to solutions due to his deeper understanding of the source of the issue or problem. Similarly, he also responds to product specific queries or issues from distributed teams. For instance, as part of his regular communication with global touchpoints, DM replies to all the queries of the 'e-support' team located in India and Japan.

There is also high volume of activities that are solely dependent on the DM at any given point of time. For example, DM solely handles majority of the bug fixing and support activities which represents 50% of Team R's workload. Since the team has a strong commitment of initial response (i.e. 24 hours) and solution time (i.e. 3 days from the issue creation), the team heavily relies on the DM for timely completion.

This high level of dependency on DM warrants continuous availability and responsiveness for proper coordination. Though there are no specific issues to coordination per se, the strong dependence on a single person not only puts a large amount of stress and pressure on the DM, but it also poses real risks related to single point of failure and breakdown.

4.5.2.2 Dependency issue & Impact

Several dependencies related to development and bug fixing are shown as high risk in Table 4.5 as they either have significant impacts on causing delays or have the potential to cause delays if not properly managed. Being the prime task and expertise knowledge resource in the team, the DM needs to respond in a timely manner to many incoming requests, which creates a high risk of delay in knowledge acquisition. For example, as part of the role description, the DM says, "*part of it is very technical component, and that's why tend to be come through me to have a look at the problem, I break it down and then they go down to the team*". Therefore, if there is a confounding part of the task which is unclear or needs solution advice from the DM, all the developers and tester will ask the DM. As it is difficult for DM to instantly respond to all those requests, this could create delays in getting the necessary information.

In addition to the dependencies discussed above, the DM serves as the channel of information exchange between his team and external entities such as top management and other development teams. He filters unnecessary details and disseminates only important information that is relevant to the team, "*he's the main gate, I suppose you could say. and everything comes in and he will filter out all the crap and condense what we need to look at*" (PDV2). Besides, Team R members do not have adequate knowledge about external teams, for example, what other teams are doing, the specific expertise of each team, and how their work is interdependent. Due to a lack of such knowledge, they must rely on DM to identify and locate any needed expertise outside the team and then communicate with them.

All the issues discussed above create a *knowledge acquisition bottleneck* situation, as shown in Figure 4.18. Knowledge acquisition is the process through which domain-specific knowledge is extracted from a knowledge resource (e.g. human expert) to solve any problem in an expert system's development (e.g. software product, which can turn into a bottleneck due to relatively narrow bandwidth of knowledge transfer, acquisition delay, knowledge dissemination trap (Wagner, 2006). Such a bottleneck situation is considered to be a threat to effective team coordination in global software development (Anh et al., 2012).

"*[DM] explains what needs to be done in that particular sprint*" [PDV3]

"*If any requirement is not clear then I ask to [DM]*" [PDV3]

"*If there's any confusing part which is unclear, then definitely I go to [DM] and he mostly knows everything*" [PDV3]

The above statements made by one of the developers clearly indicate the high knowledge dependency on the DM. Other participants shared similar statements, that prove that there is a potential risk of a knowledge acquisition bottleneck, which will significantly impact the tasks of the developers and testers. For example, if the DM is unavailable or low responsive to answer the queries or provide details about a user

story, developers would rely on presumptions to meet the tight task schedules. Due to the lack of details, there is a high probability that the presumptions will be incorrect, which would result in serious consequences. By the time the problem/issue is correctly identified, it might be too late or very close to the end of the sprint. Another sprint may be necessary, which will require additional time and effort (a form of delay). Similar issues can arise in testing where the test cases are not correctly identified, causing additional work delays.

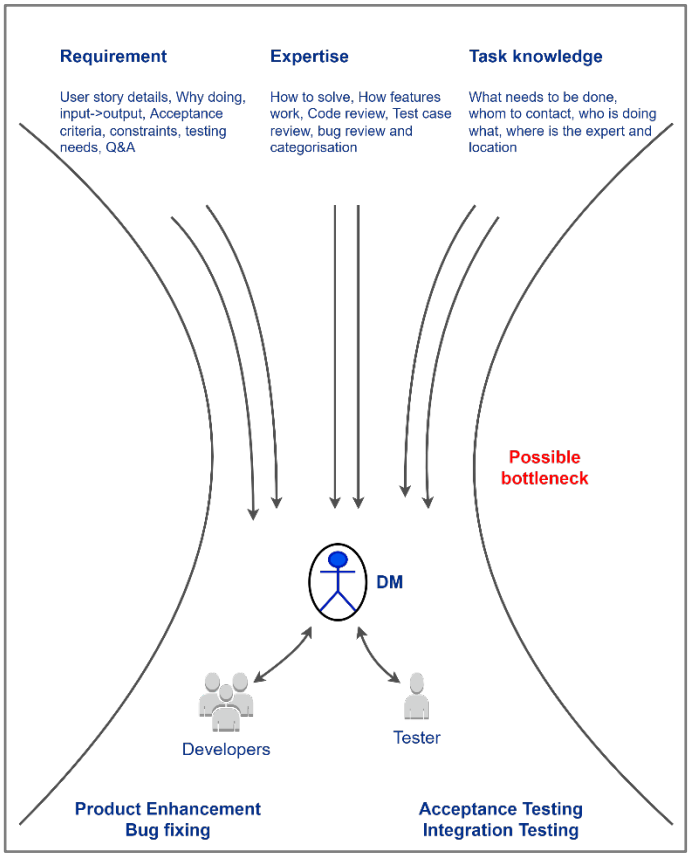

Figure 4.18: Possible Knowledge acquisition Bottleneck on Development Manager:

Another issue at the bug fixing and support work is due to high amount of decision making and task allocation dependency on DM. As previously discussed in section 4.2.2, DM is solely responsible for reviewing all incoming bugs, categorising, and assigning them to the team that creates significant risk of delay due to his unavailability or low responsiveness - team members can only acknowledge the receipt of the bugs but cannot act on them until the DM is available to review and take further decision.

Figure 4.19 presents a scenario that illustrates a potential delay of one to three days in the bug fixing process due to the unavailability of DM.

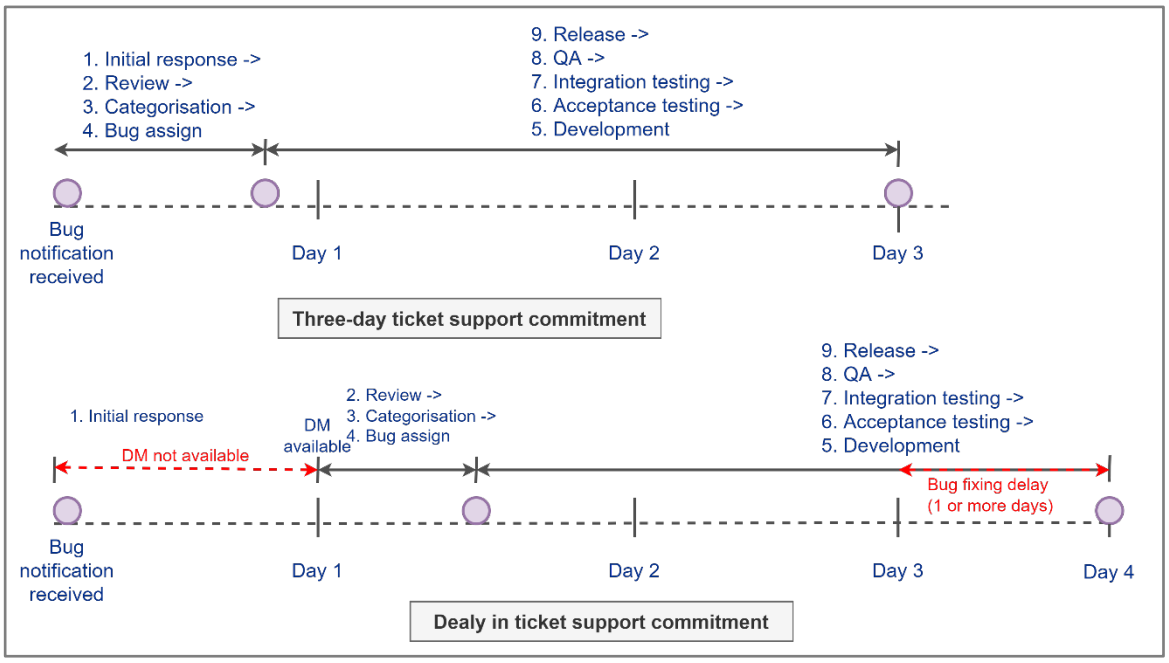

Figure 4.19: Possible range of delay in Bug fixing due to unavailability of DM

The first timeline shows the possible work cycle in a typical three-day ticket supporting process when the DM is available. The second timeline shows exemplar delay of one day in bug review, categorisation and task assignment cycle that cannot proceed due to DM's unavailability. Additional delays can result in due to time zone differences with the Integration and QA teams which is 4 hours and 7.5 hours difference respectively between NZ-Japan and NZ-India. The above exemplar scenario shows an estimated delay ranging from one to three days due to the availability and response time issue on both sides.

In addition, the significant amount of workload as a solo decision-making and task distribution authority creates extensive pressure on his role as line manager that has obvious impacts both on his and the teams' performance. The DM is responsible for the performance evaluations of staff, their professional development, and team management activities. The DM is overburdened due to his multiple roles as team representative, decision making authority and as the central expertise in the team,

which allows little room to monitor and investigate team's performance. Evidence from the interview data shows that Team R does not conduct regular 'Retrospective' sessions as they don't have enough feedback available to share or discuss. The DM primitively decides about the performance of the individuals and the process which is then shared individually or in a group session. Though it seems more like a performance evaluation session rather than retrospective session, the session highlights the discrepancies in the current team management process, which is detrimental to team nurturing. Since the DM is overburdened with loads of activities, there is a high chance of burnout which will significantly impact overall team performance.

One of the main reasons for the DM's overwhelming amount of bug fixing and support workload could be attributed to the lack of clarity in the distribution of development resource. Firstly, as there is no specific metric to measure the incoming work requests for Team R, the distribution of work resources to fulfil the demands is not always well-balanced. For example, though almost 50% of the workloads are bug related and support tasks, only one out of the four developers are assigned to work with the DM. Therefore, It is presumable that the 3 days strict issue resolution commitment, primarily relying on the DM, is overpromised that could lead to single point of failure. Secondly, we could not find evidence of any kind of analysis conducted by the team or DM to identify the reasons to prevent or reduce the large amounts of bugs and support tasks. The problem also came us in his statement when the DM says, "*One thing I have been missing is a sort of analysis of different cases*". He further described the analysis as verifying the quality and impacts of all possible solutions to select the best one. Since the impacts of the solution "*normally comes out later*" as an issue that might require additional work to fix that. The code review session is another point when this kind of analysis and understanding is done, however, a deeper analysis should provide a better solution to reduce future bug occurrences. Reduction in the amount of bug-

related tasks would potentially help in cutting down the workload on the team, particularly on the DM, and will create opportunities for efficient team management.

4.5.2.3 Analysis of coordination mechanisms

The DM is extremely responsive and physically available for most of the local coordination needs; therefore the dependencies are managed without any issues. Besides, a variety of coordination mechanisms are used to manage the dependencies on the DM. For example, the scrum coordination meetings such as daily standup, sprint planning, sprint review, and sprint retrospective meetings serve as points of information exchange with local team members. The code review and test case review sessions are also useful mechanisms used to share expertise knowledge and develop shared understanding. Apart from these meetings, team members personally communicate with the DM for any kinds of information regarding their tasks and issues. These informal face-to-face meetings are effective to manage knowledge and resource dependencies.

"*If I can't understand any specific area, I will go to [DM] and ask him. Then he will guide me and give me some starting points then we can resolve*" [PDV4]

Emails and conference calls are the main mechanisms used by the distributed teams to coordinate with the DM. Weekly meeting conducted over conference call is another coordination mechanism where distributed teams discuss issues and make decisions by consulting with the DM.

4.5.2.4 Challenges

Though the above coordination mechanisms are currently working well, their effectiveness are dependent on the DM's availability and timely responsiveness, which is a real risk of coordination breakdown and a single point of failure. Reduced availability in case of knowledge and entity dependency, or poor responsiveness in decision making and work completion will have significant impacts on the timely completion of tasks. Such impacts are further amplified when it involves distributed

teams (time zone differences) and there is a high volume of information and time-critical decisions that strongly rely on the DM. Due to the complexity of this work dependency, impacts such as delays can have a snowballing effect when multiple teams are involved, and the work of those teams might impact other teams and so on. There is no evidence of such impacts in the current process because the person is taking a lot on his plate. However, this critical dependency situation is not sustainable for effective coordination.

Emails are also prone to miscommunication due to a lack of the level of shared understanding and language used. Conference calls are not effective due to a lack of body language and emotions which makes it difficult to engage and feel isolated.

"*It's harder because you can't see them, I can't look at them, and tell and read them in any sense*." [PSM1]

4.5.3 Dependency 3: Inter-module code dependency

Another predominant dependency that was identified relates to high functional overlapping between modules, for example, between replication and communication modules due to the cascading dependency effects of the code changes made in one module on the others. A deeper analysis of the dependency to understand the associated issues and challenges is presented in the following sections.

4.5.3.1 Dependency Description

Inter-module dependency occurs when the changes made in one service module impacts the functionality of other dependent modules. As previously described in the product architecture in section 4.1, strong inter-relation exists between different services due to the logical structure of the data. This relationship leads to the cascading dependency situation where changes in one service module may have effects on services in other modules. Since each team is responsible for a single service module, the module inter-dependency leads to reciprocal dependency between teams. For instance, Team R has the most severe form of code dependency with Team

C located in US because of strong overlapping in the service level functionalities, "*we do it on top of framework, and Team C I guess takes the other half... of the backend stuffs"* (PDV1).

This dependency can be further categorised as a 'Technical' dependency since both the teams depend on the technical aspects of the backend services that should complement the changes on each side. This inter-module code dependency has a domino effect on several parts of the product development process and in most cases, in the form of bugs that may impact the work progress of different teams. For example, a code change in one module that is linked to other modules could potentially break the build that can block the integration of the respective team's work output. This is a global dependency situation that requires efficient collaboration to avoid coordination breakdowns.

4.5.3.2 Dependency issue and impact

This dependency has a high potential to hinder the work progress of multiple teams and create significant delays in the overall development process. The majority of the study participants reported that a lot of bugs that they handle often resulted from a single change made by Team C. Since Team R and Team C work in the functionally overlapped modules, code changes on either side could potentially break each other's build.

"*Sometimes other teams' bug which causes our stuff to break and then we get a lot of bugs, but it's all one cause of other team's issue. Sometimes you can't deploy your fix*' [PDV1]

The criticality of such a bug is that it is difficult to detect whether the bug is created by the changes made by their own code or by others. In some cases, it took several days to detect the issue, only after which the team can communicate with Team C and then start resolving the bugs – this blocks the ongoing work progress and create delays. Moreover, since most of these bugs are coming from the US-based Team C,

geographical and temporal aspects can create additional delays. Some participants conferred that it might be easier to fix the bugs if the team was co-located. The difficulty involved in solving the bugs by coordinating with distributed teams is illustrated using an example below where the delay can range from 3-4 days (shown in Figure 4.20).

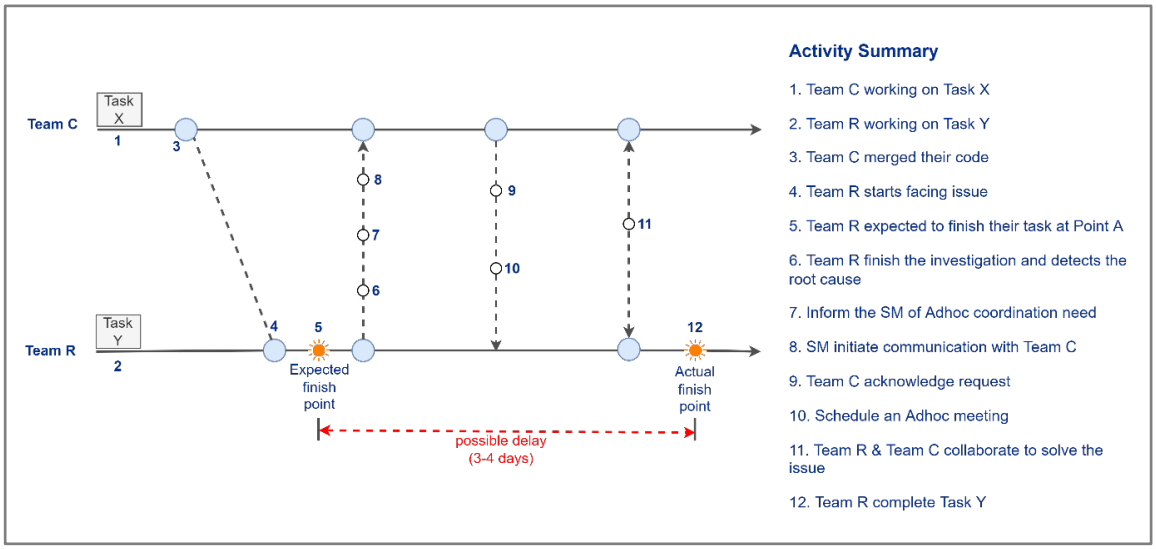

Figure 4.20: Example delay scenario because of inter-module dependency

The delay caused by this reciprocal dependency has several impacts. Firstly, if identification of the root cause takes longer, then communicating with distributed team and solving the bugs requires additional time. This will impact the release schedule and can result in under delivery of release goals. Secondly, it can cause delay in fixing client issues. Code changes made by other teams' can cause unanticipated bugs in Team R side resulting in breaking some of the working features – this can further prevent Team R from deploying their code changes to solve critical client issues. This delay potentially prolongs the client-side interruptions and severely impact the organisation's reputation.

The two primary reasons that contribute to this issue are: 1) lack of understanding about the code inter-dependency and 2) lack of awareness of possible impacts across the development teams and their members. Developers do not understand the correlations between the module and services, and hence, could not anticipate the possible effects of any code changes on other sides. While the lead developer

acknowledged the significance of this understanding (that it would make the developers more confident about their changes), he also recognised that the size of the product made it difficult to capture all the interdependencies. Moreover, Team R members do not know what other teams are working on and how their work may impact other teams work until they face a specific issue,

"*We didn't know what's other team were doing, until they have been blocked by us. it takes a while to figure out the problem, to then introduced the effects…*". (PSM)

4.5.3.3 Analysis of coordination mechanisms

Several coordination mechanisms are used to support this inter-module code dependency. Few mechanisms, such as scheduled and Ad-hoc team synchronisation meetings, facilitate the development of shared understanding about the inter-module code dependency and other teams' work activities, and the rest, e.g. email, Jira, are used to communicate and collaborate to resolve any issues caused by this dependency.

Weekly 'Tech Sync meeting' is the primary mechanism used to synchronize between the development teams to share their work activities and collaboration needs. The meeting provides an opportunity for the team members to know what other teams are working on, their expertise and how their work fit into other team's work, which helps them to understand the impacts of the changes they made. It also helps to raise issues that may require the involvement of other teams. Since this tech sync meeting is conducted using conference call and screen sharing, the key challenge of this distributed coordination mechanism is to maintain continuous attention of the participants. If it is not of their interest, people are less likely to pay attention to what other teams are talking about and as a result, fail to capture other team's activities and expertise that will help to resolve inter-related work issues.

Email and Jira are the secondary methods of communication to gather information about other teams. Although emails are frequently used in communicating with external

teams, participants undisputedly acknowledged that it is not effective as this asynchronous mechanism lacks the level of shared understanding and sometimes difference in language creates misunderstanding that provokes people to make assumptions. Therefore, extra efforts are required to develop equal understanding to abate the confusions and presumptions which requires additional time that causes delay.

To coordinate the dependency issues, Team R depends on emails and Ad-hoc conference meetings organised by the SM. The SM acts as 'Boundary spanner' to coordinate the Ad-hoc communications with external teams, mostly between Team R and Team C. Once requested, she initiates the communication over email and schedules ad-hoc meetings to discuss and resolve the issues.

4.5.3.4 Challenges

As indicated by the SM, temporal difference is the primary hurdle faced in this ad-hoc distributed team coordination. She quoted that "*the hardest ones are the ones those we can't talk to on the phone, because of time difference*". Besides, due to lack of body language over conference calls, it is difficult to convey the urgency of the situation and engage teams which ends up not giving enough time to solve the conflict. The lack of personal contact with global teams causes low priority in request processing which creates uncertainty in information flow and sometimes difficulty in obtaining key information. If this situation occurs several times per week, it will create unnecessary delays. Furthermore, we have observed 'Territorial (combative)' mindset if the bug crosses international boundaries which makes the coordination challenging. Because of this mindset, distributed teams tend to give higher importance of their works than that of others which creates tension and conflicts between teams, and in extreme, requires interference of higher authority.

Regulating coding standards and community of practices across the teams could help in alleviating inter-module code issues which was lacking in this case. For example,

Team R uses separate branch for local changes and merge their codes in the final branch after successful testing for better code manageability and to avoid code conflicts in the other modules. Though this should be the norm for all other development teams, it was not clear what norms other teams follow which reflects the lack of regulations around code management that also challenged this dependency coordination.

4.5.4 Dependency 4: Work-output Dependency between Testing and QA

Though the dependency between testing and QA activities is well-known, it was identified as a work-output dependency due to its impact on the progress of the software development activities relates to the dependencies. The misalignment of work cycles of the testing and QA, and overwhelming work pressure on the testers were found to the main issues leading to coordination problems.

4.5.4.1 Dependency description

In this dependency, the work of the tester depends on the work-output of the developers and similarly, the work of the QA depends on the work-output of the tester. For instance, in Team R, all the requirements implemented by the developers need to be tested locally before it goes to the global QA person. In the current development cycle, Team R's tester starts testing only after the developers have completed all their sprint development works which follows to another sprint. After testing, the work is forwarded to the overseas QA for checking and approval before it can be released.

"*The local tester is testing thoroughly all the technical works, a bit low level. After its passed, we are confident about what we did is correct. then we are giving to the other (offshore) testers, so they will do some sort of regression and integration testing*" (PDV4)

The following sections present a deeper analysis of the situation to understand the issues and impact of this dependency.

4.5.4.2 Dependency Issue and Impact

At the time of the interviews, the development and testing activities were done in separate sprints where the work completed by the developers in a current sprint were tested in the next sprint. Since Team R was using 2-weeks sprint, it required at least 4 weeks cycle to complete the development and testing before going for QA approval. The global QA work cycles were not aligned with the local development and testing cycle. Lack of coordination between these three groups until after a large amount of work was done in a sprint had major consequences as discussed below.

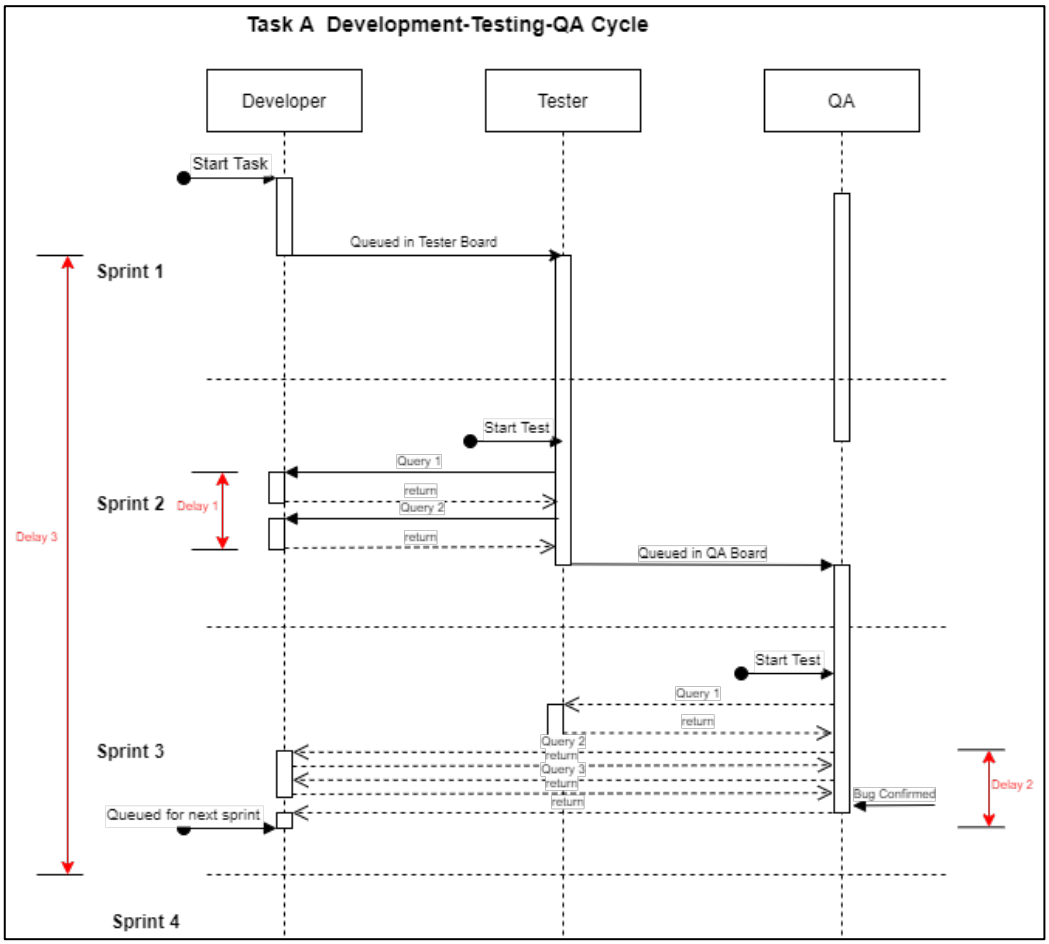

Figure 4.21: Delay diagram for work-output dependency on Tester and QA

Firstly, the amount of work that need to be coordinated required multiple iterations of information exchange and feedback loop between the tester-developers, and the QA-development team (i.e. developers and tester) which had the potential to create delays, e.g. delay 1 & 2 as shown in Figure 4.21. For example, the tester shared that, prior to

testing, he needed to know the details of the changes that were made by the developers and that the frequency and amount of information exchange depended on the complexity of the work, "*the more complex the work, the more I need to talk with the developers*". This statement implies that the developers had to stop the work in their current sprint and go back to the previous sprint to be able to respond to the tester's queries. This not only interrupted both the developer's and tester's workflow creating additional delays, e.g. '*delay 1*' (Figure 4.21), but the overwhelming amount of workload on a single tester against 5 developers of the team creates a high risk of activity bottleneck leading to work completion delays.

Likewise, the QA requires additional information either from the tester or from the developer during their work cycle which is almost four weeks after the development has been completed. The current process is susceptible to two types of delays. Firstly, it is difficult for the developers and tester to promptly respond to the queries due to the long gap (almost a month gap) between the work completion and the request. It takes time for the developers to recall what they have done to respond to QA's query; hence there is an information processing delay. Secondly, the time zone difference and lack of continuous availability of members from either side cause additional delays in this information exchange cycle as depicted in '*delay 2*' in Figure 4.21. Such delays can block the QA work progress and thus may impact the release deliverables.

Another consequence is that if there is a bug found either on the testing or QA side, the development team need to reschedule the fixing of that bug that may defer until the next sprint which also contributes to some additional delay. For example, if QA finds a bug in **Task A** that is being scheduled in a two-week sprint, tested in the next sprint, and the QA is performed in the third sprint, which would cause 6 weeks delay to re-schedule and fix that bug (e.g. '*delay-3'* as shown in the Figure 4.21).

The issues discussed above have predictable impacts on the release deliverables. The prescribed 4- months release cycle (i.e. equivalent to 8 sprints) does not harmonise

with the development, testing and QA cycles which makes it challenging to successfully release the developed features. For example, the user stories developed in sprint 1 are tested in sprint 2 and QA is performed in sprint 3. Subsequently, the testing and QA will not be finished for all the user stories developed in sprints 7 & 8 as shown in the following Figure 4.22. Therefore, those user stories must be either crossed out from the current release or have to be released without testing and QA approval. Consequently, there is a high chance of under-delivery or bugs in the release deliverables.

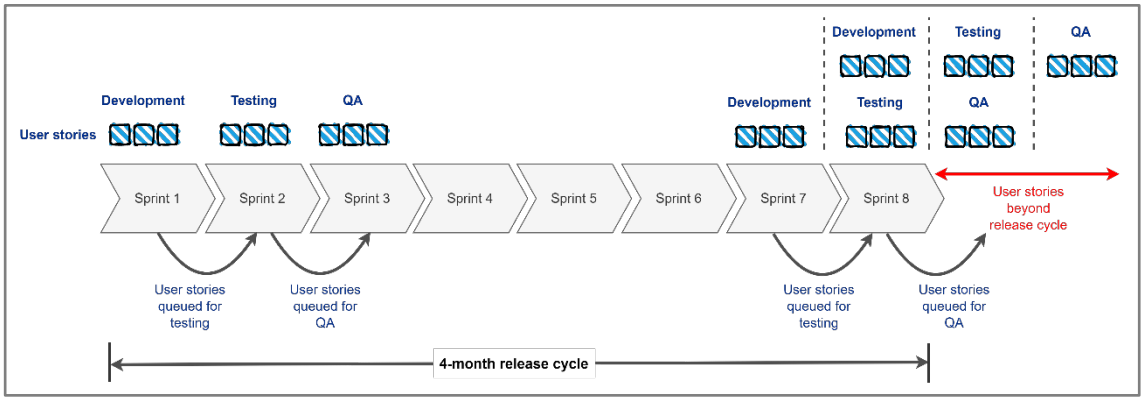

Figure 4.22: Release Cycle vs. Sprints Cycle

There can be several ways to manage this dependency. For example, the same work dependency where the work and coordination cycle are broken down into smaller pieces and the same with the tester and QA cycle that all could be done in much quicker feedback loop cycle. For instance, if the developer completes one user story, then got tested and then the QA looked at it, all in the same sprint period would be a better way to handle the dependency issues. If required, lengthening the sprint to 3 weeks cycle to fit all the activities within the sprint could be another way to improve dependency management.

There are several ways to improve the work cycles that could reduce the delays discussed in above scenarios. As shown in Figure 4.23, integrating the development, testing and QA activities in the same sprint cycle could be a potential solution that will have minimal delays in the testing and QA cycle.

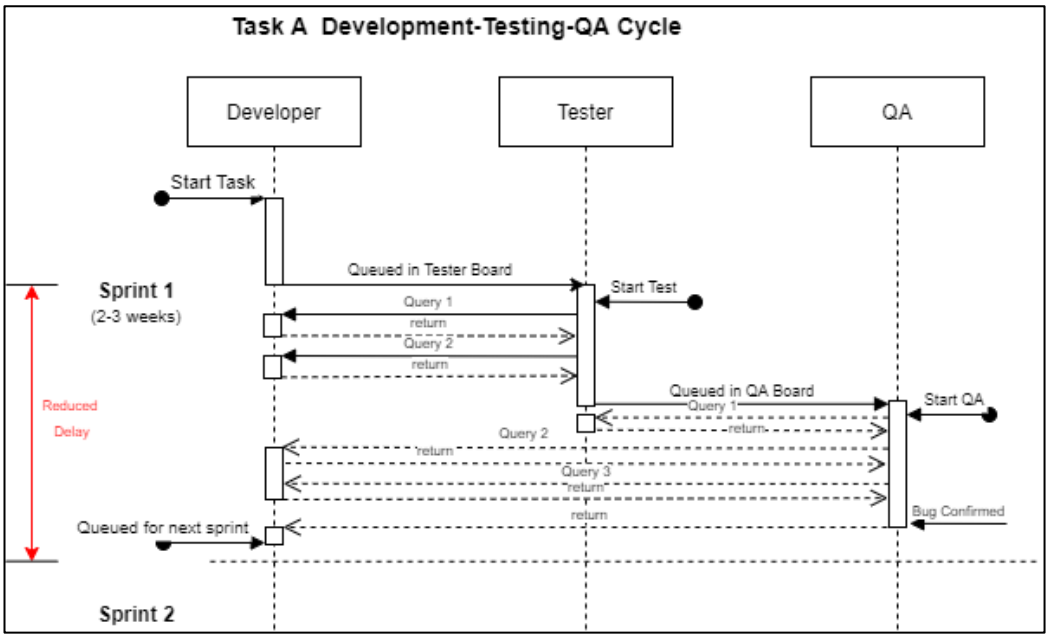

Figure 4.23: Reduced delay after altering the Development-Testing-QA cycle

One of the benefits of this approach is that the information processing and feedback loop will be quicker as it is easier for the developers to remember what has been done and respond back to queries quickly. Similarly, the testing and QA activities will be faster compared to former approach because of the smaller size of the work required. This approach will not be interrupting the workflow in the next sprints as it is having in current approach. And if there is any bug found, it would also be easier to resolve it within the ongoing sprint or in the next sprint that will reduce almost two to four weeks delay as presented above.

4.5.4.3 Analysis of coordination mechanisms

This work output dependency is currently managed in multiple sprints where all the development tasks completed in a sprint are tested in the following sprint, and finally, the successful test items are passed to the global QA team for approval. All the tasks are recorded and managed electronically using Jira and separate boards are used for development and testing activities. All the development works in a sprint are queued in the '*Testing board*' in Jira and information exchange and feedback are primarily shared over the email and Jira comments.

Slack is frequently used to track daily work updates and issues that are discussed in 'Daily Standup'. Occasionally, '*Confluence'* is used to store and share documents such as developer side 'How to' articles, test plans, and list of user stories.

The QA team uses the same testing board as the local tester for their work and all the user stories marked as 'complete' from the tester becomes available for QA to check. The QA team can see the comments and documents related to the user stories shared by the tester. Since the QA teams are external to the organisation, restricted access is provided to them, which limits the visibility of comments and other related details. Therefore, QA members need to communicate with the tester or the developers for their queries. Emails and comments on the Jira tickets are used for regular communication between the global QA and local development team.

There is a scheduled fortnightly meeting with QA teams where the development teams and tester coordinates with QA leads. The meeting is conducted over the conference call and WebEx for screen sharing whenever needed. Recently, Team R introduced '*QA Demo*' sessions during this meeting to showcase the developed features and explain how it works. This demo session seems effective as it supports the shared understanding to avoid misunderstanding and creates an opportunity for further discussion and information exchange. Unfortunately, it was difficult to evaluate the perceived benefits and challenges of this session since it was a newly introduced practice in the coordination process.

4.5.4.4 Challenges

The primary challenge of this dependency coordination is the way the development, testing and QA activities are executed. As discussed, testing and QA activities are not synchronised with the development activities resulting in a significant time gap. This significant time gap (i.e. approximately five weeks) creates inconvenience for the developers to remember the past development work, process them and respond back which can interrupt the information flow and further delay related activities. The time

difference and continuous availability requirement is another concern for the global coordination with the overseas QA team. In case of any Ad-hoc communication, either Team R must stay late, or the QA team member needs to come early to compensate for the time difference which is challenging for both sides. Besides, the overwhelming amount of testing requirements for a solo tester and slow performance of the test environments are additional coordination challenges identified in this case that need to be addressed.

4.5.5 Dependency 5: Task Knowledge Dependency

Task knowledge dependency is a sub-type of knowledge dependency that was strongly singled out in the analysis of the dependencies. Task knowledge dependency is a specific form of knowledge needed to perform development tasks, that could be purely technical or domain-specific, or situational, i.e. what is going around related to tasks. There are several other forms of task-related knowledge identified from this case that is categorised under this dependency which is summarised in Table 4.6.

These task-specific knowledge dependencies are linked to most of the critical dependencies discussed above, hence it has significance in the effective coordination of critical dependencies. For example, knowledge of task allocation, task ownership, and a fair understating of the task interdependence is required to coordinate the inter-team work-output dependencies that arise in the development and bug fixing phases. Without knowing what other team is working on, how their work can affect other teams and when another team needs to be involved in the current work, it would be difficult to coordinate their work.

Similarly, a significant amount of knowledge dependency on the DM is related to task knowledge that is required in different stages of the process. For instance, during the sprint planning session, DM discusses the purpose of the user stories (i.e. know the purpose) that creates a shared understanding of the tasks that need to be done and identifies probable solutions.

Table 4.6: Task Knowledge Dependency Categorisation:

Primary Coding	Definition	Example of data	Explanation
Why doing the task	a situation wherein the purpose of doing the task is not known and this affects progress	"*He might come and say: 'It doesn't work', because he might not have understood the reason of doing that, or run something wrong, or hasn't apply to some data properly*." (PDM)	The developer needs to understand the purpose of any task (dependency) that he is working on; otherwise he will not figure out how to complete the task properly. The DM discuss with the developer (mechanism) to clarify the purpose of the task with some probable solutions.
knowledge of blockers	a situation wherein there is lack of knowledge about work blockers and this affects progress	"*Issues that we are having, to alert people about critical client issue, opportunity for Jim's team in US to list up issues he has got, then need coordination between a lot of teams.*" (PDM)	During the 'Tech sync meeting' (mechanism) teams share about the blockers that other teams might face. Without knowing about any possible blockers, teams would not know which team to contact to solve this issue.
Task prioritisation	a situation wherein the priority of each task is not known and this affects progress	"*Somebody else has got information and says you might want to look at this. or I came up with that before, so actually, you should be working on that because it's something new that's a high priority and we need to resolve it*." (PSM2)	The whole team comes together every morning for daily standup (mechanism) where they check their work priorities (dependency). Without this checking the team would not identify the high priority tasks.
Task progress	a situation wherein the progress or status of relevant tasks are not known and this affects progress	"*There is also some sort of status page that I update weekly, that information also goes out into the Digest that produce. This is a way of communicating the status of the things but it's not a meeting.*" (PSM2)	The SM creates a weekly digest from confluence and Jira that includes the status updates of tasks (dependency) which is shared via email (mechanism) to the group of managers. Without this update, teams will not be aware of the updates of dependent works.
Task allocation	a situation wherein the allocation of each task (i.e. who is doing	"*We try to negotiate between the teams who's doing what and how are*	During the weekly tech sync meeting (mechanism), The teams gets to know about

	what) is not known and this affects progress	*they gonna interface with each other.*" (PSM2)	which team is doing what (dependency) which helps the teams to identify which the team they need to coordinate for their work.
Task ownership	a situation wherein the ownership of each task is not known and this affects progress	"*We are having multiple teams could be involved in developing a feature, one person has got ownership because otherwise we end up with not only these gaps in the Silos of developing*" (PSM2)	Teams need to know the owner of any joint task (dependency); without knowing the task owner teams will not be able to communicate and coordinate the joint work properly. This temporary ownership has been a challenge since there is no proper division of responsibility exists.
What has been done	a situation wherein past actions related to a task is not known and this affects progress	"*First I need to know from the developers what are changes they did, how did they do the developer side testing, what are the objects I need to look at, what are the field I need to look at*." (PTS)	The tester attends the developer side meetings such as code review sessions (mechanism) to understand what has been done (dependency); without this information he would not know what changes and testing are done and what things need to be tested.
What to do and when	a situation wherein the schedule of each task is not known, and this affects progress	"*Actually, when we are doing the Sprint Planning, at that point, we know what need to be done and discussing at that point... we have some understanding over what we are going to do*" (PDV4)	In the sprint planning session (mechanism), all the team members discover and plan the tasks that needs to be (dependency) and schedule their works; without this understanding the team would not be able to estimate and schedule the works.
Task inter-dependencies	a situation wherein the dependencies between the tasks are not known and this affects progress	"*Do QA know what they are doing, do they understand it, are we blocking QA? and so then there is a role they have in the end*" (PSM1)	Teams need to know how their work depends on or impacts other teams work (dependency); without this understanding, teams will not engage with other teams to coordinate their work. There is no mechanism found to properly develop this understanding.

Without this information, "*stories are not really practical...*" and it is easy for the developers to "*get stuck*" or choose the "*wrong path*" as indicated by the DM. Therefore, having effective mechanisms in place for developing a good understanding of the task-related information is vital for coordinating the development and testing activities.

4.5.5.1 Analysis of Coordination mechanisms

Several practices in the current SD process are supporting the task knowledge sharing between the local and distributed team members. The group meetings including the distributed team members create opportunities for exchanging various types of task knowledge. For example, the weekly technical synchronisation meeting is the place where development teams share the recent client issues that other teams might counter (i.e. blockers), notify the task interdependencies that needs coordination between distributed teams.

"*We have touchpoints in terms of the 'Tech Sync Meeting' on every Thursday, that syncs up with other development teams. Issues that we are having, to alert people about critical client issue, opportunity for the team in US to list up issues, then need coordination between a lot of teams.*" (PDM)

This meeting is also important to gather other sorts of task-related knowledge such as which team is working on which task (i.e. task ownership), new ideas and development changes (i.e. what has been done). The DM shares these meeting outcomes to his team over email so that they are aware of the task knowledge which helps them to develop situational awareness.

Similarly, the pre-scheduled regular meeting with the support teams is good for gathering knowledge about support-related tasks, their progress and how they were being handled (i.e. what has been done) which facilitates confirmation of the task execution process. Besides, most of the agile practices are good points for the team's internal task knowledge sharing, e.g. sprint planning sessions for knowing why doing

the task, how to solve the task, who is being assigned to a task, and daily standup meetings for knowing tasks progress and blockers.

The majority of the local task knowledge sharing mechanisms are working well since Team R members are well coordinated by regular synchronisation meetings. However, there is a lack of mechanisms for globally distributed inter-team task knowledge sharing which is affecting the inter-team coordination process. For example, there is no mechanism found for allocating or assigning task ownership which impacts the identification of interdependent tasks and hence, failed to involve necessary teams earlier in the process. Additionally, the inter-team coordination conflicts discussed in section 4.5.1 are also linked to the lack of task knowledge sharing between the development teams. Therefore, effective mechanisms need to be applied to resolving issues resulting from task knowledge dependency.

4.6 Interpretation of the Findings

In this study, we focused on understanding '*How are the key dependencies coordinated effectively in DASD?*'. For the Pigeon case, this understanding is gathered from the SD activity and coordination process analysis. From these analysis findings, five key dependencies with associated risks are outlined. The coordination mechanism analysis of these dependencies highlighted several lacking and coordination challenges. In this section, these findings are discussed in detail that further contributed to proposing improvement ideas for effective coordination.

4.6.1 DASD Dependencies

Overall, the above discussed dependences are categorised into three high-level dependencies following Strode's dependency taxonomy (D. E. Strode, 2016). According to the categorisation, dependency 1 and 6 are the grouped under activity dependency as both these dependencies represents the work-output dependency which refers to a dependency situation where an activity requires a timely output from a previous activity to be completed. The work-output dependencies on the DM (part of

dependency 2) are categorised as Business process dependency since the bug reviewing and categorisation activities performed by the DM need to be carried out in certain order (D. E. Strode, 2016). All types of information and expertise dependencies on the DM (part of dependency 2) are categorized as knowledge dependency. Additionally, the task knowledge dependency (dependency 5) is also categorised as Knowledge dependency since several types of task-related knowledge are required to properly perform the development activities which could potentially affect the project's progress. Finally, the authority dependency on DM (part of dependency 2) and inter-module technical dependencies are categorised under the resource dependency following the dependency taxonomy. The dependencies categories are summarised in Table 4.7 with their cause and possible impacts to supplement further discussions.

It is noticeable that two of the most impactful dependencies involve globally distributed teams, which is a contemporary challenge for coordination (Ekasari et al., 2022). Another interesting finding is that all three high-level dependencies involve a single resource person which in conjunction represents an additional dependency, i.e. *Entity dependency* where the resource person needs to be available and responsive otherwise there is a high risk of coordination breakdown.

The analysis findings also indicated that three of the dependencies (i.e. Dependency 1, 3 & 5) primarily involved coordination between globally distributed teams. Another two (knowledge and resource dependency on DM discussed under dependency 2) indirectly involved coordination of globally distributed teams. For example, being the point of contact and primary decision-making authority of Team R, all the distributed teams need to communicate with the DM for their queries and work needs, and his availability and responsiveness might have an impact on other teams' work coordination. These findings could be interpreted as the level of coordination between globally distributed teams potentially having higher impacts on the DASD process if they are not well-managed.

Table 4.7: Key Dependencies, their causes and Impacts

High-Level Dependency Type adapted from Strode (2016)	**Pigeon Dependencies**	**Potential Impact/Risks**	**Cause of the Issues**
Knowledge Dependency	Knowledge dependency on DM (Dependency 2) Task knowledge dependency (Dependency 5)	-High risk of Knowledge acquisition bottleneck - Delay in work completion due to unavailability or late response - Delay caused by additional work resulting from incorrect assumptions	- Most of the task and expertise knowledge is confined to single resource person (i.e. DM) - Unavailability or late response to queries
Activity Dependency	Work-output dependency between Development teams (Dependency 1)	-Interruption in scheduled works and under-delivery of release goals - Poor Quality Work Output may lead to technical debt and rework - Frustrations between teams	No mechanism for early identification and notification of dependent work
	Work Output Dependency on Testing and QA (Dependency 4)	- High risk of work completion delay in testing and QA cycle - High chance of Under-delivery of release deliverables - Poor quality work output from tester and QA - high risk of activity bottleneck at tester	- Ineffective work cycle for development, testing and QA activities - Overwhelming amount of workload at a time on tester and QA
	Business Process dependency on DM (Dependency 2)	- High risk of deteriorated performance of DM due to burnout - Probable risk of detriment to team performance	- Exceptionally burdened with responsibilities associated with multiple roles - Lack of clarity in development resource distribution
Resource Dependency	Entity dependency on DM (Dependency 2)	- High risk of Single point of failure - Potential risk of failure in time commitment and delay in critical client issue fixing due to unavailability or low responsiveness	Solo decision-making and task allocation authority for certain type of work

Resource Dependency	Technical Dependency between Modules (Dependency 3)	- High potential to block work progress of development teams - Potential to interrupt release schedule or under-delivery of release goals - Prolonging client-side interruptions and hamper organisation's reputation	- Lack of shared understanding of the technical dependency - Lack of awareness of possible impacts of code

All the impacts and risks identified in this case are grouped into two high-level categories- *Delay and Poor Quality* as shown in Figure 4.24. Delays are one of the key impacts resulting from the poor or mismanagement of the key dependencies. There are several types of delays resulting from the dependencies that are grouped under two subcategories: *direct* and *indirect*. Direct delays are predictable that can create specific delays such as blocking the work progress, interrupting the release schedule, and preventing fixing critical client issues. For example, as discussed in inter-module code dependency, distributed team members' lack of understanding of code dependency could result in bugs that directly impact fixing Team R's critical client issue. Because of the possible high impacts or risks associated with them, direct delays need to be addressed immediately.

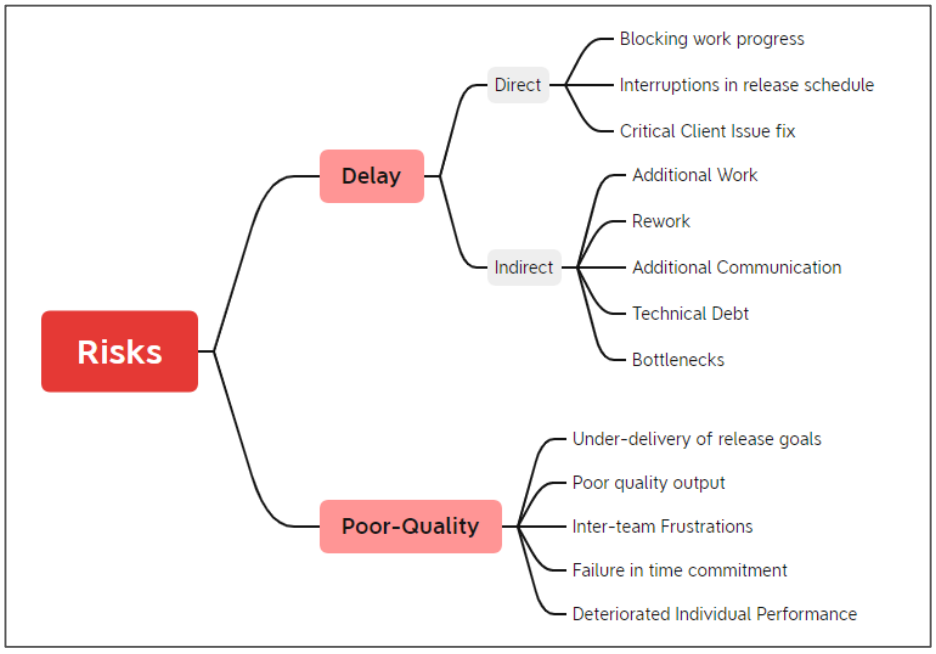

Figure 4.24: Emerging concepts of Dependency risks from study findings

Indirect delays are uncertain in nature and mostly by-product of other impacts. For example, any additional work required due to the poor-quality work outputs will cause delays. In most cases, the amount of work needed to fix the issue is uncertain and are difficult to estimate the overall delay. Compared to the direct delays, indirect delays do not block the work progress or release or critical client issue fixing, therefore, do not require immediate attention. But this types of delays are less likely to happen; therefore

difficult to predict their occurrence in advance. Therefore, the Ad-hoc mechanism needs to be applied to reduce the delay or likelihood of this type of delay; otherwise, they can potentially turn into blockers and cause high impacts. For instance, the expertise and decision-making bottleneck identified at the DM are examples of indirect types of delay that should be resolved by promoting substitutability to avoid any single point failure due to his unavailability.

Quality can be defined as, "*the degree to which a system, component, or process meets the expectations or needs of the customers or users*" (Galin, 2004). Since dependency, in general, defined as the management of needs between the activities, the quality of the output from these activities directly depend on how well these dependencies are managed. If the dependencies are not managed properly, there is high probability of poor-quality work-output. For example, the high amount of workload on the tester creating work-output bottleneck could potentially result in poor-quality output for meeting the deadlines. Besides this, poor-quality relationship between the development teams resulting from the mismanagement of work dependencies would deteriorate performance (at both individual and team levels) in terms of work commitments and release schedule. All in all, poor-quality output in any form impacts the quality of the software development process and its outputs. Therefore, it is plausible to categorise them as a risk of poor-quality output. It is also noticeable from the above two examples that risks of poor-quality output have impacts on both individual and team performance. For example, while the tester is burned out from work load, it is usual to be frustrated which will directly impact his performance. The similar is true for the teams when two teams are highly interdependent, but they have a poor relationship and regular conflicts, which will impact both teams' performance. These results are consistent with the findings of (Lindsjørn et al., 2016) which showed that quality concerns are directly linked to the performance of the teams and the organisation; therefore, they have long-term impacts on the success of the agile development teams. Since lack of proper coordination is one of the primary challenges

in achieving quality in global software development (Misra & Fernández-sanz, 2011), it is important to mitigate these issues leading to poor quality.

The causes of dependency issues discussed above are classified into three high-level clusters: *Knowledge management*, *Work management*, and *Process management*. Any dependency risks caused by knowledge-specific issues are categorised under *Knowledge Management*. For example, lack of shared understanding about any technical dependencies, and presumptions made by the developers due to unavailability or late response of any required information are related to knowledge management Slow response to the queries is one of the side effects of Knowledge acquisition bottleneck is also a knowledge management issue. Finally, lack of awareness of probable impacts of code changes in other parts of the product makes the developers less confident while making any changes, "*if you understand it, that makes the work lot more easier and you're lot more confident in your changes rather than going and changing a little bit and just gonna hoping it's not breaking anything*" (PDV2).

The work management category relates to the issues resulting from mismanagement of dependent activities. The majority of the issues faced while managing dependent works between distributed teams are caused by the lack of an appropriate mechanism for early detection of work dependencies and timely notification of the works needed from other teams. The poor management of work-output dependencies has direct impacts such as delays in the delivery schedule. Lack of clarity in the work distribution is another outcome of poor work management which potentially causes dependency issues.

Finally, the Process management category relates to issues resulting from ineffective management of the software development process. Process management activities in software development include the definition, implementation and management of the development work process within a defined organisation. The primary motivation

behind the process management is to facilitate an environment that improves the quality of the software being developed and productivity of the members involved in this process. As discussed in the findings, ineffective work cycles for Testing and QA, and hierarchical decision-making structure are the issues created by poor management of the software development work process. Our analysis indicated that inability to understand and adopt the agile mindset in the agile software development process is one of the primary causes of the problems identified in the testing and QA work cycle. All the prescribed agile frameworks suggest iterative work cycles that can deliver shippable product feature, that means all development works including testing and QA should be performed in the same work cycle (e.g. sprint). Having multiple separate sprint development, testing and QA works is creating misalignment with agile values, and hence, causing dependency issues. Likewise, following traditional hierarchical decision-making and work allocation authority is contradictory to the self-organising team values where team members collaborate and self-organise themselves to perform necessary activities. These contradictions to the agile mindset are caused by the incorrect implementation of agile process management, which is contributing to process management issues. A complete view of this classification is presented in the following concept map (shown in Figure 4.25).

This classification has twofold benefits for this research. Firstly, this categorisation will support in the development of theoretical concepts that can be used to explain any dependency issues better. For example, the knowledge management category would help in explaining the issues in knowledge creation, knowledge management and knowledge sharing throughout the SD process. The work management category would help in understanding possible anomalies in scoping, estimation, scheduling, distribution and progress tracking. Finally, the Process management category would better explain the possible causes of mismanagement in defining, implementing, and maintaining the SD work process.

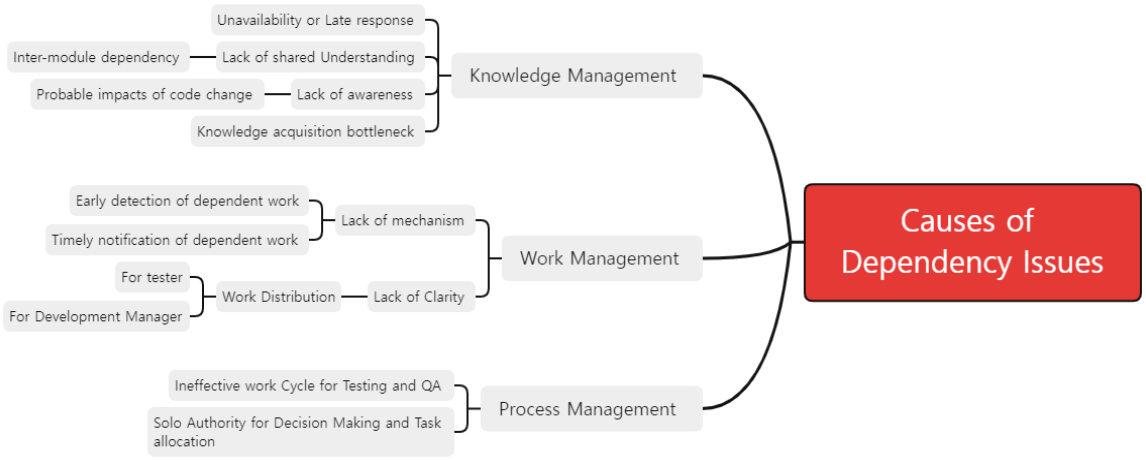

Figure 4.25: Conceptualisation of the Causes of dependency issues

Secondly, it will help the practitioners to identify the fundamental reasons of any problems that will further guide them in tackling complex coordination issues. For instance, if there is a knowledge acquisition bottleneck in the process, it is necessary to extract the knowledge from the expert resource and distribute them within the team to create redundant knowledge sources. Collective problem-solving techniques such as pair programming, mob programming (Zuill et al., 2016) can be a probable solution to quickly develop task knowledge and problem-solving knowledge which are the key knowledge dependencies n DM located in this case.

Similarly, to improve inter-team work collaboration in globally distributed setting, appropriate mechanisms should be put in place to distinctly manage the inter-team work dependencies. For example, during the release planning, an inter-team dependency analysis session could be conducted and based on the analysis, a '*Scrum Team'* could be formed including key participants from each dependent team. Additional teams can further be involved in later stages if required. This scrum team would be responsible to manage the dependent works and should conduct regular Scrum-of-Scrums meetings to coordinate the work progress. One of the benefits of this focused group meeting instead of the current 'tech sync meeting' is that it will avoid participants having disjoint interests who do not feel interested in the conversation, '*I think a lot of people don't listen… In group meetings or conference calls, you will find people who want to be alone, doing his stuff*' (PSM1).

On the other hand, adopting an agile mindset in every part of the distributed SD process would wave opportunities to resolve process management issues. For example, integrating development, testing and QA activities in the same sprint would potentially reduce coordination overheads. Likewise, encouraging self-organisation practices within the development team would help in managing workloads without any intervention of a solo decision maker.

4.6.2 Coordination Mechanisms

All the coordination mechanisms used to manage the critical dependencies are presented in Table 4.6 using *Coordination activities*, *Coordination tools and artefacts,* and *Structure*. Coordination activities include all the activities that are performed to manage both local and global dependencies. Coordination tools and artefacts are used to support the coordination activities. The structure represents the *Availability*, *Proximity* and *Substitutability or Specialisation* of team members that is part of the coordination.

All the coordination activities that are used to manage the dependencies can be divided into two subcategories: *Synchronous* and *Asynchronous*. Synchronous activities require all the participants to be available at the same time, either pre-arranged or Ad-hoc basis. The frequency of these synchronous activities can be per project, per sprint, daily or Ad-hoc. For example, the 'Tech Sync meeting' is the primary mechanism to coordinate between the development teams that is a pre-scheduled group meeting conducted every week. On the other hand, sprint planning session is a f2f group meeting conducted at the beginning of the sprint to plan the sprint activities with the team members. Asynchronous activities do not require on-the-spot availability of the participants and are used to communicate, exchange information, and perform control and process management activities. The frequency of these activities is the same as the synchronous activities: per project, per sprint, daily and ad-hoc, but this form of activity is more common in day-to-day coordination.

The coordination activities mentioned above are supported by several tools and artefacts. Different varieties of tools are also used while communicating and sharing artefacts to support the coordination activities. For example, WebEx is used for conference call and sharing the screen during the synchronous group meetings which facilitates shared understanding. Electronic Story boards are used as information radiator within the team to track progress and facilitate task awareness, i.e. what is going on, who is doing what, what to do and when etc. This types of boards are also used to synchronise inter-team work dependencies. The artefacts that are used in coordination are either produced or used by the coordination activities. Different forms of artefacts are being used in this case, for example, user stories, Issue tickets, storyboard, developer written 'How to' docs, and software code.

Organisational Structure is considered to be part of coordination mechanisms because it decides the arrangements of, and relations between different parts involved in the SD process (D. Strode, 2012). Proximity, Availability and Substitutability are three antecedents suggested by Strode that contributes to the coordination. *Proximity* refers to the closeness of the parties involved in any dependency situation; for example, the development manager is in Close Proximity with other team members, while Team C and Team R are not in close proximity since they are geographically and temporally dispersed from each other. *Availability* ensures the continuous presence and responsiveness of any resource whenever needed. Though continuous availability and responsiveness are particular concerns in distributed coordination (Ramesh et al., 2006b), the findings of the current study indicate that it is equally important to manage local coordination needs. For instance, to coordinate the dependencies involving the Development Manager, his availability and timely response can make a difference to achieve release goals. Finally, *Substitutability* is achieved when there is an existing alternative(s) in the team to fulfil the expertise and skills requirements to achieve project goals. Collective decision-making and cross-functionality within the team is a recommended practice in Agile software development that encourages Substitutability

(Hoda, 2011). In this case, there is no Substitutability present for the Development manager in the team to perform decision-making and task allocation activities or respond to expertise and skills requirements which is creating a high risk of bottlenecks and delay. It is interesting to note that, based on the feedback of this analysis to the organisation, the role of the DM has been modified to mitigate some of the dependency risks.

The analysis of the key dependencies highlights that not all the coordination activities, tools and artefacts, and structure applied in this case is effectively managing the coordination needs. The decision rule applied to measure the effectiveness is "*Does the current coordination mechanism(s) reduce or eliminate the risks?*". Based on the answer to this decision-rule question, we have categorised the mechanisms as either 'Effective' or 'Not effective' as indicated in Table 4.8. Additionally, we have used explicit and implicit components from Strode's coordination effectiveness model (D. E. Strode et al., 2011). We will explain the evaluation process using an example.

For instance, the pre-organised synchronous group meeting is the primary mechanism used to coordinate the inter-team work priorities (Dependency 1). Our analysis indicates that this mechanism is *effective* in managing competing work priorities between the development teams because the team representatives (DMs, SMs) are present (explicit components) in this meeting to inform, acknowledge and discuss their work priorities (implicit components). If required, the Product Manager (i.e. right person) is also involved in this meeting to resolve conflicting issues. The Frequency of this weekly meeting is well-suited to avoid any delays. On the contrary, there is a persistent problem identified in this dependency is timely identification and notification of any dependent work. The existing mechanisms, i.e. weekly tech sync meeting, is 'not effective' to resolve this issue because there are no particular measures taken by the teams to manage this issue. For each dependency, we have presented the outcome of the evaluation of coordination effectiveness along with the coordination mechanisms in Table 4.8.

For better understanding the coordination mechanisms, the data of the second column of Table 4.8 is further broken down into three tables covering three dimensions: '*Tools & Artefacts vs. Coordination Activities'* in Table 4.9, '*Coordination Activities vs. Tools and Artefacts'* in Table 4.10, and '*Coordination Activities vs. Roles involved'* in Table 4.11. Table 4.9 presents all the tools identified in the process to support different coordination activities that illustrates the possible applications of each tool in different coordination activities. For example, Jira is the most important coordination tool that has been involved in most of the coordination activities while coordinating the key dependencies. User story is the main shared artefact used in the coordination process which is the representation format of the user requirements. User stories are being used from planning to release and play an important role in the coordination of the development activities. The storyboard is another important shared artefact that is an important radiator of task-specific information, such as what needs to be done and when, who is doing what, what is going on, is there any impediments. Teams could coordinate their work in daily sprints and review their performance based on this information.

The second table (Table 4.10) presents slightly different perspective to show the tools and artefacts involved in a specific coordination activities. For instance, Sprint planning is group planning activity conducted before every sprint that primarily uses Jira as an online tool to share and manage artefacts such as user story, electronic story board, product backlog and sprint backlogs.

Table 4.8: Coordination Mechanisms, their effectiveness, and challenges

Dependency Code	Coordination Mechanisms	Evaluation of Coordination Effectiveness	Coordination Challenges
Dependency-1	Coordination activities: Synchronous Scheduled and Ad-hoc group meetings via conference calls, Emails, Asynchronous communication via Email and Jira, Ad-hoc communication for Boundary Spanning activities Coordination Tools and artefacts: shared user stories and tickets in Jira, WebEx for screen sharing, Skype for Business Structure: Distributed teams both on-shore and off-shore in different time zones, SM as Coordinator	- Scheduled synchronisation meetings are effective in managing competing work priorities among teams - Not effective in early identification and notification of dependent works to prevent delays due to work interruption and team conflicts	- Difficulty in Distributed teams communication due to time difference, rigid mindset across teams - Miscommunication due to communication channels and language barriers
Dependency-2, 3 & 4	Coordination activities: Face-to-face formal and informal meetings for local coordination, Hierarchical task allocation and decision making Asynchronous communication Email and Jira Coordination Tools and artefacts: User story, Backlogs, Electronic Squad boards, Jira tickets, Developer written 'How to' docs WebEx for screen sharing, Skype for Business, Slack Structure: Co-located team members, Distributed teams both on-shore and off-shore in different time zones, Continuous Availability required for DM	- Local coordination needs are effectively managed, but it is subject to the high *proximity* and *availability* of the DM - Not effective in managing expertise and decision-making *substitutability* to avoid knowledge acquisition bottleneck and single point failure - Not effective in developing shared task knowledge - Not effective in workload balancing and address risks related to team performance and nurturing	- Changing the hierarchical structure of decision making and task allocation - Resolving the central dependency structure

Dependency-5	Coordination activities: Synchronous scheduled group meeting via conference calls and screen sharing, Asynchronous communication via Email and Jira, Ad-hoc communication for Boundary Spanning activities Coordination Tools and artefacts: shared user stories and tickets in Jira, shared code management using TFS, WebEx for screen sharing, Skype for Business Structure: Distributed teams both on-shore and off-shore in different time zones, SM as Boundary spanner	- Synchronous meetings are effective to develop shared understanding about other teams, their expertise and tasks - Communication channels are not effective to resolve ad hoc needs and are susceptible to creating further issues	- ineffective communication channels is hindering the shared understanding - Misunderstanding in the team level about the process, technology and tools - 'Territorial (combative)' mindset between distributed teams - Non-technical boundary spanners to resolve technical issues
Dependency-6	Coordination activities: Consecutive sprints for development, testing and possibly for QA, Face-to-face formal and informal meetings for local coordination, Synchronous scheduled group meetings via conference calls, Demo sessions, Emails for global coordination Coordination Tools and artefacts: shared user stories and tickets in Jira, WebEx for screen sharing, Skype for Business Structure: Co-located developers and tester, Global distributed QA team	- Inefficient work cycle between the development, testing and QA activities - Demo sessions and synchronous group meetings are effective for shared understanding of the features and testing requirements	- Time difference and Continuous availability of the remote person is challenging while communication with QA - Sluggish Test environment is challenging for the tester

Table 4.9: Tools & Artefacts used by Coordination Activities

Tools/Artefacts	Coordination Activities
Jira	Group Meeting (Formal & Ad-hoc)
	Daily Standup
	Sprint planning
	Sprint
	code review
	QA Demo
	F2F meetings
	Sprint Review
WebEx	Group Meeting (Formal & Ad-hoc)
	Voice Conference Call
	Screen Sharing
Skype	Video Conference Call
Confluence	Group Meeting (Formal & Ad-hoc)
Slack	Daily Standup
TFS	Code Review
	Group Meeting (Formal & Ad-hoc)
	F2F meetings
Retrospective Tool	Sprint Retrospective
User Story	Group Meeting (Formal & Ad-hoc)
	Daily Standup
	Sprint planning
	code review
	QA Demo
	F2F meetings
	Sprint Review
	Email Communication
Story Board	Group Meeting (Formal & Ad-hoc)
	Daily Standup
	Sprint planning
	Sprint
	QA Demo
	F2F meetings
	Sprint Review
Release backlog	Group Meeting (Formal & Ad-hoc)
'How to' Docs	Group Meeting (Formal & Ad-hoc)
	Email Communication
Software Code	Code Review
	Group Meeting (Formal & Ad-hoc)
	Screen Sharing
Sprint Backlog	Sprint Planning
	Daily Standup
	F2F meeting
Product Backlog	Sprint Planning
Burndown chart	Daily Standup
	Sprint
	Sprint Review

WebEx is a communication tool used for conference calls and screen sharing, Jira for sharing work items and radiating work status through electronic storyboards and Confluence is used for document sharing. This information will help the readers to identify the type of tools and artefacts that can be used for a particular coordination activity which can further guide them to choose the better suited one for their context.

Table 4.11 presents another perspective indicating the roles involved in each of the coordination activities and their availability preference are also specified to signal their importance in that activity.

For example, for managing the multiple dependencies on a single resource person, DM is the primary role involved in most of the face-to-face coordination meeting, such as Sprint planning, Daily standup, Code review meetings. Rest of the development team members are the secondary roles that are required in those meetings and the SM plays the meeting facilitator role to make those sessions time effective. For group meetings involving distributed development teams, the Scrum Master of each team plays the role of boundary spanner between teams to organise and manage the sessions. Besides this role, the development team representatives (e.g. DM for Team R) need to be involved in this group meetings to present their issues that need coordination with other teams, “opportunity for X’s team in US to list up issues he has got, then need coordination between a lot of teams. US based teams and NZ based teams” (PDM). The developers might be included in this meeting on ad-hoc basis, so their presence is considered optional. This information will help the readers understand the necessity of the roles that need to be involved in any coordination activity and their contribution to the effective management of impactful dependencies.

Table 4.10: Coordination Activity vs. Tools and Artefacts

Coordination Activities	Tools	Artefacts
Group Meeting (Formal & Ad-hoc)	WebEx	User Story
	Jira	Electronic Story Board
	Confluence	Release Backlog
		'How to' Docs
F2F Meeting (Formal & Informal)	Jira	User Story
	TFS	Electronic Story Board
		Sprint Backlog
Daily Standup	Jira	User story
	Slack	Electronic Story Board
		Sprint Backlog
		Burndown chart
Sprint Planning	Jira	User Story
		Electronic Story Board
		Sprint Backlog
		Product Backlog
2-Week Sprint	Jira	Electronic Story Board
		Burndown chart
Code Review	Jira	User story
	TFS	Software Code
QA Demo	Jira	User Story
		Electronic Story Board
Sprint Review	Jira	User Story
		Electronic Story Board
		Burndown chart
Sprint Retrospective	Online Retrospective Tool	
Email Communication	Email (Any email tool)	'How to' Docs
	User Story	
	Jira	
Voice Conference Call	WebEx	
Video Conference Call	Skype for Business	
Screen Sharing	WebEx	

Table 4.11: Coordination Activity wise Roles

Coordination Activities	Roles Involved
Group Meeting (Formal & Ad-hoc)	Boundary Spanner: SM DM, Developers (Ad-hoc)
F2F Meeting (Formal & Informal)	DM (Primary), Developers, Tester Facilitator: SM
Daily Standup	Development Team, SM
Sprint Planning	Development Team, SM, Proxy PO (Ad-hoc)
2-week Sprint	Development Team, SM
Code Review	Development Team
QA Demo	Boundary Spanner: SM, Development Team
Sprint Review	Scrum Team
Sprint Retrospective	SM, Development Team
Email Communication	Anyone involved in the Dependency
Voice Conference Call	Boundary Spanner: SM, DM, Developers (Ad-hoc)
Video Conference Call	Boundary Spanner: SM, DM, Developers (Ad-hoc)
Screen Sharing	Boundary Spanner: SM, DM, Developers, Tester

* **Scrum Team**: PO, SM, Development Team, **Development Team**: DM, Developers, Tester

4.6.3 Coordination Challenges

There is total 33 different types of challenges recognized in the way of coordination and those challenges are further classified into 6 high-level categories based on their core concerns: *Communication, Dispersion, Organisational, People, Process and Technological Issues*. The majority of the challenges (28 out of 33) are identified from the response to the question asked to the participants about the challenges they faced while coordinating their dependent activities. Rest of the challenges (5 out of 33) are identified from our analysis of the critical dependencies. The categorisation process is illustrated using one of the interview excerpts given below:

"…*with emails, different languages, its India, Poland, and France. There are several opportunities for miscommunication anyway, due to language difference, people make assumptions*" (PSM1)

Since email is one of the key communication channels used for communicating with distributed teams, there is a high possibility of miscommunication because of language differences. Similarly, miscommunication can result from poor contextual details in the email,

"*in a mail received in the morning, 'I have got these jobs that I need to run in parallel'; it is not clear what do you mean by jobs, do you mean how to get the table or something else.*" (PDM).

All these opportunities for miscommunication are caused by the communication channel (i.e. email) used in this coordination process; hence 'Communication channel' is used as a primary category. The core concern of these challenges is the issues in communication, therefore it is further categorised under 'Communication Issues'. 'Misunderstanding' and 'Continuous availability and responsiveness' concerns are the other challenges related to communication identified in this case. All the challenges with their primary and secondary categories are presented in Table 4.12 with their descriptions and example evidence from the data.

Table 4.12: Categories of Coordination Challenges

Secondary Category	Primary Category	Description	Example Evidence from Data
Communication Issues	Communication Channel	The channel used for communication is creating issues, e.g. miscommunication, work interruptions, that is impacting the communication and relationship building among distributed teams	"*telephone calls are good in a way, but you can't see the person. because you cannot see the person, you cannot have the same relationship with the person, i.e. the body language, facial expression etc. especially when there is lots of people on the call*" (PDM) "*With emails, different languages, its India, Poland, and France. There are several opportunity for miscommunication anyway, due to language difference, people make assumptions*" (PSM1) "*Email is slow and there's interruption. You have to respond couple of email while you are working with something else, which forces you to contact switch"* (PDM)
	Misunderstanding	Misunderstanding about the requirements because that the requirements are not communicated or presented in a format that can be translated to a user story	"*Another challenge I guess is sometimes we get requirements request that are not a user story*" (PSM2)
	Availability and Responsiveness	Continuous Availability and responsiveness is a persistent challenge while communicating with distributed teams	Identified from the data analysis of critical dependency on the DM (Dependency 2)
Dispersion Issues	Physical Distance	Physical distance is creating challenges in frequent communication and dependent work management	"*If the person were co-located, the guy might come over and have a chat to me about it and we get it sorted. when you are in co-located, you can have a chat while having a cup of coffee and ask some questions rather than bugging in via email or some other way*" (PDM)
	Time Difference	Time difference between the distributed teams or PO is creating difficulty in communication and resolving critical	"*Knowledge transfer between the QA and tester is hard because of the time difference*" (PTS)

		bugs or client issues	
Process Issues	Lack of understanding	Lack of understanding about 'How the system works' and 'what the product does'; hence face challenges to correctly operate and handle the system. It could happen in any part of the process, particularly in the client side.	"*Misunderstandings about 'what the product does'. for example, from the client team's perspective, they assume that it's a point of click, and you are good to go. It's a self-managing system, that don't need to do anything. But actually, the system is not like that.*" (PDM)
	Misunderstanding	Misunderstanding may arise within the development team due to lack of understanding about the product, process, Technology, tools which is creating difficulty in coordination	"*Misunderstandings in the team level. About process, performance, or might be technology. It might be about the technology, that might people need to learn. Might be about the different level of people thinks about the process, some says we will do it this way, another says we will do it this way. And it can be just about the product and the technology we are using. I thought it did these, no I thought it does this. The touchpoints, i.e. planning poker, sprint planning, code review are all good points to find out where is the misunderstandings are*" (PDM)
	Code Management	Mismanagement of shared artefacts (e.g. software code) between the teams is creating build issues	"*I mean people globally are checking stuffs in code base which was causing the build fail and we didn't know what's part causing that problem*" (PDV2)
Organisational Issues	Responsibility distribution	Mismanagement in the division of responsibility is creating challenges in dependent work coordination	"*I think the division of responsibility is a real challenge. I would prefer to be able to say we own this feature, or this team owns this feature*" (PSM2)
	Hierarchical Authority structure	The organisational and managerial structure defined by the organisation is creating coordination challenges	Identified from the data analysis of critical dependency on the DM (Dependency 2)
	Central Dependency structure	The organisational and managerial structure defined by the organisation is creating coordination challenges	Identified from the data analysis of critical dependency on the DM (Dependency 2)
	Workspace	The open office/workspace setup defined by the organisation can create stress and anxiety to the people	"*I have seen that Open plan offices just like we've got here, I get the feeling that lot of people don't really like them*"

			(PDV2)
	Work environment	The external interferes arising from workplace environments of individuals can create challenges in performing coordination activities	"*I face some challenge while working from home, it's not official rather personal. It mainly those meetings which is a bit longer and I need to focus, thinking needed or a bit technical compared to daily standup. It's not a problem for me but its disturbing others of the conference due to the noise*." (PDV4)
People	Lack of engagement/ participation	Lack of engagement due to interpersonal issues such as language, culture, trust, personal skills, personal attitude and preferences	"*I think a lot of people don't listen and even in short and face-to-face conversation*" (PSM1) "*Getting them to talk and getting them to listen is another challenge*" (PSM1) "*It's really challenging if the teams don't understand how they affect the other team or how the other team affects their work, it's harder to engage them*" (PSM1)
	Territorial Mindset	Lack of engagement due to interpersonal issues such as language, culture, trust, personal skills, personal attitude and preferences	Identified from the data analysis of work-output dependency between development teams (Dependency 1)
	Lack of Trust	Lack of engagement due to interpersonal issues such as language, culture, trust, personal skills, personal attitude and preferences	Identified from the data analysis of work-output dependency between development teams (Dependency 1)
Technical	Communication Tools	Tools used for communication has persistent issues that is creating challenges in coordination activities	"*There's problem with tools. All of our tools we've problematic and so even if you could get a phone call, you can't hear them properly or you end of the call after 20 mins*" (PSM1) "*Yes, we have challenges from WebEx*" (PSM2)
	Process Tool	Difficulty is managing the tools used for work or process management	"*The manual management of whiteboard is waste of time*" (PSM1)

There are several factors that contribute to these coordination challenges. It is important to identify and consider those factors since they have both positive and negative effects on the selection and effectiveness of the mechanisms. All the distinguishing factors are grouped into three categories as shown in Table 4.13, those are *Contextual* (Temporal, Geographical), *Technological* and *Organisational* (Management, Team). Each factor contributes to several types of implications for the selection and application of coordination mechanisms in this context. For example, multiple participants shared their experience with technological issues while communicating and coordinating with external teams, particularly at the distributed segment of the process.

"*There is a problem with tools. All of our tools we've problematic and so even if you could get a phone call, you can't hear them properly or you end of the call after 20 mins…*" [PSM1]

Similarly, time difference, physical distance, language, and trust are the factors associated with temporal and geographical contexts. Finally, the factors in the organisational high-level category relate to concerns associated with people, process, roles, and authority. The key purpose of extracting these factors is to help the selection and adoption of coordination mechanisms in alleviating the coordination challenges in DASD and similar contexts.

Table 4.13: Categorisation of Factors associated with Challenges

Category	Sub-category	Category details
Contextual Factors	Temporal	Time difference
	Geographical	Physical distance, Language, Trust
Technological Factors	Technology	Communication Channel, tools
Organisational Factors	Management	Process, Roles and Authority
	Team	People, Expertise, Mindset

4.7 Summary

This chapter has presented the analysis and findings of the first case studied in this research and identified key dependencies that are associated with delay and poor-quality impacts. A repertoire of activities supported by tools and artefacts, called as coordination mechanisms are required to manage the dependencies. It is noticeable that the same mechanism could address multiple dependency requirements, but their effectiveness may vary based on the coordination requirements. Six categories of coordination challenges are recognised impeding the coordination of the dependencies, among which people and communication issues are the predominant ones. Finally, three types of factors are identified that have an influence on the selection and application of the coordination mechanisms in this context. The next chapter (Chapter 5) would present the findings of the similar analysis performed on the data collected from the second case, followed by the comparative analysis of both case findings presented in Chapter 6.

Chapter 5 : Case 2 Findings

This chapter presents the key findings from analysing a second local organisation as a case study in Distributed Agile Software Development (DASD) and its coordination. The work studied in this organisation is the replacement of an existing organisation-wide Commercial-Off-The-Shelf (COTS) software package with a new enterprise COTS software system. The product vendor and their team of developers and domain experts were based overseas in multiple time zones, and they worked closely with the local development team and users to implement and customise the COTS system. The local team used an agile way of working to deliver customised functionality incrementally. Several significant and high-impact dependencies were identified, some successfully managed, and some were more challenging. Similar to the previous organisational case study analysis, the project and process are discussed first in sections 5.1 & 5.2, followed by the presentation of activity analysis and findings in section 5.3. The coordination analysis process in which the high-impact dependencies are being identified is described, and the results are presented in section 5.4. In section 5.5, these findings are further interpreted to develop an overall understanding of coordination in a COTS-based DASD project context. Finally, this chapter concludes with a summary presented in section 5.6.

5.1 Project Context

The case organisation is a medium-sized organisation in New Zealand that relies on information systems for the provision of utility services to customers, as well as managing assets and maintenance. All the products of the organisation support three core value streams: Customer, Operation and Planning. Each product represents one of the three product streams that use an existing software system. For better business operations and customer experience, the organisation approached a new COTS ERP system and, at the time of the data collection, were 12 months into the planned two-year installation and customisation process. The project targeted the successful

transition of the current system to the new cloud-based Software-as-a-Service (SaaS) product suite, and this project is the focus of this study. We have coded this project as "**Bluebird**" and will use this name in the rest of this chapter.

The organisation is a council-controlled organisation providing one of the largest utility services to the Auckland region. The overall project activities were circled around the four customer journey maps: resource, business, network planning and asset. The journey maps concentrated on the successful management of assets and resources, their distribution and provisioning of services for a rich customer experience. The organisation adopted an agile way of working for almost a year prior to the commencement of this study, which took place in August 2018. A summary of the project context is presented in Table 5.1.

Table 5.1: Overview of the Bluebird Project context

Case Name	**Bluebird**	
Organisation	Organisation Type	Commercial
	Organisation Size	Medium
	Business activity	Utility Service Provider
	Market	Auckland Region (Council-Controlled)
Project	Project Status	On-going
	Project Purpose	COTS ERP system customisation
	Project Criticality	Medium
	Project Stakeholders	Global vendor and distributed teams and experts, NZ-Based POs and Project teams with distributed team members Local Subject Matter Experts
Product	Product Size	Medium
	Product Status	Immature
Team	Number of Teams	7 Squads
	Team size	10 members of O&M squad
	Interviews	7 members (O&M squad) 1 Project Manager (Vendor-side)
Development Method	Agile Approach	Scrum

There are seven teams (termed as Squad) involved in this customisation project and each of these squads focuses on delivering the functionality of different value streams in the new system. All the squads had a PO who frequently collaborate with other teams based on the release goals. A Chief PO leads the group of POs who are responsible for setting the project goals and priorities based on the three core value streams.

One of these seven squads is the Operation and Maintenance (O&M) squad which was the subject of this study. This squad was responsible for two core products: IPS (for customer billing and inventory) and EAM (Asset management and call centre module). The O&M squad comprised ten core members who were responsible for the configuration and customization of the O&M module in the ERP system. As illustrated in Figure 5.1, eight members were co-located in the NZ site and another two members (one FD and one TD) intermittently worked remotely from USA and Singapore, respectively for various reasons. These two global members were resourced from the US-based vendor who had specialised knowledge of the new COTS system and worked full-time as part of the O&M squad. The TD was responsible for scripting and configuring new product customizations (including unit tests), whereas the FD focused on the user interfaces, business processes and low-level software integration. Coordination of the team meetings and dependent activities was challenging at times when these two resources worked from offshore, as discussed in the Findings section. All the interview participants of this case were from the O&M squad, except the Product Manager, who was common for all the squads. The list of participants from the Bluebird case indicating their roles and codes is presented in Table 5.2.

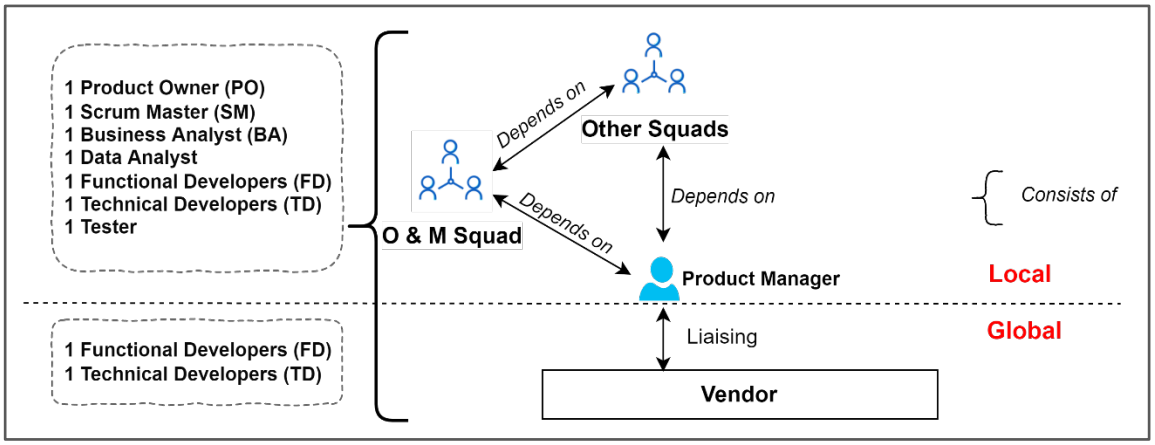

Figure 5.1: The O&M squad and its coordination context

The SM of the O&M squad, who is also accountable for three other squads, led the agile implementation of the squads by facilitating most of the meetings and agile ceremonies. The BA coordinates with the Team PO to write user stories and groups them by user personas that are relevant to the O&M squad. He also coordinates some inter-team dependencies related to the current sprint and manages the 'Risks and Issue Board' used to coordinate the inter-team dependencies relevant to the current release. Besides, there are several Subject Matter Experts (SMEs) with the specific domain expertise of the current system, who are external to the team, involved in most of the development activities and closely work with the squad to interpret the existing features of the new system. An NZ-based Product Manager (PM) is responsible for most of the communications with the vendor and their technical teams. Figure 5.2 indicates the key stakeholders of the project and illustrates their relationships. It is noteworthy that the time zone differences between the New Zealand-based and distributed entities were: (1) six hours for the vendor resources, (2) ten hours for remote FD in Hawaii (3) four hrs for the remote TD in Singapore.

Table 5.2: Bluebird case participants, their roles and codes

Participant No	Participants Role	Code	Location
1	Product Owner	BPO	NZ
2	Scrum Master	BSM	NZ
3	Technical Developer 1	BTD1	NZ, Singapore
4	Technical Developer 2	BTD2	NZ
5	Functional Developer 1	BFD1	NZ, Hawaii
6	Functional Developer 2	BFD2	NZ
7	Business Analyst	BBA	NZ
8	Tester	BTS	NZ
9	Project Manager	BPM	NZ

There are three main areas of dependencies between the vendor and the O&M squad: *Support*, which handles first-level enquiries, issues and bugs; *CloudOps*, which is responsible for the deployment of new features and data to the cloud; and *Development*, which fixes new bugs identified in the COTS product as well as maintain and enhance the existing COTS product. Coordination with this distributed vendor team has been crucial for the success of the project, as discussed in the results section.

5.2 Overview of the COTS-Customisation process

The O&M squad adhere to most of the core Scrum practices and values, with two-week sprints. An overview of the entire software process for the O&M squad is depicted in Figure 5.3 and a detailed view of this workflow is presented in Figure 5.4. The seven squads coordinate their work to deliver a release every twelve weeks (six sprints). It is planned that every three sprints newly finished features are released to a group of Beta testers who provide feedback for refinement of the features, data and User Interface (UI) for future sprints in the release. At the end of the release cycle, all the functionalities within the release are deployed and made available to the relevant user groups. Overall, the work and information dependencies between squads were a challenging coordination issue and significant effort was spent on managing this, including regular maintenance of a squad dependency board as an information radiator and Scrum-of-scrum meetings (see the Findings section). Our study, however, focuses

on the interactions and effort to coordinate dependencies between local and distributed entities related to the O&M squad's work within the release cycle.

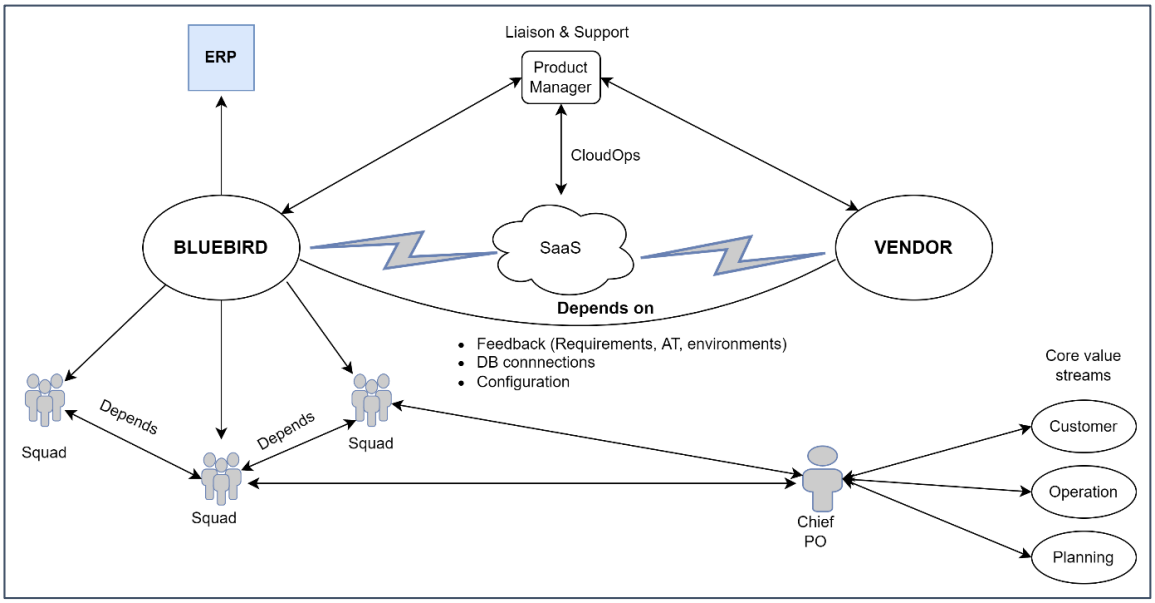

Figure 5.2: Project structure including the key stakeholders

Requirements Discovery is continuously undertaken but is more intensive for the two weeks prior to the start of a new release (i.e. *Release Planning*). Business requirements are identified and clarified from other parts of the business, including SMEs, end-users, vendor representatives and the original Request for Proposal (RFP) documents. Squad POs, led by the chief PO, coordinate and cooperate to manage these business requirements and order them into release cycles. The O&M PO then transforms the business requirements for the next release into brief user story Epics and groups them into the personnas identified by the squad. Each persona represents a user of the products which may be affected by that customisation. The BA then converts the Epics into finer-grained user stories and explicitly identifies inter-squad dependencies for each story. These user stories are recorded in the Product Backlog for this release and further prioritized and organised into six sprints by the squad during *Sprint planning*.

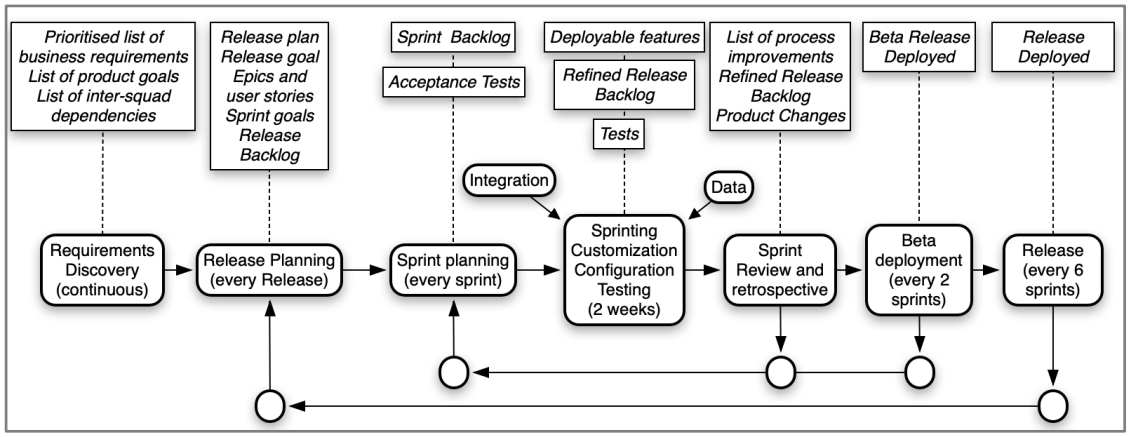

Figure 5.3: Overview of the COTS Customisation process

During the *Sprinting* phase, as illustrated in Fig 5.4, the main activities were coding customizations, testing features, and creating deployment plans. The O&M squad conducted daily *Scrum stand-ups* where both local and distributed team members participated, including PO and SMEs who were involved in that sprint. Regular consultations were sought from the architecture squad for design and solution advice. The Vendor provided the coding standard that was being followed for all the customisation and development works. Backlog refinement sessions were conducted with the PO and BA to clarify the requirements and prioritisation. A lot of sprint information was shared through artefacts on the walls around the workplace (e.g. descriptions of personas, release and sprint goals, definitions of “done” and “ready”, and project risks). Jira was used as an electronic storyboard for workflow transparency. Separate physical squad boards were maintained for intra- and inter-team work dependencies, which were placed in the squad’s workspace and in the corridor, respectively within clear visibility. Code related to customizations was shared between squad members for development in an online code repository.

There was separate development, testing, user acceptance testing (UAT) and production environments, and customizations were moved from one to the next for final deployment to the production. While the deployment of features was the responsibility of other technical squads, the O&M squad provides the planning and writing instructions for deployments related to their products.

At the end of the sprint, a *Sprint Review* was conducted often with some stakeholders attending electronically. Generally, a *Sprint Retrospective* was held on the same day as the sprint review and was facilitated by the squad PO. A *Beta release* was deployed in every two sprints and goes through testing and feedback from selective user representatives before it was ready for final deployment. The final *Release deployment* accumulated three latest beta-release features resulting from six consecutive sprint outputs.

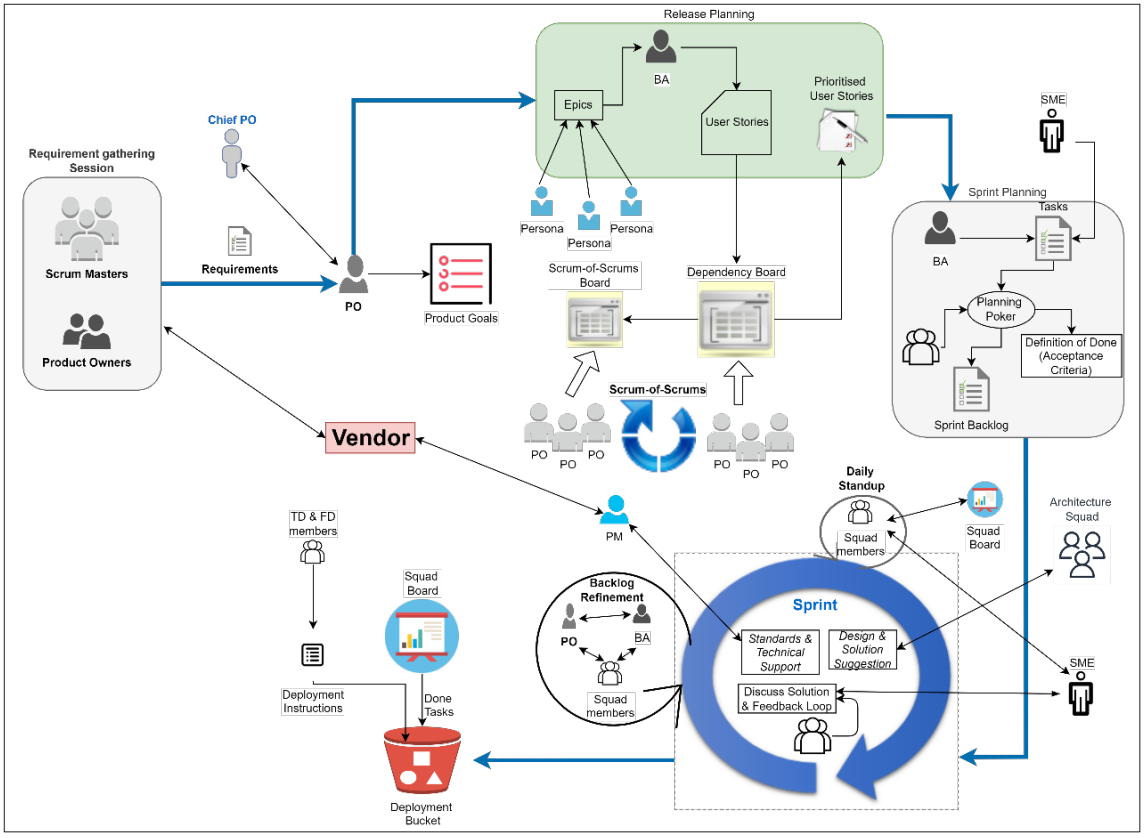

Figure 5.4: Detailed view of the COTS Customisation Process

5.3 Activity Analysis & Findings

The activity analysis outcome presented in this section follows the same process as the previous case described in Chapter 4, section 4.3. In Bluebird, the activity elements were mapped to the release cycle within the COTS customisation process that starts with a list of release goals that are gradually converted into a working product. All the elements presented in this mapping are encountered while analysing the COTS Customisation process discussed in section 5.2. These elements are then mapped into

a complete abstract visualisation shown in a diagram presented in Figure 5.6 that provides an overview of the entire COTS customisation activity. In the following sections, the elements of this activity are discussed to represent the actors, their organisation, interactions and overall work process to achieve the release goals.

5.3.1 Object and Outcome

An example release-cycle of this project that is mapped using an Activity-based framework takes business requirements as a shared object for this activity that is being organised into release cycles. Collaboration among a group of people involved in this activity is required following the predefined process to convert this shared object (business requirements) into an outcome i.e. a new feature deployed in the new COTS system.

5.3.2 Mode of operations

Extensive collaboration among different entities is required to transform the shared object into a deliverable product which is considered a mode of operation in this project. Since various tools are used to support this collaboration between local and distributed entities, the mode of operation can be considered as a technology-driven collaboration mode.

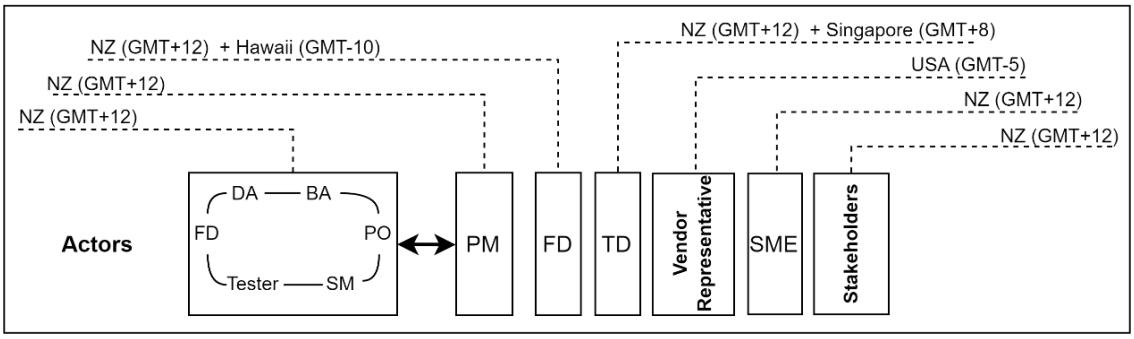

Figure 5.5: Actors and their locations for Bluebird project

5.3.3 Actors and sites

The customisation of the new COTS product was carried out in collaboration with the Vendor, who are geographically distributed and there was a seventeen-hour time

difference between the client and vendor's development site. All the actors and their locations with their time zones are indicated in Figure 5.5.

As shown in Figure, the O&M squad, is a NZ based team, works in close collaboration with two vendor-side experts (i.e. one FD, one TD). As discussed earlier, these two experts intermittently work from Hawaii and Singapore and NZ sites for multiple reasons. There is a 23-hour time and a five-hour time difference with Hawaii and Singapore respectively with the NZ site. The PO of the squad directly communicates with the PM, who is based in NZ, for any vendor-specific queries and support.

There are three vendor-side teams involved in this customisation process, those are- Product Development team, Support team and CloudOps team. The PM coordinates with these teams through the designated representative for all kinds of support and issue solving with these three vendor teams.

There are several SMEs in the client side who have strong domain knowledge and expertise about the existing SAP system. They are external to the squads and located in the client site who are consulted for queries and understanding the functionalities of the existing system to implement them in the new system. They are involved to breakdown the user stories into tasks, guiding to achieve any solutions based on the current system knowledge.

Additionally, there are some internal and external stakeholders of the product who needs to be involved in different stages of the customisation process. For example, the DevOps team is an internal team located in the client site that is closely aligned with the operations of the product and they perform the UAT, Sandbox Deployment of the customisations. O&M squad needs to coordinate with them for testing and deployment. All the existing customers are considered as external stakeholders who will be affecting their works; therefore, their perspectives need to be considered in the COTS customisation. All these stakeholders are located in NZ and are communicated by the PO as per the need. Since the customisation process was still underway at the time of this study, these stakeholders had not been involved. However, the organisation is well

aware that these stakeholders will face the major impacts from this customisation and had planned to include them from the early stage of the product rollout.

5.3.4 Means of communication and coordination

Several modes of communication and coordination are being used in this project to coordinate between local and distributed sites. There are two segments of communication and coordination exists in this case. One segment focuses on the coordination between the client and vendor site entities, while another focuses on the coordination between the entities present in the client sites. Both *Synchronous* and *Asynchronous* mode of communication are used in cross-site communications. Email and Jira are examples of asynchronous communication mode; whereas phone or video conference using screen sharing are most common modes used in synchronous communication.

For local site coordination, squads conduct face-to-face meetings to manage inter-squad work dependencies. Email is also frequently used to communicate between the squads. Intra-squad communications are done either face-to-face or electronically via chat tools and video conference. For example, whenever the vendor-side FD or TD is working offsite, they frequently communicate with each other using chats and for daily standup and other group sessions, the offsite member is connected in phone call using a dedicated Smartphone device connected to a gimble stabiliser.

Different types of shared artefacts aid the communication in different stages of the project. For example, Jira is used to log jobs or defects that requires support from vendor-side teams. Epics, User Stories, Product Backlog, Sprint Backlog, Deployment bucket are several other forms of artefacts that are being used to support communication and coordination throughout the project lifecycle. Dependency board, Definition of Done (DoD), Definition of Ready (DoR) are examples of physical artefacts that facilitate information sharing and communication. For example, Physical Dependency Board is being used to record inter-squad dependencies and all the inter-

squad coordination meetings are conducted in front of this board to visualise the dependencies and their status.

5.3.5 Means of work or mediating instruments

A repertoire of mediating instruments has been used to manage and perform work activities in collaboration with collocated and distributed entities involved in the development process. Since all the squads are following the agile way of working, most of these instruments adhere to the practices prescribed by the agile framework. For instance, to track the requirements in each phase of the development process (from the business requirement to deployment), a popular collaborative software development tool (i.e. Jira) is used. Different kinds of task management, issue tracking, and progress tracking options are used that supports collaboration between multiple groups and individuals from both the client and vendor-side. For example, vendor-side CloudOps team raised incidents in Jira when a bug is identified. Electronic *Squad board*, *Risks and Issue board* items are also maintained using this tool. A collaboration wiki platform (i.e. Confluence) is used for notes and tags management which is integrated with Slack (IM tool) for real time notification of changes in the documents.

Each squad maintains an Electronic Squad board for task management during sprints which has *To-Do*, *In Progress*, *Blocked*, *Reviewing* and *Testing* column. Besides, for local coordination, physical white board and paper board are respectively used for intra- and inter-team work status and progress tracking. A cloud-based collaboration tool (i.e. Bitbucket) is used for both code versioning and DevOps management. For automated testing, Selenium is used by the client side teams.

Most of the scrum prescribed practices have been used for managing iterative software development life cycle. The business requirements are translated into user stories that is being used in the sprint planning session for estimation and self-task assignment. A daily time-boxed standup meeting is conducted throughout the two-week sprint duration. Backlog refinement sessions and peer review are conducted intermittently during the sprinting phase. To foster informal face-to-face communication, all the

squads are organised side by side in an open plan workspace. Highly inter-dependent squads are placed close to each other to share the same workspace for improved collaboration.

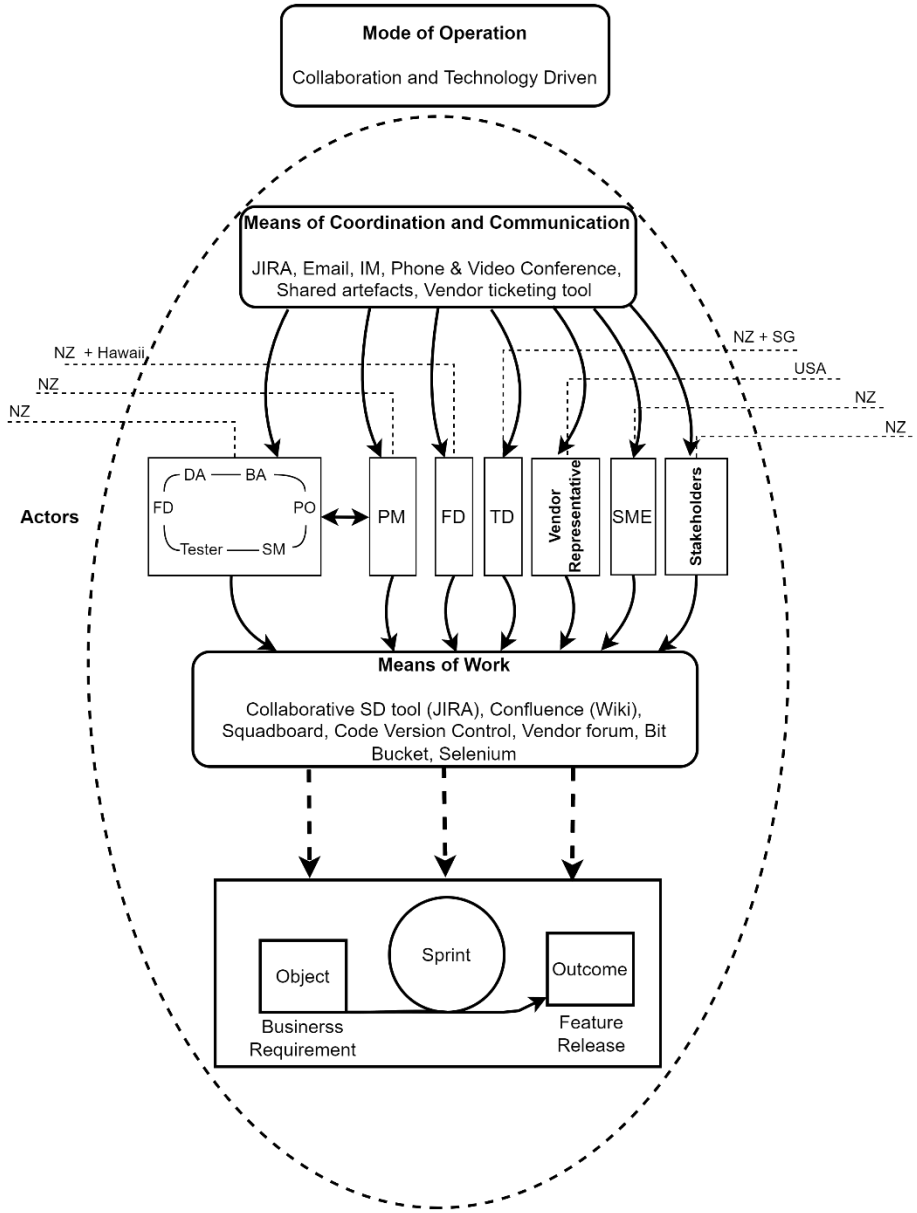

Figure 5.6: Elements of COTS Customisation activity in the Bluebird case

Formal 'Definition of Done' and 'Definition of Ready' declarations and module-wise 'Goal of Project' are posted on the walls of the squads to facilitate shared understanding about the quality requirements for shippable product releases. Scrum-of-Scrums meeting is conducted once a week to coordinate inter-squad dependencies and identify risks and issues in the ongoing release. This meeting is conducted in front

of the physical dependency board to visualise the dependencies between the squads, locate potential risky dependencies and issues that need to be escalated to the PM and synchronise any change in the priorities. All the identified risks and issues are then copied in the 'Risk and Issues board'. Predefined beta-release (i.e. after every two sprints) and production release cycle (i.e. after every six sprints) are followed by all the squads. Each production release needs to pass through the QA and UAT phases.

All the elements discussed above are incorporated in Figure 5.6 to portray their relationships in this DSD context. As suggested by Korpela et al. (Korpela et al., 2002), these elements need to fit together to form a 'Coordination' strategy that facilitates the successful transformation of the business requirements (Object) into production release (Outcome).

5.4 Coordination Analysis and Findings

The analysis process of this case is performed using the framework of analysis developed for this study and we have followed the steps similar to the previous case (as discussed in section 4.4). The process started with the identification of dependencies in the prescribed COTS customisation process followed by O&M squad and the dependencies are coded (i.e. primary coding) based on our understanding of 'who is dependent on whom', 'for what' and 'why'. For example, The Technical Developer (TD) depends on the Functional Developer (FD) for his knowledge of the functional features of the vendor provided COTS product to solve client-side issues and bugs. The primary level codes were updated based on the review and discussion with the supervisors. All dependencies were then classified and grouped into secondary level codes based on their dependency purpose. For example, all dependencies related to Vendor were grouped together and coded as 'Dependency 1' and named as 'Dependency on Vendor'. Coding and classification of the data were done independently by the researcher and one of the supervisors, and the final codes and classifications were agreed on through discussion and negotiation.

Table 5.3: Summary of the Critical Dependencies and their impacts

	Dependency Type + Local/Global	Dependency Description	Impact	Explanation/Justification
1	Dependency on Vendor (global)	Client-side squads extensively relies on Vendor for their expertise, support, and authority over the COTS product for the customisations and integration of the features	Client-side squads' work often depended on the shared understanding or delivery of the work at the vendor-side which has high potential to block the work progress and result in delays in the squads' work	Only the vendor had the expertise and authority (dependency) to complete certain tasks such as bug-fixing and product enhancement, as well as cloud-related tasks such as module configuration, cloud infrastructure setup, and database connection. The onsite PM acts as the liaison person (mechanism) to support boundary spanning activities with the vendor-side teams.
2	Dependency between Squads (local)	Client-side squads need to synchronise themselves in multiple phases of the SD process for their expertise, work outputs, decision making and risk management	Since squads' work progress is dependent on other squads', it could potentially delay the work or wait until they can get hold of each other to discuss and manage the information or work needed	Each squad is specialised in part of the COTS system customisation on which other squad is dependent on. For example, O&M squad depends on DH (Data Hub) squad for data collection from the existing system, processing and loading to the new system (dependency) that is necessary for them to do the customisation work. "…data comes from the data hub squad, they refine it which requires Business knowledge… very few times to get hold of them" (BTS1)
3	Dependency between co-located members and remote squad members (local + global)	O&M squad members depend on both the local and remote members for their expertise, information exchange, work output, decision making to manage work deliverable co-dependencies	The coordinated participation of each member has been crucial to the timely information exchange and work completion which could potentially impact the work deliverables and team trust	All the development works done by the squad heavily relies on the expert product knowledge (dependency) of the vendor-side TD and FD. By participating in regular scrum process activities and ad hoc personal communication (mechanism), squad members coordinate their information and work dependencies with them.

4	Dependency on squad's External entities	The identification of the client-side customisation requirements and their successful implementation heavily rely on the external entities (e.g. SME, Pilot users) for their business domain knowledge	The availability and participation of the external entities have been key to the shared understanding of the existing system and the customisation requirements that has impacts on the quality of the work outputs	The SMEs and end users need to be consulted for identification of business requirements and their implementations (dependency). The SMEs participate in the planning sessions and daily standup sessions (mechanism) to respond to the information and feedback needs of the squad.
5	Technical dependencies	The understanding and adherence to the technical constraints was crucial to perform the customisation and integration activities	The shared understanding of the technical relationships between the products and satisfying those constraints is must while working on the customisation; otherwise it may create risks of delay and quality issues	IPS and EAM system has strong dependency on each other which mandates proper data integration (dependency) between them. Developers load the data in IPS system (mechanism), which is then integrated through a specialised integration tool.

Each of the dependencies is further gone through impact analysis to diagnose impacts and their root causes, followed by coordination activity analysis to explore coordination activities, tools and artefacts, and their performance to mitigate the impacts. Finally, all the coordination challenges associated with the dependencies are pointed out with the cofounding factors. The results of the analysis are presented in the next sub-sections, followed by a discussion section to interpret the key findings from this case study.

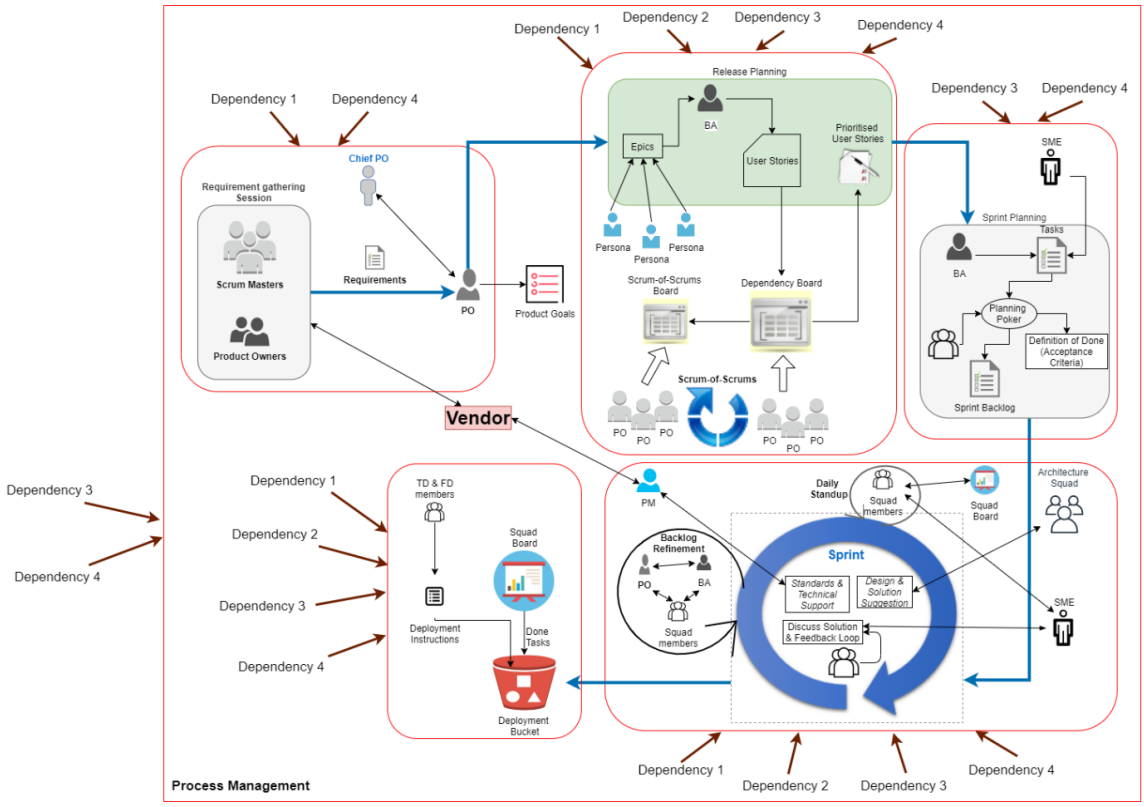

Figure 5.7: Dependencies identified in the existing COTS customisation process

The findings are organised into five key dependencies that were identified in the existing COTS customisation process, 1) dependency on vendor (*Dependency 1*), 2) inter-squad Dependency (*Dependency 2*), 3) dependency between co-located members and remote squad members (*Dependency 3*), 4) technical dependencies (*Dependency 4*). These dependencies are located in different parts of the current SD process as indicated in Figure 5.7. It was interesting to discover that the most critical and problematic dependencies are those ones with the distributed parties involved that is discussed in the findings section.

A summary of these dependency categories is presented in Table 5.3 with their definition, possible impacts, and justification of their categorisation under each theme. Each of the high-level categories are developed through the coding process of all the dependencies identified from the case data analysis. The categorisation process for each of the critical dependencies is discussed while presenting the findings of the respective dependency. However, a detailed list of all the dependencies with their primary and secondary coding is included in Appendix C for reference.

5.4.1 Dependency 1: Dependency on Vendor

One of the key dependencies that were poorly coordinated was the dependency between the development teams and the vendor. The co-located development teams relied on the vendor for requirements, their product expertise, work outputs, and authority over the infrastructure and strategic decisions. This coordination was critical because of the time delays and thin communication channels which made it difficult to get Adhoc support from the vendor. A summary of the Squad-vendor coordination needs, and barriers is shown in Figure 5.8. The following sections first introduce the dependencies on the vendor in the COTS-product customisation process, then discuss their issues and impacts, and finally presents the analysis of current coordination mechanisms and their challenges.

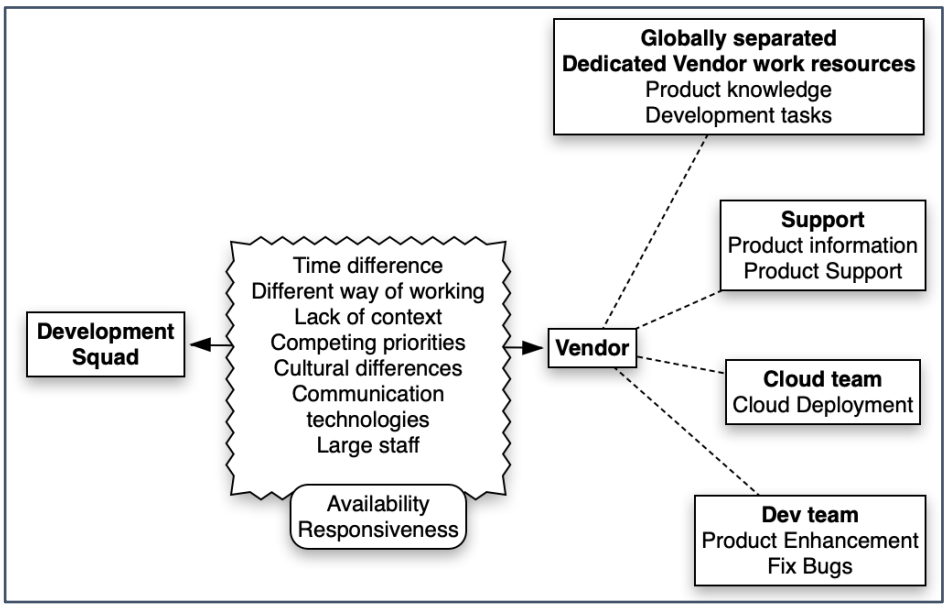

Figure 5.8: Summary of Squad-Vendor Dependencies and challenges

5.4.1.1 Dependency Description

The vendor dependencies were related to different forms of knowledge (e.g. requirements, domain and project specific information, expertise), work-output and decision-making dependencies. Some of these dependencies were anticipated prior to release or sprint planning (e.g. often technical and product knowledge were shared during sprint planning sessions, as user stories and associated tasks were clarified). Other types of dependencies became evident during the sprint execution as understanding and learning deepened (e.g. development of acceptance tests uncovered shortcomings in the understanding of the user story, or a technical problem triggered the need for a deeper technical understanding of the product from a scarce knowledge resource). The dependency categorisation process is presented in Table 5.4 with illustrative evidence from the data.

The requirement dependency on the vendor occurred when client squads needed to know the system requirements and scoping the customisation as part of the requirement discovery session. This includes the configuration, customisation and data migration requirements based on the 'as-is' process analysis of the current system that will be transferred into the new COTS product. These product-specific requirements were required for identifying the release goals and further breaking down the user stories during release planning. The expertise and domain knowledge dependency occurred whenever the development teams needed support during the development and implementation of the customised product features. The vendor had specialised teams for product development, support and Cloud-based operations, who can understand the client requests and support accordingly. At the same time, several types of information and feedback were exchanged between these two distributed parties to gain a shared understanding of the product, its functionalities, issues, and their solutions to fulfil the acceptance criteria.

Table 5.4: Dependencies on the Vendor in Bluebird case

SL#	Primary Coding	Description	Part of the Process	Evidence from the data
1	Requirement	Client teams rely on vendor to identify SRS and configuration requirements, scoping the customisations activities (configuration, customisation, and data migration requirements)	Requirements Discovery	The knowledge of the requirements comes from understanding of the current system and from the vendor, which is further considered as Release goals. [BBA]
2	Expertise	Different vendor team members' expertise is required to understand and support the client-side requests	Sprinting, Deployment	The vendor has multi-talented people working who support the requests from client sides and provide support accordingly. [BSM1]
3	Domain Knowledge	Vendor-side teams have product specific knowledge to solve client-side issues and bugs	Sprinting	There is specific product development team, support team and a CloudOps team for each different elements of the product. [BPO]
4	Information	Information exchange about the incidents raised by the CloudOps team (in Jira)	Sprinting	Technical Developers need to communicate CloudOps teams for the incidents raised by CloudOps teams in Jira [BTD1]
5	Feedback	Vendor-side feedback is required on the Acceptance test/criteria based on User stories	Release planning and Sprinting	BA receives the feedback from the vendor about the Acceptance criteria & Tests case based on the User stories and adjusts them accordingly [BBA]
6	Resource	Squads relies on the Vendor to provide coding standards that the squads need to follow to avoid quality issues	Sprinting	Vendor provides the coding standard for writing codes, and developers follow that guideline to meet the standard. For this reason, they do not perform any formal code review session [BTD2]
7	Work output	Product side configurations are done by the vendor that are required for the customisation and deployment of the features	Sprinting	CloudOps team support is required for any feature implementation or infrastructure support due to security concern [BTD1] if a functionality is not working, we are going back to the Devs, they will investigate it and then got that this is product problem, raise an incident ticket [BPO]
		Vendor-side development teams are responsible for product specific bug or defect solving which has impact on the client-side customisation work	Sprinting	
		Vendor's CloudOps Team is responsible for cloud-related tasks and supports	Integration-Deployment	

		Integration Squad depends on the Vendor-side teams to create DB connections and resolve connection issues	Sprinting	
		Client heavily relies on vendor for all product specific support and request handling which is required for their development activities	Sprinting	
		Client-side squads relies on the vendor to identify and report any incidents found in the product caused by the customisation	Sprinting, Integration - Deployment	
8	Authority / Decision making	Client and vendor collaboratively decide which part of the customisation are performed by whom, i.e. scoping	Requirement discovery, Sprinting	There is dependency between Product team (vendor) and Development team (client) for the infrastructure and customisations (decisions) [BPM]
		Client depends on the vendor for strategic decisions for product specific customisation (e.g. which customisations are viable to align with the 'as-is' business process)	Requirement discovery, Release and delivery planning	
		Vendor is solely responsible to take decisions about the infrastructure and their support	Sprinting, Integration - Deployment	
9	Liaison Person	Client-side squads depend on PM (Onsite) to support Boundary-spanning activities with the vendor, e.g. liaising with vendor for technical supports, escalate issues in the vendor-side, prioritise Adhoc work requests	Project management (in any phase of the development)	"*We have a very thin channel through how the project manager who looks after resourcing at that end of it... so he's a programme team resource that sits across the whole lot*" [BPO]

There were several types of work-output dependency that occurred during the development and implementation phase. For example, product development teams were responsible to perform the product side customisations that was required to integrate the existing feature in the COTS product. When both the product teams and development teams completed their side customisations, CloudOps team performed all the integration and implementation activities. In addition, all kinds of technical supports such as requests for creating database connections and fixing connection issues, infrastructure and product supports activities were also performed by specialised support team. All these kinds of work-outputs were important for on-schedule completion of the release goals.

Besides, there were several occasions when important decisions were made in collaboration with the vendor and development squads. For instance, whenever there was a misalignment with the existing features that would require customisation, both parties collaboratively took part to decide what were the customisations needed, scoping out who would perform which part of the customisations, how the customisation would be developed and implemented in the new product and what kind of supports required from the vendor. These kinds of decisions were crucial for moving forward and meet the release goals to succeed in this COTS-based product customisation project.

The complex nature of the above dependencies on the vendor leaded to the high dependency on the Project Manager (PM) who acted as a liaison person between the client and vendor teams. The PM is a member of the programme management team who "*liaises with the product teams for each platform*" [BPO] and mediated most of the communications between the parties for all kinds of work and expertise coordination. At the same time, he had the authority to make decisions on behalf of the clients and escalate the work priorities and support issues in the vendor-side for prompt support. Though the PM was a Programme team resource, because of his involvement and importance in vendor-client coordination, this dependency is discussed here rather

than in the inter-squad dependency section (i.e. Dependency 2 discussed in section 5.4.2).

Overall, due to the nature of the project, all the dependencies on the vendor are inevitable and poor coordination of them could potentially lead to a number of dependency issues discussed in the next section.

5.4.1.2 Issues and Impacts

All the dependencies discussed above were further analysed to understand the circumstances that triggered them and how they could impact the overall project delivery. Vendor dependencies mainly existed due to the shortcomings in the base COTS system that did not completely meet the client requirements. There are three potential outcomes that intrigued opting for the new COTS ERP system:

(1) the vendor agrees to extend the functionality of the product for a more general market,
(2) the client decides to customise the COTS system to suit their requirement,
(3) the client accepts that the desired requirement is not worth the effort.

The second situation best described the needs of the Bluebird case, where client-specific customisations were made to meet specific desired requirements, at the client's cost. The need to know how to implement this customisation, often relied on the knowledge and expertise of the vendor about their product and its customisation capabilities. To add to the coordination complexity, the squad-vendor dependencies required coordination interactions with several vendor teams including the product development team, the support team, and the Cloud Operations team.

In the current situation, because of the two-way communication involved in the coordination of dependencies, there were several opportunities for delays such as:

- delays in recognising a dependency,
- delays in communicating a coordination need,
- delay in getting decisions and support
- poorly communicated coordination needs

All these delays potentially contributed to further delays in work coordination and project deliverables schedules. An example of the communication complexity that may be involved in dependency coordination involving information exchange (knowledge sharing) between the vendor and a development team member is illustrated in Figure 5.9. This example illustrates a large number of opportunities for delay between information seeking, sharing, clarification interaction and receiving an actionable information. This sequence of information exchange could potentially face long delays (a day or two) because of the unavailability of the vendor (due to the time zone difference).

In addition, since the squad members did not personally know members of the vendor team, there was uncertainty on who is the right person to contact for specific work-related information. This lack of knowledge about source of expertise or contact person leads to potential delay in getting required information, and hence, delays in work completion.

In terms of work-output dependencies, there were two aspects of creating delay in getting the work needed by the client's development squads. Firstly, misalignment between the Agile process of the squads and the more linear process of the vendor resources had an impact on the social and technical congruence in software teams. This was compounded by another fact that the vendor resources had other clients' work to do and so their priorities did not always coordinate with the squad's needs, as a result, requests were getting queued. The PO says, "*we've struggled at times with time frames around certain defects get raised and also this tickets get raised certain*

elements of those teams…". As a result, the squad's work progress had to wait until the vendor's development work was completed. Since the client squads had cascading work dependencies (i.e. one squads' work-output required by another squad to complete their work), there was high risk of cascading delays that had effect on the sprint progress.

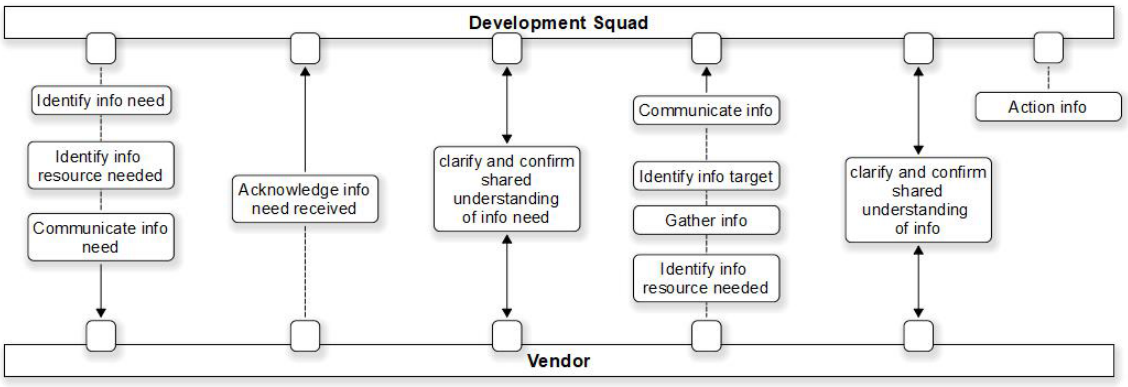

Figure 5.9: Example of Delays in Coordination with Vendor

5.4.1.3 Analysis of Coordination mechanisms

The main coordination mechanism for this dependency was the channel through a dedicated resource person onsite designated as *Project Manager* (PM). The PM was a part of the client's Program Team resource who liaised with each product teams on the vendor-side. He primarily represented and negotiated the vendor-side product customisation requirements during the requirement discovery sessions. He attended the '*Scrum-of-Scrums*' (SoS) sessions to acknowledge work and support requests for all the development squads and coordinated them with respective vendor teams.

To avail the expertise of the vendor teams, squads used a ticketing system provided by the vendor to log all sorts of work and support requests (a.k.a. Incidents). All the recorded tickets were linked to respective Jira tickets (client's issue tracking tool). The vendor-side FD and TD, who had access to the vendor ticketing system, were responsible for creating the tickets and other developers could communicate by commenting on the tickets to understand and resolve the issue. At the same time, the squad members could send emails to the respective person handling the tickets for additional information and support. The vendor also maintained an online forum which

was a good source of information and resource for product specific queries. Since the communication and information exchange via this forum was asynchronous, this mechanism was not effective in case of urgent and complex issues due to possible delays in getting response which might create interruptions in the customer side.

The Scrum Masters (SM) of the squads coordinated with the PM to escalate issues during the SoS meeting. The PM further conducted meetings with vendor teams to coordinate whatever tickets are open and resolve any pertaining issues. Additionally, one of technical developers mentioned that they had some CloudOps team member's contact which helped them to coordinate and resolve cloud-related issues promptly.

5.4.1.4 Challenges:

As the PM worked as a liaison between the customer and the vendor-side to facilitate the communication and support related issues, it was challenging for the PM to manage all the dependencies timely. At times, there were quite a lot of requests at the same time, and this single channel of coordination was insufficient since it created difficulties in coordination, which resulting in frustrations among the squads. As the PO said,

"*we've a very thin channel through how the project manager who looks after resourcing at that end of it… so he's a programme team resource that sits across the whole lot*" [BPO]

While managing the request handling and support dependency, the squad needed to coordinate with the product development and the support team of the vendor. As this process involved people of different expertise and due to the organisational, temporal, and technical differences, it required extra effort to coordination things effectively. For example, one of the technical developers mentioned that whenever they needed support from the CloudOps team, they had to raise the incident through vendor's support request system, then it was assigned to any person to solve the issue. This process took longer than expected due to several factors, such as time difference, team's task load and task priorities. In such cases, the squads had to wait until the

issue was resolved that might affect the sprint progress. The SM said, "*As multi-talented people are involved, it asks more effort for coordinating, and sometimes requests may wait in the queue for support*" [BSM].

There were other possible issues that created challenges in the successful coordination. For example, in some situations, only certain people had access to particular resources and information (e.g. Jira tickets) which constrained the involvement of a resource person which relied upon the availability of that person. As the PO mentioned, while working with a vendor product team, they faced difficulty in communication flow perspective, as "*some people [only] have access to the tickets".* Because of these multi-faced challenges factors, the coordination with the vendor was a contest for the squads.

5.4.2 Dependency 2: Inter-squad Dependency

Another problematic coordination was the dependencies between the collocated and remote team members where they have high expertise and work deliverable co-dependencies. The dependencies between squads related to cross-cutting concerns such as integration and data management that were handled by separate squads, as well as functional dependencies between different value streams. This is characterised by the coordination between the squads for their expertise, work-outputs, and collaborative decision making throughout the development process. The inconsistencies between the sprint schedule of different squads and slower response to resource and work-outputs were identified as potential risks that could hold up the work progress of squads and create massive delays in release schedule. The analysis of this dependency and its associated issues are presented below.

5.4.2.1 Dependency Description

Due to the nature of large-scale software system development, a significant number of dependencies between different functional modules required coordination among multiple client's development squads and supporting roles. Each squad was responsible for certain products and focus on delivering the functionalities of those

products respectively in the new COTS system. For example, the O&M squad, the focus of this study, was responsible for the configuration and customization of two products and related modules in the ERP system, which includes, for example, the asset management module (requisition, work orders, purchasing) and the call centre module.

While working on these modules, they needed to collaborate with other squads, such as the *Architecture squad* for overall system architecture and design solutions, *Data-Hub squad* for data processing and management and *Integration squad* for integration of the functional modules and other systems integration. The primary needs identified for this dependency were classified as *Requirements*, *Expertise*, *Work-output* and *Decision Making* as shown in Table 5.5 along with the description and supporting evidence from data.

Since the ownership of product features and their customisations decisions were distributed among the squads, intense collaboration was required among them to identify the customisation requirements from multiple perspectives, such as product, feature, persona. For example, the O&M squad was responsible for identifying all customisation requirements for the customer billing and asset management products. However, they needed to coordinate with other squads to discover additional customisations that might be required from their perspective and vice versa. The release goals and planning of the release cycles were dependent on this collaborative requirement discovery outcomes.

Expertise dependencies occurred whenever one squad relied on other squad for their specialised skills that was needed about any feature and their implementation. For instance, the O&M squad members strongly relied on the Architecture squad for their expertise knowledge about the design of the existing system. They needed to coordinate with them to understand how the existing design supports any feature, and at the same time, got some solution ideas to implement that feature in the COTS system.

Table 5.5: Inter-squad dependency categories and their descriptions

SL#	Primary Coding	Description	Part of the Process	Evidence from the data
1	Requirement	All the squads depend on each other to participate in requirement discovery sessions and identify different types of customisation requirements related to workflow, UI and functionality customisation	Requirements Discovery	Squads participate in two-week requirement discovery sessions (mechanism) before ending the current release cycle to gather requirements (dependency) for the next release [BPO]
2	Expertise	Each squad has specialised knowledge and skills in specific area of the system and related features which other teams depends on	Release planning, Sprinting, Integration-Deployment	"*The data comes from the data hub squad, they refine it which requires Business knowledge*" [BTS]
		Development teams depends on SME for their expertise of the existing system	Requirement's discovery, Release planning, Sprint Planning, Sprinting	The SME's have strong technical knowledge about the current system domain (dependency) which is required to get the solution idea for developing Proof of Concept (PoC)
3	Work-output	Development squads depends on Integration squad for creating connections (collaborating with vendor-side team) to access vendor-side resources and perform integrations	Sprinting, Integration-deployment	The integration Squad works for all the value streams and each squad is dependent on their support for integrating for testing purpose (dependency) [BTS]
		squads depend on O&M squad for work order (work requirement)	Sprinting	"*Once a record is created in EAM which a master data, it becomes work order for other teams*" [BTD1]
		Data hub squad collects data from the existing system, refines them and shares with O&M squad	Sprinting	"*The data comes from the data hub squad, they refine it which requires Business knowledge*" [BTS]
		Deployment squad relies on O&M squad for developing deployment plans and share deployment instructions in Bit Bucket	Integration-Deployment	The FD and TD plans for the Deployment and develops the Deployment instructions that is

				required to share when any feature is done and put into the Deployment Bucket. [BFD1]
4	Decision Making	Development squads relies on each other to plan for the releases, negotiate dependent work priorities and perform risk management (programme level)	Release planning, Sprinting	During the Scrum-of-Scrums meetings (mechanism), squads meet in front of the Dependency board to identify potential risks (dependency) causing from the interdependencies [BPM]

Besides, development squads heavily relied on the Subject Matter Experts (SMEs) who had strong technical expertise in the current system domain, its functionalities, and data requirements. Without their expertise, it was difficult to have a shared understanding of the current system functionalities which would impact the customisation outcomes. Squads needed to coordinate with other teams to avail this kind of expertise during the sprint planning and sprinting phase.

Work-output dependencies occurred in multiple stages of the release process, particularly during sprint progressions. It occurred when one squad relies on another squad for completing the work needed which might hold up or delay other squad's work progress if not completed on time. For example, the O&M squad relied on data hub squad for loading the required data to the system, architecture squad for design solutions, integration squad for connection setup and integration activities, DevOps squad for preparing the testing and production environments and providing related supports. Other squads needed to plan and complete their activities to align with the work schedules of O&M squad to avoid dependency issues.

While managing such work and expertise dependencies, squads also relied on each other for collective decision making about the work priorities and scheduling of the activities for each sprint. Besides, squads had to coordination with each other to identify impediments in the interdependent tasks which might lead to risks. And then,

negotiate and manage them properly (i.e. timely and effectively) to mitigate possible risks. Additionally, all the development squads depended on another squad called as '*Pirates squad*' which consists of people involved in managerial activities. For example, there was a Test Manager in that squad who managed all the testers working in different squads and took decisions about the frameworks and tools used by the testers. He used to take the decisions of all the changes in those tools, and all the testers had to coordinate with him to implement the changes for respective modules or work segments. Similarly, the environment manager from the same squad was responsible for audits and security concerns and squads relied on his advice and suggestions to avoid audit and security issues.

5.4.2.2 Issues and Impacts

Several types of issues and dependency risks were identified related to inter-squad dependency which had potential high impacts in the way they were managed. The unplanned work-output dependencies that occurred during the sprint progression had higher impacts than the prior-known dependencies. For example, the dependencies identified during the requirements discovery and release planning phases were the dependencies that the squads were aware about, and they could plan and coordinate their activities to manage them properly. However, these dependencies could change during the sprint progress and squads required to develop a mutual awareness of such changes to coordinate themselves by adjusting their works. For example, if any connection issue with the vendor-side was blocking work progress of O&M squad that had cascading effect on other squads and release delivery, O&M squad needed to coordinate with Integration squad on ad-hoc basis to resolve this issue. There are two possible delays in this scenario,

1. the additional coordination will need additional effort and communication between the teams that will create some delays
2. the integration team must coordinate with respective vendor team to resolve the issue which will add some delays due to time zone difference and varying work priorities

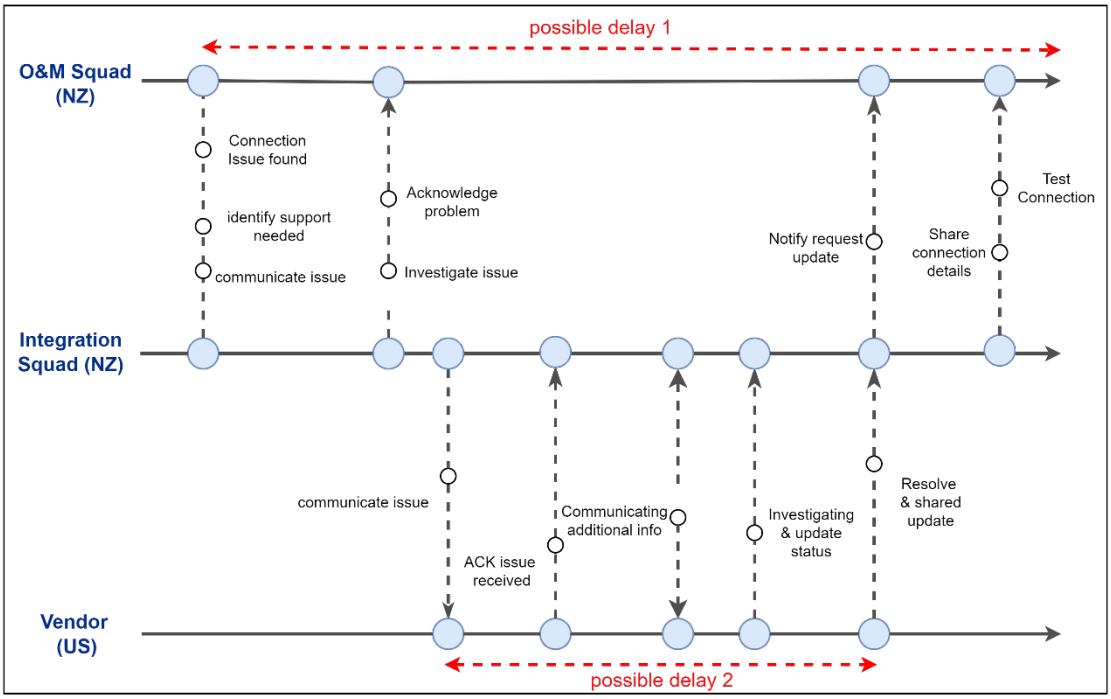

Figure 5.10: Opportunities for Delay in inter-squad dependency

As shown in Figure 5.10, this example depicts the communication complexity that involves multiple rounds of information exchange and work-outputs exchange between two collocated and one distributed team. This creates several opportunities for delays between the moment O&M squad identifies and communicates the connection issue, and get it resolved. As we estimate, this delay can range from two to three days depending on the complexity of the problem. On the contrary, integration squad must coordinate with vendor team where they have 18-hour time difference which creates opportunities for delays in communication. Moreover, the vendor might have other work priorities that might result in additional one to two days delay that can increase the overall delay for O&M squad in solving the connection issue.

Though the client development squads were co-located, several participants mentioned that it was difficult to communicate and resolve issues quickly which had potential to create delays. For example, one participant of this study shared a similar situation happened while communicating with the data hub squad and said that

"*the data comes from the data hub squad, they refine it which requires Business knowledge…there are very few times to get hold of them*" [BTS].

Though being co-located, this implies that there is higher probability of delays while coordinating adhoc requests. One of the probable reason is the differentiating work priorities between the two teams. The Data hub squad was common for all the squads, therefore had to prioritise the incoming requests which might be inconsistent with respective squads' defined priority. Similarly, the availability and participation of the SMEs who were part of another squad, had been the key to gain a shared understanding of the existing system and the customisation requirements. Therefore, their availability and responsiveness had impacts on the completion and quality of the work outputs. For example, if the SME is not present at the time of the planning or stand-up meetings, it will force the developers to make presumptions which may lead to incorrect customisations of the features. Subsequently, it will increase the chance of delay for additional work and information exchange resulting from this issue.

5.4.2.3 Analysis of Coordination mechanisms

There are a number of coordination mechanisms identified to support various types of inter-squad dependencies. Some are synchronous, and others are asynchronous, some are tools driven and some are artefacts driven. Each of the mechanisms are discussed separately in the next paragraphs.

Scrum-of-Scrums (SoS) Meeting

One of the key coordination mechanisms for resolving these inter-squad dependencies and risks was the 'Scrum-of-scrums' meeting. Squads conducted this face-to-face weekly meeting in the open workspace to discuss the interdependencies for the current sprint and the release overall. The squads shared their dependent activities to acknowledge and get feedback from the PM and based on the feedback proceeded to the next process. All the key decision making personnel from each squad, such as PO, SM attends this meetings with the PM; therefore decisions-making process was faster and the issues were resolved quickly in this weekly coordination meeting which is the impersonal mode of information exchange (Van De Ven et al., 1976).

Scrum-prescribed team meetings

Various forms of team coordination meetings prescribed in the Scrum framework were also being used to manage the dependencies between the squads. For example, to coordinate the expertise and domain knowledge dependency on SMEs, O&M squad invited them in the *Sprint planning* and *Daily standup* meetings to ensure their presence for answering queries and providing feedbacks on the designs and POCs. Additionally, the *two-week requirements discovery* sessions were used for collaborative requirement analysis which positively supported finding inter-squad dependencies.

Dependency Board and Risk & Issues Board

The squads collectively maintain a '*Dependency board*' for listing all the inter-squad dependencies for all the sprints in the release. Each squad maintains separate 'Scrum-of-scrums board' to track and manage items of interest from the dependency board. These artefacts act as 'information radiators' that makes the dependencies visible and tracking the progress of dependent works in a synchronous way. Once all the dependencies for the next release are identified in the Dependency board, squads also identify potential risks associated with the dependencies and put them in the 'Risks and Issues' board. All the items listed in this board are carefully managed and mitigated 'as you go' basis during the release progress.

Adjacent seating arrangement

Since the O&M squad had strong dependencies with another squad and needed frequent communication and coordination to manage them, the higher management decided to place them together, side-by-side in the open workspace. This technique (i.e. form of coordination mechanism) facilitated awareness of the presence of the key resources in other squad and increased informal information exchange. And thus, improved the coordination experience between the squads, as "*the process got better*" [BTD2] and reduced the possibility of delays in information sharing and work coordination.

5.4.2.4 Challenges:

While working in a large system software development, it is important how well the all the squads coordinate with each other to manage their dependencies and meet the release deadlines. Though the current mechanisms discussed above were supporting most of the inter-squad coordination needs, there were some enduring challenges as surfaced by the participants. Interview data indicated that aligning the squad's sprint activities was a perceived challenge while managing inter-squad work dependencies. One of the primary reasons of this challenges was the self-organising behaviour of the squads. For example, there was differences in terms of the sprint length and schedules (i.e. start and end of the sprint) between the squads which created inconsistencies in accommodating and managing adhoc work requests. Due to inconsistent sprint start and end dates, identifying any dependent tasks could be one week behind for other squads, e.g. if one squad starts the sprint planning on 'Wednesday' and another squad starts their planning on following week 'Thursday', the first squad will have to wait for another week before they are notified of any dependent work. It would create additional stress on the squad to decide and include new work items in the sprint. This might lead to probable delays in getting work-outputs and also impact the quality of the relationship between them.

Additionally, there was a known criticality in regulating between the security and testing framework which was challenging the coordination between the 'Framework' and 'Support' squads. This conflicting framework issue was creating tension between these two squads that interrupted the work process in other squads.

5.4.3 Dependency 3: Dependency between co-located members and remote squad members

For periods of time throughout the project one or the other of the two dedicated vendor developers (the FD and TD) worked remotely from the remainder of the squad that were co-located in New Zealand. The coordination during these periods of team member distribution was therefore a focus of our study. Table 5.6 summarises the

significant dependencies identified from the interview of both the co-located team members and remote team members along with supporting evidence from the data. These are requirement, information exchange and feedback, expertise, domain knowledge, work-output, technical, entity and team decision dependencies.

Most interviewees noted the extra coordination burdens of working with these two developers while they were away. The reduced availability and responsiveness of one of the vendor developers was critical for ad hoc information sharing and technical decisions. Besides, overwhelming amount of testing at the end of the sprint was leading to a potential bottleneck situation that had been a critical issue for co-located team coordination.

5.4.3.1 Dependency Description

Software development is a collective work-output of multiple experts in the team who needs to collaborate, communicate and synchronise their works to achieve the desired goal. These coordination and collaboration needs might be for various purpose, for example, while doing the planning poker, each team members need to form a shared understanding about the task requirements and how it can be solved. Therefore, team members need to exchange information and details of the task which preferably comes from the BA or the PO in the team. Similarly, the tester in the team depends on the developer side work-outputs to perform the testing activities; and if the activities of the developers and testers are not synchronised properly, coordination breakdowns may happen. There are also other forms of dependencies occur between the team members during the sprint progression that are crucial and if not managed properly will impact the sprint goals.

Since different persons with different personality and expertise were involved in the intra-team dependency, it is critical to harmonise their activities that relied on the continuous information exchange and feedback loop in both formal and informal ways. In a DASD context, team members might be partially distributed which will cause three forms of dependencies, the dependencies between the co-located members, the

dependencies between the co-located and distributed members and finally, dependencies between the distributed team members. Since only one of the members were working remotely at a time, the first two forms of dependencies were present in the Bluebird case. All the intra-team dependencies that were presented in Table 5.6 for both the co-located and distributed members are separately discussed in the following sub-sections.

Requirement dependency

Squad members depend on the consistent understanding of the requirements in the form of epics, user stories to participate in the estimation and planning sessions. This shared understanding also promotes knowledge of the tasks and subtasks that need to be executed. While team members are remote, they also need to actively participate and give feedback to confirm their understanding to align with other squad members. Moreover, this shared understanding helps in reaching to agreements about the other dependents and probable solution. The development of shared understanding requires continuous information and feedback exchange between the members.

Information exchange and feedback dependency

In COTS customisation process, the squad members need to discuss and share their ideas and give feedback on the intermediary artefacts such as POC (Proof of Concept). The POC is a kind of high-level design that represents whether the functionality is doable or not and at the same time, verifies that the functionality suits the requirements. The squad members' feedback on the artefacts helps to back up their shared understanding of the customer needs and decide the next work process. The tester says, "*when there is a broken part notification received, we all gather and discuss and prioritise and if needed include in the sprint*". Feedbacks also fulfil the usability constraints that is an important dependency according to the coordination theory (Malone & Crowston, 1994).

Table 5.6: Example Dependencies between co-located and remote squad members

#	Primary Coding	Description	Part of the Process	Evidence from the data
1	Requirement	Customisation requirements and release goals in the form of Epic and personas come from the PO that is required for planning activities	Release planning, Sprint planning	PO writes the Epics to translate the requirements (dependency) based on Persona and shares with the squad (mechanism) which is further broken to user stories by BA.
2	Information exchange and Feedback	The planning and development activities relies upon the information exchange and feedback from both co-located and remote members	Release Planning, Sprint planning, Sprinting, Integration-deployment	When the developer creates any POC, they share it with the tester to get feedback and ideas for correction (dependency)
3	Expertise	The development team relies on the expert product knowledge of the vendor TD and FD	Sprint planning, Sprinting, Integration-deployment	Vendor TD and FD have strong knowledge about the product that other members rely on for understanding the existing features, architectures and develop customisations (dependency)
4	Domain Knowledge	Customization design relies on shared understanding of related business goals	Release planning, Sprint planning	While writing the user stories and breaking down into tasks (during release & sprint planning), members relies on PO to get details information based on persona and their workflow (dependency)
5	Work-output	The development work by the squad members relies on the work of the other members (both co-located and remote)	Sprinting	TD of the squad loads the data in the system, which is supplied by other squad members (dependency)
		Testers' work depends on the completion of customization coding by the developers	Sprinting	Tester approach to TD to create a random test schedule (dependency) and share with him to reduce the stress at the end of sprint.
6	Technical	Remote customization work relies on external access to the shared development environment.	Sprinting, Integration-deployment	While working remotely, members need to use remote connections (dependency) to access the

				development environments that needs to be setup and permission.
7	Entity	Scrum activities depend on participation of co-located and remote members	Release planning, Sprint Activities, Integration-Deployment	"*When there is a broken part notification received, we all gather and discuss and prioritise and if needed includes in the sprint*" [BTS]

Expertise and domain knowledge dependency

The squad members had a strong dependency on the two dedicated vendor developers, in terms of both sharing their product knowledge as well as their technical skills and work capacity. The two vendor-side TD and FD members had specialised knowledge of the new COTS system and technical aspects associated with the customisations. For example, whenever the developers need any clarification of any UI or business process customisation, they would consult with the vendor FD. The FD is expert in Functional consultancy, who has knowledge about the business process to manage the UI and aligning the software features, would perform functional analysis and provide ideas for implementation. Squad members maintains a constant communication with him for getting technical details about the product modules, feedback on design and work-outputs, guidance for configuration and customisation solutions. At the same time, the vendor-side TD has technical skills to solve customisation issues, write codes and scripts required to develop any critical customisations. Most of the participants revealed that they heavily relied on both members which require extensive amount of communication which is strenuous while they are working remotely.

Work-output dependency

The squad members have cascading work deliverable co-dependencies between the remote worker and the rest of the team. Squads' joint work collaboration contributes to the effective utilisation of the team knowledge and expertise. While working in the inter-dependent tasks in different phases of the release cycle, squad members rely on the work-output from other members. For instance, during the release planning phase, BA

needs the Epics written by PO to translate into technical requirements in the form of user stories. These User stories are the basis of sprint planning sessions that other members use for estimation and developing acceptance criteria. Similarly, development and testing activities require strong work output coordination during the sprint progression. For example, tester's work relies on the timely completion of development and unit test from the developer, *"…after the UT, we send it to QA for testing, and they test it based on acceptance criteria*" [TD]. The tester also requires a random test schedule from the developer's side to plan and organise his activities. In the deployment phase, the vendor FD and TD jointly create deployment plans and develops the deployment instructions in a specific format (i.e., Business Object Document BOD) without which deployment cannot be performed. While some of the squad members are working from globally distributed locations, this type of dependency between tasks needs to be acknowledged and managed in a timely fashion to avoid any disruption of the workflow.

Technical dependency

While working remotely, the TD and FD members required remote access to the development resources to perform their daily activities. The remote working condition relied on the connection setup and access permission that needs to be arranged. Due to the company wide security policy, these technical configuration and setup work needed to be completed timely to avoid any interruption and work completion delays. Since these types of technical requirements only applicable for the O&M squad's distributed team members, these were considered as intra-team dependency that required proper coordination.

Decision making dependency

There are several types of individual and group decisions making is necessary in different phases of the process. For example, the PO is the central decision-making authority who decides the priority of the user stories in the backlog. He is also the authority to select the owner of the user stories, providing direction and advise for

resolving internal and external issues and take decisions about the blockers during the sprints. The owners of the stories take decisions about the internal and external entities that needs to be involved based on task-subtask dependencies. The deployment of the squad's work deliverables depends on the vendor-side TD and FD who has the authority for planning the deployments and related instructions.

Entity dependency

All the local squad members depend on the coordinated participation of each member in the scrum process activities. Squad members participate in estimation and planning process to give their feedback and develop a mutual understanding of the work process and requirements. The local squad members depend on the remote member's presence and participation to perform the planning and task estimation process. The time of these sessions needs to be adjusted based on the time and availability of the remote members that require prior negotiation.

5.4.3.2 Issues and Impacts

The most critical issue in this complex multi-site, multi-time zone work settings was maintaining a continuous communication to avail required technical specialisation on solution development and bug fixing that caused potentially delays because it was "*harder to share information*" and took longer time, e.g. "*Ten mins work can take 30-40 mins…*" [BTS] in case of remote team member. Though the FD was quite responsive and available most of the time, there was a delay in getting Adhoc decisions and feedback due to the time difference which was inevitable. For example, when there was a collaboration required between the FD and TD, then "*we need to coordinate… agree on a time*" [BTD1] before they can work together, and this scheduling required one-two days based on their availability which was a time delay that was required only for global coordination (as shown in Figure 5.11).

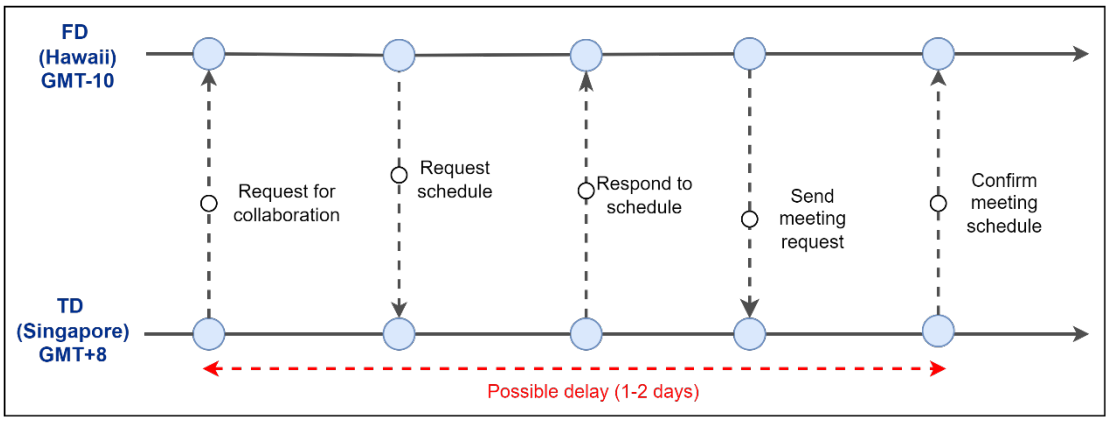

Figure 5.11: Example time delay in global communication

Another critical issue identified in the testing phase of the sprint which had high potential to create risks. As per the current process, a large amount of testing work started two days before the sprint end. The sudden work pressure created stress for the solo tester and made it difficult to prioritise testing works that could lead to potential bottleneck situation. Moreover, significant amount of discussion and information sharing was necessary to have a shared understanding of the changes made, which was amplified by the number the defects identified during testing that would impact the quality of the output. For instance, if there were 15 anomalies identified by the tester that requires minimum 15 mins communication for each one would cost almost four hours, i.e., half of a working day. For all the confirmed bugs, it would be difficult for the developers to diagnose and solve the issue within the remaining time of the sprint. In either case, the squad would suffer from poor-quality work outputs (similar to the issue portrayed in Figure 4.19 in Chapter 4) that would cause poor system performance resulting from technical debts.

5.4.3.3 Analysis of Coordination mechanisms

The main coordination mechanisms that were used to address dependencies between local and remote members were the scrum-prescribed meetings. The squad used short-iterative sprints to ensure a manageable number of tasks were performed by the squad members. For managing the work dependencies, it was important to break down the tasks and assign them to squad members and track progress regularly to locate

any further dependency or impediments. These task breakdown, estimation and self-task allocation activities happened during the sprint planning sessions, and further coordinated using the daily standup meetings. In terms of the participation of remote team members, these meetings involved different forms of technology-mediated communication either one-to-one or in the group.

Several agile practices also supported resource related dependencies. For example, during the daily standup meetings squad members shared their requirement of any document or material so that everyone got acknowledged and could further coordinate the way to resolve them. By ensuring the participation of all the squad members in different group sessions like release planning, sprint planning, daily standup, retrospectives, the squad managed their knowledge related dependencies. The scrum master of O&M squad uses a '*Fist to Five*' technique to measure the confidence of the team members about the probability of completing the sprint goals on time. The scrum master gauged the consensus in the beginning and in the middle of the sprint to get the agreement of the squad about the sprint progress. An average confidence of three or below was considered alarming that required urgent attention to identify and resolve the issues hampering the sprint progress.

The organisation maintained an open workspace where all the development squads seated in the same floor and squads having higher dependencies were seated adjacent for easier access to knowledge and expertise. Moreover, putting two separate wall clocks indicating remote member's time helped to improve the sense of their availability and correct time to communicate with them. As the remote members spend a significant amount of time with other members in a co-located space, they already developed the shared awareness of other members expertise and their skills. They also developed a mutual trust that helped them to know each other's preferences while approaching to other' whenever needed. For example, the tester mentioned that he "*…would prefer to call rather than slack, cause it's prompt*" [BTS]. For ad hoc feedback

or discussion, squad members also communicated via email, slack, or WebEx and in some cases, skype calls.

The SM of the squad scheduled these intra-team coordination meetings to ensure the participation of the remote members. While working from Hawaii, the FD shifted his working hours to align with that of NZ-site members. He was almost always active and open for communication using slack and at the same time, maintained a continuous open WebEx channel for creating an ambience of co-located work environment. According to the participants, these additional efforts made the coordination process easier and smooth.

5.4.3.4 Challenges

Although several pre-planned and adhoc coordination mechanisms were applied for coordinating the local and remote members, there were still challenges in the way of coordination. One of the reason was that the coordination depended on knowing if the remote person is 'available or not', "*sometimes challenging for getting adhoc decisions, maybe he is not there, or not in front of the PC, so can't respond*" [BTD]. Even if the person's status is showing available in slack, he might not be interested in engaging the conversation as it might interrupted his workflow. Therefore, developing the shared '*awareness of presence*' was challenging because it was difficult to determine when the remote person is available and, at the same time, ready for communication.

Another challenge that was creating frustration in this global coordination was the additional efforts and initiatives which was required to subside interruptions caused by time zone differences and technical complications. For example, while conducting conference calls for group sessions, the remote person always got priority to discuss his issues to avoid interruptions or disconnection due to technical reasons. Additional coordination difficulties such as, two remote team member staying next to each other in the same hotel, but one was available all the time and other was not, were also common in distributed coordination.

5.4.4 Dependency 4: Technical Dependency

Several technical aspects of the product customisation produced a number of dependencies that had impacts on the execution of development and deployment activities in the process. The difficulties in data extraction from the legacy system and data format accuracy were the potential risks identified in this technical coordination. Additionally, the accessibility issues due to security concerns have been a persistent challenge in this case.

5.4.4.1 Dependency Description

A technical dependency is a common type of dependency in software development which occurs when a technical aspect of the development needs to be managed properly that may affect the development progress. In this COTS customisation process, the ERP system had multiple product modules that were interdependent on each other for their operations. These dependencies required synchronisation of data or actions or exchange of execution instructions etc. For instance, O&M squad was involved in customisation and implementation of two functional products (i.e., IPS and EAM) that had a strong data exchange relationship between them as illustrated in Figure 5.12. From the illustrated inter-relationship, we could extract three specific types of technical dependencies, 1) data extraction dependency between current system and IPS module, 2) master record creation dependency between ION and EAM, and 3) work order dependency between EAM and related modules.

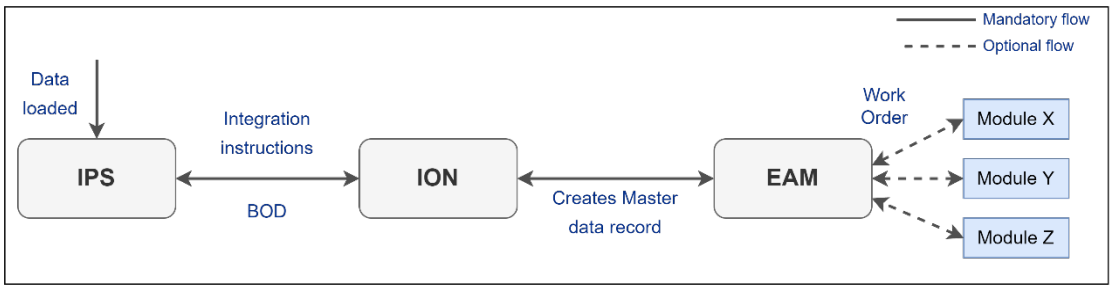

Figure 5.12: IPS-EAM Technical Dependency

Initially, the developers had to export data from the existing system and load them in the IPS module for its operations which represents the first dependency. Once the data

was being loaded, the vendor-side developers (FD & TD) could generate the integration instructions, which needed to be in a particular format (i.e., BOD). These instructions were required by the integration module (i.e., ION) for creating master data records in the EAM module based on the data loaded in IPS as represented in the second dependency. Therefore, the EAM module functionalities relied on the successful data insertion and integration instructions to produce work orders to be used by another module as mentioned by the third dependency. For example, a master data in EAM module contains several asset related information that was used by the procurement module for its purchase operations. Due to this complex technical interdependency, appropriate coordination mechanisms should be applied so that the modules could work properly and produce desired outputs that other modules relied on.

Another significant dependency was related to the testing environments that required proper coordination. There were separate sandbox environments configured for development, testing, QA, user acceptance testing (UAT) and production. Though the activities performed in each stage is different from others, their environments should be same to ensure the accuracy of the outputs produced in each environment. Each environment needed to be constantly updated and synchronized with other environments so that they have uniform setup, and all types of testing were performed under the same configuration. Any difference in the environments would result in conflicts or mismatch in the outputs that would require additional time and efforts to resolve the issues.

In terms of system accessibility, a technical dependency was identified as additional configuration and permission required to access both the old and new systems when squad members are working remotely. Due to the security concern, the systems were not accessible outside the office location. Remote members must go through proper authentication, and configuration and setup process in both client and vendor-side to allow access to the systems. All the technical dependencies identified in Bluebird are illustrated in Table 5.7 with their descriptions and example evidence from the data.

Table 5.7: Summary of Technical dependencies identified in Bluebird

SL#	Primary Coding	Description	Part of the Process	Evidence from the data
1	Inter-module dependency	One module relied on another module for its functionality and operation that might have rolling affect in other modules of the product	Sprinting, Testing, Deployment	The tester checks the new development affects other parts of the IPS and EAM. The BA facilitates and manages the work orders in EAM that other module relies on.
2	Technical Environments	The technical environments and its configurations being used in different phases of the development need to be synced; otherwise, there is a high chance of delays and conflicts in release progress	Development, Testing, Integration, Deployment	"*Now we'll have sandbox Dev, QA, UAT and production. Issues with having UAT when it first got stood up, they took a copy of production into it. And they realised that there was some stuff that hadn't been configured. So, it's been constant chasing our tails*." [BPO]
3	Access Permission	Additional configuration and access permission was required for the remote members; otherwise, could not access to the systems (both old and new) while working remotely	Sprinting, Development	Due to the security concerns, setup and permission were required for getting access to the systems from outside environment which was difficult.

5.4.4.2 Issues and Impacts

The first issue that was identified relates to the strong dependency between IPS and EAM module of the product. First, this technical dependency consists of multiple steps of actions in which multiple squad members are involved. For example, IPS module depends on the data (i.e., customer billing data) that needs to be loaded by the O&M squad developers. This data should be extracted from the existing system and processed by the Data hub squad before being loaded into the system. Several interviewees mentioned that it was difficult to export or process data from the existing system which might create delays that would impact the development and testing

activities. Because of the asymmetrical data structure, it takes longer to process the data collected from 2,000 assets which can potentially cause uncertain delays. Moreover, the data hub squad "*takes too long*" to respond queries, and at times differentiated work priorities also hold up data processing that would have cascading impacts on inter-squad dependencies (as discussed previously) and subsequent work schedules. Because of these multi-part, multi-team dependency structure, this inter-module dependency was prone to risks of causing potential delays which can block the work progress of multiple squads.

In terms of the environmental dependency, one of the potential risk factors was the inconsistency of the configurations between the development, testing, QA, UAT and production environment which could lead to delays in the customisation process. For example, the PO recalled a recent problematic incident where there was a mismatch identified in the implementation of a new UAT environment from an existing production environment. While troubleshooting, it was found that the UAT was not configured correctly as "*there was some stuff that hadn't been configured*" [BPO]. As a result, the Environment Manager squad had to do multiple rounds of communication with O&M squad, which the PO verbalised as "*constant chasing our tails*", to identify and fix the anomalies. And because of such time-consuming coordination effort, there was a delay in the release delivery cycle. Another critical impact of such issue could be deteriorating relationships between the squads. Since it was evident that there was a dispute in detecting '*who was responsible*' for that issue, it would potentially impair the relationship and mutual understanding between the squads.

5.4.4.3 Analysis of Coordination mechanisms

Several coordination mechanisms were present in the current process that supported the technical dependencies. The scrum-prescribed team meetings, such as sprint planning, daily standup meetings are the primary mechanisms used for identifying and managing the inter-module dependencies. Team members collectively identified the data and integration requirements for specific customisations and synchronise the

activities to avoid potential blockers. The peer review done at the code level also helped the team members developing a shared awareness of the technical relationship between the modules and how their code changed had impacted the modules.

In case of any unexpected issue, resulting from the technical dependencies, occurred during the sprint, squad members escalated those issues in the daily standup for ad hoc coordination. Slack and WebEx were used for general and ad hoc communication with remote members to manage the activities involving the technical dependencies. For example, the integration instructions were created by the vendor-side FD and TD, and other developers required to coordinate with them for this activity. The technical dependencies involving multiple squads, the weekly SoS meeting was the mechanism used for sharing and negotiating activities associated with the technical dependency.

5.4.4.4 Challenges:

The main challenges associated with this dependency were the inconsistency between the old and new system data requirements. Due to the structural inconsistency between the systems, data extraction and processing activities took longer which interrupted the activity flow in dependent modules. Besides, data collection and processing for 2000 assets was time consuming which had been a challenge for the Data Hub squad. Adoption of agile methodology aided in improving the situation by dividing the data processing and loading into 400 assets at a time. Participants also reported that there was a continuous friction between O&M and Data hub squad because of this slow data processing and delayed response which was another enduring challenge in this technical dependency coordination.

In terms of connectivity and environment related issues, several participants mentioned about their struggles while working outside from the organisation premises. In this global COTS customisation process, the remote member needed to get smooth access to the code repository, DB connections and other environmental settings which was problematic due to security constraints. This issue influenced the remote members' work-quality that could potentially hamper the overall squad progress.

5.5 Interpretation of the Findings

This section discusses the findings and highlights the insights to answer the key questions of this study. At the same time, we have discussed the lessons learned and based on that, made some suggested recommendation for practitioner and researchers.

5.5.1 DASD Dependencies

All the dependency types coded under each key dependencies (i.e. Dependency 1, 2, 3 & 4) are grouped into three high-level categories using Strode's taxonomy: *Knowledge*, *Activity* and *Resource* (D. E. Strode, 2016) and presented in Table 5.8. A knowledge dependency occurs when there is a kind of requirement, expertise, domain or task specific knowledge is required to complete any activity and without this information, there is high probability of delay. There were several causes identified that could lead to delays in getting required knowledge. For example, the unavailability of the vendor (due to time zone differences) at the time of need for knowledge dependencies could cause long delays (a day or more) between knowledge seeking, sharing and clarification interaction sequences due to the time zone difference. On the contrary, the reduced responsiveness and availability of the remote worker was causing delays for satisfying adhoc information needs. Several participants also shared their troubles in getting required expertise since they did not know who has the skills needed. Due to this lack of awareness, identifying remote expertise, communicating with them, and acquiring required expertise required additional amount of time that potentially caused delays. Besides, miscommunication due to language difference and lack of shared understanding about the required information had been a prime issue for delays in managing knowledge dependencies.

Table 5.8: Categorisation of key Dependencies and their impacts

Dependency Type adapted from Strode (2016)	**Dependency Types**	**Potential Impact/Risks**	**Cause of the Issues**
Knowledge Dependency	Requirement Expertise Domain Knowledge Task knowledge	Presumptions Delays in work completion Delay in Project schedule	Reduced availability and responsiveness due to time zone difference. Lack of knowledge about remote expertise. Inadequate opportunity for adhoc communication between squads. Increased risk of misunderstanding. Language and cultural difference.
Activity Dependency	Work-output	Delay in getting work-output	Misalignment of work style or process. Differentiated/Competing work priorities. Uncertain work progress. Additional effort and time required to coordinate with remote member.
Resource Dependency	Authority Entity Technical	Delay in work completion Poor quality work output	Reduced availability and responsiveness. Testing becomes a bottleneck. Firewall and security create barrier.

An activity dependency occurs when the output of a previous or concurrent activity is required to complete a task-in-action and without the output the task cannot be completed. The potential delay in getting dependent work-outputs was a key risk identified that might be caused by work style or process misalignment, uncertain progress of dependent works, competing work priorities and additional efforts while coordinating with remote members.

A resource dependency occurs when a technical or authority resource is not available which can potentially create delays or poor-quality work outputs. The primary reason of the dependency risks associated with this type of dependency was the reduced availability or responsiveness of the resource person who either had decision making authority or required knowledge and expertise. This was compounded by the fact that

the solo tester with squashing work pressure was creating potential bottleneck and struggles in the sprint progression.

It is noticeable that several of the dependencies involved distributed parties having a minimum of six hours' time difference. Two of the dependency types (i.e. knowledge and resource dependency) have impacts directly linked to dispersions factors such as time zone, and language and cultural difference. Nevertheless, the work-output dependencies were indirectly impacted by the global distribution of the team members as additional time and effort required for coordination.

All the impacts and risks associate with these dependencies are either '*Delay*' or '*Poor-Quality*' type. For example, when any task is pending for a decision from the remote member which has been delayed due to the time difference will directly impact the work progress. On the contrary, when there is a delay in getting feedback from the vendor experts, the developers may use presumptions to complete their work without waiting one to two days for getting the response back. This has high potential for rework because of the incorrect or partially correct assumptions that the developer made. This kind of rework or additional work are form of future delays as it does not have immediate impact on the current work progress or project schedule. As discussed, while direct delays need immediate attention and effective mechanisms to resolve the issues, indirect or future delays require adhoc mechanisms to reduce the delay or likelihood of this type of delay; otherwise they can potentially turn into blockers and can cause higher impacts.

The quality impacts or risks are associated with the quality of the outputs resulted from poorly coordinated dependencies. For example, the high amplitude of work stress on the tester at the end of the sprint potentially results in poor quality work output. This kind of issues have high potential to cause distress and frustrations in the tester which will have severe impacts on both personal and project performance.

All the reasons of dependency issues are categorised into three high-level categories depending on their root causes: *knowledge*, *work* and *process management*. As shown in Figure 5.13, lack of knowledge about remote expertise, misunderstanding, and language and cultural differences are the root cause of knowledge dependency issues. Activity dependency issues are resulted from competing work priorities and uncertain work progress that are caused by work management problems. Finally, inconsistencies in the process management are causing the misalignment of work styles, inadequate adhoc communication opportunities between squads, reduced availability and responsiveness, and bottlenecks at testing. Understanding the dependency issues and their root causes would help the readers and practitioners in reducing the risks and their impacts under each category.

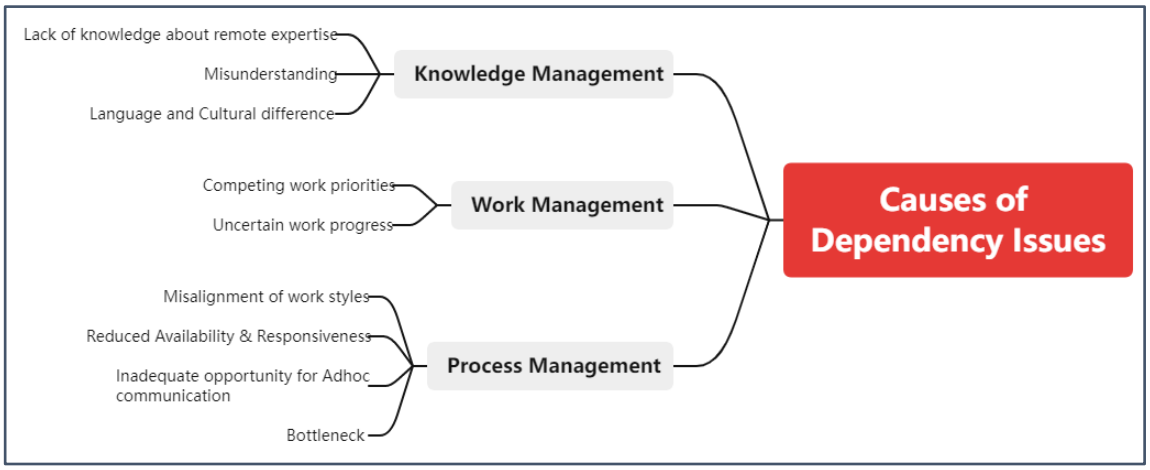

Figure 5.13: Categorisation of Causes of Dependency Risks

5.5.2 Coordination Mechanisms

The three high-level dependencies discussed above are addressed using a number of coordination mechanisms summarised in Table 5.9. It is mentionable that some activities are recurring across the mechanisms since they address more than one type of dependency. As discussed in previous chapter, coordination mechanisms consist of coordination activities, coordination tools and artefacts and structure mechanisms. For example, the two-week requirements discovery sessions were the primary coordination activity used by the squads to gain knowledge about the requirements. This requirement knowledge was transferred to the team level through the release planning

and sprint planning sessions. Furthermore, the squads could discuss and identify additional requirements during weekly inter-squad Scrum-of-Scrums meeting. During this meeting, the squads developed a shared understanding of the interdependent activities (i.e. Task knowledge), prioritised and managed them by negotiating in the presence of the PM. Any kind of expertise required from other squads were also discussed and manged in this meeting. Teams' internal knowledge dependencies were managed by scrum prescribed regular meetings, such as sprint planning, daily standup, sprint retrospective. Remote team members participate in these sessions via conference calls.

Several of these coordination activities are supported by different tools and artefacts. For example, inter-team activity dependencies for the ongoing release was listed in the Dependency board which was posted in the open workspace for clear visibility across the squads. This board was effective to radiate the inter-squad dependencies and their status. Other artefacts such as physical squad board, SoS board, DoD, DoR are posted on the wall around the squad's workspace. Examples of some physical artefacts used in this case are shown in Figure 5.16. Jira was the most important coordination tool used while coordinating both inter- and intra-squad dependencies. Synchronous conference calls via Skype, WebEx administered and liaised by the PM were the primary mechanisms used for coordination vendor dependencies. Besides, O&M squad maintained asynchronous communication channel using email and Jira based on the personal relationship with the vendor-side teams. For intra-squad coordination, Slack was used by the team members for one-to-one communication and information exchange between the local and remote team members. A summary of the coordination tools and artefacts and roles supporting the coordination activities is presented in Table 5.10.

Table 5.9: Summary of Coordination mechanisms, their effectiveness and challenges

High-level Dependency	Coordination Mechanisms	Evaluation of Coordination Effectiveness	Coordination Challenges
Knowledge dependency	**Coordination activities:** 2-week Requirement Discovery, Release Planning, Scrum-of-Scrums (SoS), Sprint planning, Fist to five, Daily standup, Backlog refinement Sprint review, Synchronous meetings via conference calls, Face-to-face formal and informal meetings for local coordination **Tools and artefacts:** Dependency Board, Risk and Issues board, SoS Board, Physical and Online Squad board, User story, Epics, DoD, DoR, Release Backlog, Sprint Backlog, Jira, Email, Slack, WebEx, Online Forum, **Structure:** Vendor and Client and multiple squad members are distributed in different time zones, Highly Interdependent squads sit adjacently, Collocated PM as Vendor Liaison, SM as Boundary spanner for squads, Open communication channel for continuous availability	- Weekly SoS meetings are effective to share understanding of expertise and knowledge requirements between squads - Scrum prescribed coordination meetings are effective in adhoc information and feedback sharing - Visible shared artefacts (e.g. Dependency board) are effective for creating awareness of dependent tasks and information radiating between squads - Personal relationship with the Client teams is effective for adhoc feedback and support - 'Fist to five' is an effective to improve task awareness and identify urgent coordination needs	- Difficulty in communicating with distributed members and vendor teams due Time zone difference - Miscommunication due to communication channel and language barriers - Restricted information flow due to limited access to resources (e.g. issue tickets, its progress etc.) - Members are reluctant to talk which makes the coordination process hard

Activity Dependency	**Coordination activities:** Scrum-of-Scrums (SoS), Sprint planning, Daily standup, Fist to five Synchronous meetings via conference calls Random test schedules, Face-to-face formal and informal meetings for local coordination **Tools and artefacts:** Dependency Board, Risk and Issues board, SoS Board, Physical and Online Squad board, Jira, Email, WebEx, Skype, Bit Bucket **Structure:** Vendor and Client and multiple squad members are distributed in different time zones, Highly Interdependent squads sit adjacently, Collocated PM as Vendor Liaison, SM as Boundary spanner for squads	- Weekly SoS meetings are effective to coordinate interdependent activities - Scrum prescribed coordination meetings are effective in identify blockers early and adhoc coordination needs - Visible shared artefacts (e.g. Dependency board) are effective for knowing the interdependent activities and track their progress - Personal relationship with the Client teams is effective for getting quick support - Communication channels are not effective to resolve ad hoc needs from vendor - Not effective in workload balancing for tester	- Reduced availability and responsiveness due to time-zone differences - Insufficient channel for communicating with vendor to resolve all squads' requests - Difficulty in coordinating between self-organising squads due to misalignment of sprint progression - Difficulty in coordinating work requests with vendor due to process misalignment and competing priorities - Firewall and security are critical for remote members
Resource Dependency	**Coordination activities:** two-week Requirement Discovery, Release Planning, Scrum-of-Scrums (SoS), Scrum prescribed coordination meetings, Synchronous meetings via conference calls Face-to-face formal and informal meetings for local coordination	- Pre-scheduled coordination meetings are effective for assuring the availability of required resources - Inefficient work cycle between the development, and testing	- Reduced availability and responsiveness due to time-zone differences - Data extraction from old system is critical and time consuming - Firewall and security are critical for remote members

	Tools and artefacts: Dependency Board, Risk and Issues board, SoS Board, Physical and Online Squad board, User Story, Epics, Jira, Email, Slack, WebEx, Skype, Online Forum, Code Management tool, Bit Bucket **Structure:** Vendor and Client and multiple squad members are distributed in different time zones, Collocated PM as Vendor Liaison, SM as Boundary spanner for squads, Open communication channel for continuous availability		- Too much dependency on PM creates difficulty in getting vendor supports

Figure 5.14: Examples of Physical Artefacts used

Other mechanisms used to manage the knowledge, activity and resource dependencies includes the structure mechanism that specifies the proximity, availability and substitutability. The organisation applied the open plan office structure for engaging workspace environment to increase the proximity between the squads, and particularly, arranged side-by-side seating arrangement for highly dependent squads for uninterrupted access to each other. For improved communication and coordination with the vendor teams, a dedicated liaison person (i.e. the PM) was assigned who was co-located with the development squads at the NZ site. In terms of availability, other than the scrum-based team coordination meetings, there was no mechanism established to facilitate continuous availability and presence awareness of the remote members. However, one of the remote members, out of the box, maintained a constant open communication channel via WebEx and slack which was effective to feel as like a collocated member which improved his availability.

Table 5.10: Summary of Tools, Artefacts and Roles per Coordination activity

Coordination Activities	Tools	Artefacts	Roles involved
Requirement discovery sessions	Jira	Requirements Backlog	POs, SMs, PM
Release planning	Jira	User Story, Epics, Persona, Squad boards	PO, BA, SMEs
SoS meeting	Jira, Email	User Story, Epics, Dependency board, SoS Board, Risks and Issue board	POs, SMs, PM
Sprint Planning	Jira, WebEx, Smartphone, Gimble stabilizer	User Story, Epics, Release backlog, Physical and Electronic Squad board, DoD, DoR	Development Team, SMEs
Fist to Five	Jira, WebEx, Smartphone, Gimble stabilizer	Physical and Electronic Squad board	Development Team
Daily standup	Jira, WebEx, Smartphone, Gimble stabilizer	User Story, Sprint Backlog, Physical and Electronic Squad board, DoD, DoR	Development Team, SMEs
Sprint Review	Jira	Sprint Backlog, DoD, DoR	Development Team
Sprint Retrospective	WebEx, Smartphone, Gimble stabilizer	Physical Retrospective Board	Development Team
Synchronous Group meetings with vendor	Jira, WebEx, Skype	Dependency board, Risks and Issue board, Electronic Squad Board	POs, SMs, PM
Open communication Channel for remote member	Slack, WebEx	Electronic Squad board	Development Team
F2F meetings (local coordination)	Jira	User Story, Sprint Backlog, Physical and Electronic Squad board	Development Team

While evaluating the effectiveness of the coordination mechanisms, the same decision rules are applied, and the results are presented in Table 5.9 for each type of dependency. It was remarkable to find that majority of the coordination activities were effective to reduce or eliminate the inter-squad dependency risks. For example, the inter-squad SoS meeting was one of the effective coordination mechanisms used to manage all three dependencies. The reason is that all the key members of each squad was present face-to-face to discuss and manage the dependencies that were visible in

the dependency board in front of them which fulfilled the conditions of explicit components for coordination effectiveness (i.e. the right thing is present at the right place and at the right time) suggested by (D. E. Strode et al., 2011). Since the key resource entities from the squad were present in this meeting, it is easier to negotiate, and make quick decisions about the inter-squad dependencies. Furthermore, the 'dependency board' and 'risk and issues board' were used for separately managing the general and critical dependencies respectively which was effective to prioritise the dependency risks and blocker to reduce their impacts. In terms of structure mechanisms, The customised seating arrangement for closely dependent squads was effective to enrich workspace awareness as suggested by (Mishra et al., 2012).

While evaluating the mechanisms for proximity and availability of remote members, the technology-mediated communication channel and the personal efforts laid by one remote worker who synchronised his time at work with the squad's and had a video link always on while he was at work in a dedicated space increased the coordination opportunities and reduced delays in workflow.

However, some coordination mechanisms were not effective enough to manage the dependencies. For example, the thin communication channel via PM for the vendor-client coordination was not sufficient to resolve adhoc needs. Sometimes getting attention and solving the issues took longer than expected that created opportunities for delays in the process, "*we've struggled at times with time frames around certain defects get raised and also these tickets get raised certain elements of those teams*…" [BPO].

Furthermore, the work cycles between the development and testing activities were not efficiently managed in the sprints which was creating high risk of coordination bottleneck. The tester confirmed that this inconsistency was notified during the retrospective sessions and the squad decided to plan random test schedules during the sprints and to give hands with the tester to reduce his workload. Despite the efforts of

the squad members, the situation did not change that much, and the tester was still stressed out at the end of the sprints which was prone to poor-quality work delivery.

5.5.3 Coordination Challenges

There are a total 14 different types of challenges identified in the coordination process those are grouped under into six high-level categories: Dispersion, Communication, Vendor coordination, Process, Organisation and Technical challenges. Table 5.11 presents the primary and secondary categories of all the coordination challenges with their descriptions. The grouping and categories are created based on the root causes associated with the challenges that will helps the practitioners and researchers to find ways to resolve them. For example, The SM mentioned that it was difficult to engage one of the remote members and get his attention when they required some information, either because he was not at work, or because he tended to get focused on his own work. Even when they did get his attention it would often take some time to get the technology set up to have synchronous communication (e.g. web conferencing). This contrasted with the other remote worker who was generally available and attentive to requests from the squad. Furthermore, he had set up a space dedicated to video conferencing so could be talking with the squad within minutes of a request. Therefore, it is considered as a communication challenge resulting from 'Lack of engagement' rather than dispersion issue.

On the other hand, it was difficult for the SM to get the velocity and capacity of the remote members while planning the sprint which is caused by the physical distance of the members. Similarly, due to the 18-hours' time difference between the two remote team members, they faced difficulty in communicating with each other for their information and work coordination. Though these two examples were related to distributed communication, temporal and physical distance were the primary reasons for these challenges; hence categorised as dispersion challenge.

Another key category specific to this case context was the vendor-client coordination challenges. As the success of the COTS customisation project highly depends on the

effective vendor coordination, it is important to understand and mitigate the challenges in this type of coordination. The coordination challenges of this dependency mainly related to the lack of well-documented coordination-related information exchange (communication) of distributed teams with differences in time zones, language and culture. In addition, since the squad members did not know (or trust, in a team sense) members of the vendor teams, another challenge was the uncertainty in who was the right person to contact for specific work-related information which aligns with the findings of (Brede & Šmite, 2012).

Table 5.11: Categorisation of Coordination Challenges

High-Level Category	Primary Category	Challenge Description
Dispersion Issue	Time difference	Time difference is creating difficulty in communicating with vendor and remote members
	Physical Distance	Physical distance is creating challenges in frequent communication and coordination with distributed team member
Communication Issue	Communication Channel	Miscommunication or reduce response due to the choice of communication channel
	Lack of engagement	Lack of engagement due to interpersonal issues such as language, culture, trust, personal skills, personal attitude and preferences
	Availability and Responsiveness	Continuous Availability and responsiveness is a persistent challenge while communicating with vendor teams and distributed members
Vendor Coordination Issue	Vendor Liaison	Solo Liaison authority for communicating with the vendor is not sufficient
	Misaligned work processes and priorities	Process misalignment and varying work priorities between vendor and client is a persistent challenge
	Lack of Trust	Squad members face lack of trust due to not knowing the vendor team members and their expertise
Process Issue	Data Extraction	Complex and inconsistent data structures of the old system is challenging the data extraction process
Organisational Issue	Aligning Self-organising squads	Misaligned sprint progression is hindering the coordination between interdependent squads
Technical Issue	Security concerns	Access to the systems from remote locations is challenging due to firewall and security settings
	Access Permission	Restricted permission to tickets is creating challenges in accessing the tickets

Furthermore, the misalignment between the Agile process of the squad and the more linear process of the vendor resources was also a source of coordination challenge. This may exemplify the findings of (Cataldo et al., 2008) regarding the impact of social and technical incongruence in software teams. The other coordination challenge mentioned by interviewees relates to the unavailability of the vendor (due to time zone differences) at the time of need for knowledge dependencies.

The challenges identified in this case are further analysed to identify their cofounding factors. Similar to the first case, all these issues could be grouped under three types of factors: contextual, technological and organisational. For example, the dispersion issues are associated with the contextual factors that includes time difference, physical distance, language and trust. All other challenges could be categorised under technological and organisational issues. The technical factors include the technology specific factors such as communication channel and tools. Organisational factors cover the issues related to management (i.e. process, roles, authority) and team (i.e. people, expertise, mindset) related issues. The categorisation of the coordination challenges to their cofounding factors is presented in Table 5.12.

Table 5.12: Cofounding factors of Coordination Challenges

Category	**Sub-Category**	**Coordination Challenges**
Contextual factors	Temporal	Time difference
	Geographical	Physical distance
Technological factors	Technology	Communication channel, Security concerns, Access permission
Organisation factors	Management	Vendor liaison issue Misaligned work processes and priorities Data extraction challenges Misalignment between squads
	Team	Lack of engagement Reduced Availability and responsiveness

5.6 Summary

This chapter has presented the analysis and findings of the second case by determining the key dependencies and associated risks and their impacts on the COTS customisation process. Various coordination mechanisms, combining the coordination activities, tools and artefacts, and structure, are being applied to manage the dependencies with the vendor and local squads and majority of the mechanisms except a few are effective to address the dependency risks. Several coordination challenges and their cofounding factors are being uncovered. The lessons learned from this case analysis are presented with improvement suggestions that would be useful to researchers since it raises a number of questions and possibilities, as well as to practitioners in similar contexts as suggestions for heightening dependency awareness and improved coordination.

Chapter 6 : Cross-Case Analysis

This chapter reports the cross-case analysis of the findings from the two case studies presented in previous chapters. The first section (section 6.1) discusses the process followed in the cross-case analysis. This is followed by discussions of the results to understand the similarities and dissimilarities, focusing on their context (section 6.2), key dependencies and their impacts (section 6.3), the primary mechanisms for coordination and their effectiveness (section 6.4), and perceived challenges (section 6.5). This chapter concludes with a summary outlined in section 6.6 that paves the way to the theoretical presentation of the concepts discussed in the next chapter.

6.1 Cross-case Analysis process

A cross-case analysis is a common approach applied in multi-case research where the findings of the data analysis are compared and contrasted for essential elements and components across cases (Khan & VanWynsberghe, 2008). The cross-case analysis results are discussed using the five aspects shown in Table 6.1 to understand their similarities and differences.

Table 6.1: Aspects of Cross-case analysis

Aspects	Description
Context comparison	Contextual Comparison is based on the elements of the Activity-based framework (i.e., work process, mode of operations, actors and sites, means of communication and coordination, means of work), including project type, maturity, project complexity, and the notion of power.
Key Dependency types	Understanding the commonalities and differences in terms of key dependencies identified in each case
Coordination mechanisms	coordination activities, tools and artefacts and structure mechanisms used to manage the key dependencies
Coordination Effectiveness	The state of coordination of each case is based on the effectiveness of the mechanisms for similar types of dependencies
Coordination Challenges	The challenges faced in each case in terms of coordination in this context

For the contextual comparison, this study has used Korpela et al.'s (2002) activity framework, as discussed in Chapter 4, along with other generic dimensions. For example, in a socio-cultural context like Global Software Development (GSD), the notion of power is an influential element that plays an essential role in communicating and coordinating harmonious actions (Wiredu, 2006). Additionally, the project type, complexity, and product maturity are essential elements for contextual comparison (Petersen & Wohlin, 2009; D. Strode, 2012).

6.2 Contextual Comparison

The contextual comparison results outline the similarities and dissimilarities between the cases based on multiple dimensions: Project, Product, Process, Collaboration and Authority. We have considered various factors under each dimension to cover a holistic view of the case contexts. One of the benefits of this comparison is that it will help us to identify the areas of improvement (i.e., misfits) in the coordination process and their relationship to the context. For example, as specified in the Bluebird case, the extensive dependency on the PM is one of the key challenges in the coordination, which is directly related to the project context. Since it is a COTS-product customisation project where the vendor has the primary authority over the product infrastructure and support, it mandates a dedicated coordinator role to facilitate the vendor-side collaboration, which was operationalised by the PM. On the contrary, no such role is identified in the Pigeon case since the project is a feature enhancement and bug fixing project where the requirements and product feature-related decisions are coordinated with the distributed POs (i.e., customers) and co-located Proxy PO (i.e., proxy customer). Table 6.2 illustrates the factors and dimensions of contextual comparison and their outcomes for each case.

There are several similarities identified between the cases. For example, both projects were of medium complexity and followed a hybrid approach (i.e., a combination of agile and traditional methods) which is common in distributed contexts (J. Barlow & Keith, 2011). While agile practices provide flexibility and improved collaboration, traditional

methods provide better control across the co-located and distributed entities. Besides, both projects are highly dependent on technology-driven collaboration methods that are used to coordinate between local and globally distributed parties located in multiple sites. For example, the Pigeon case involves five globally distributed sites to manage their software development activities, primarily reliant on the synchronised group calls mediated by WebEx and skype supported by virtual collaborative software development and issue tracking tool, i.e., Jira. Similarly, in the Bluebird case, four global sites are involved in the coordination process mediated by similar technologies to coordinate the dependencies.

Table 6.2: Summary of Contextual Comparison between cases

Factors and Dimensions	Pigeon (Case 01)	Bluebird (Case 02)
Project		
Type	Features Enhancement and Customer Support	COTS-product Customization
Complexity	Medium	Medium
Product		
Maturity	Matured	Immature
Size	Medium	Large
Software	Intelligent Healthcare IMS	ERP for Infrastructure Asset Management System
Process		
Methodology	Hybrid (Scrum + Waterfall)	Hybrid (SAFe + Waterfall)
Workflow	Scrum-based	Scrum-based
Shared Object	Release goals	Business Requirements
Collaboration		
Mode of Operations	Collaboration and Technology driven	Collaboration and Technology driven
Sites	NZ, USA, India, Japan, Europe	NZ, Hawaii, Singapore, USA
Means of Communication	Conference Call, WebEx, Skype, Email, Slack, IM, SMART tool, Shared Artefacts	Phone Conference, Video Conference, Jira, Email, WebEx, Skype, Slack, Shared Artefacts
Means of Work	Collaborative SD Tool (Jira), Collaboration Wiki Tool (Confluence), Squad Board (Virtual), Docker, Code Version Control (TFS)	Collaborative SD Tool (Jira), Collaboration Wiki Tool (Confluence), Squad Board (Physical + Virtual), Collaborative Deployment Tool (Bit Bucket), Automated Testing tool (Selenium) Code Version Control
Authority		
Notion of Power	Product Feature Dependent	Vendor-dependent
Entity	PO, Proxy PO, DM	Vendor, PM, PO

However, some dissimilarities (highlighted in red color in Table 6.2) between the case contexts have an impact on how the coordination needs are managed. For example, the software development activities in the case of Pigeon are mainly focused on new feature enhancement and customer support activities of a matured product used by clients worldwide. Whereas the development activities in the Bluebird case focus on customising a newly purchased COTS-based product to align with the existing system features. This contextual difference influences the type of dependencies, their complexities, and their impacts. For example, the poor quality work output in the Pigeon case could potentially produce a critical bug that can block client activities or damage client reputations by sharing sensitive health information. Therefore, considerable attention and effort are required in coordination before releasing or integrating any new features. In contrast, for Bluebird, any bug resulting from poor quality work output would comparatively have minimal impact since the system has not yet been released to the customer. Therefore, there is room for flexibility in quality issues in the Bluebird case compared to the Pigeon case.

Another dissimilarity between the cases is noted in the authority of decision-making. In the Pigeon case, the product owners and their representatives make most of the feature enhancement requirements and prioritisation decisions. On the contrary, in the Bluebird case, decisions made regarding product infrastructure, customisation, and integrations strongly depend on the vendor. Indeed, the client also has some authority over the decisions of the product features, customisation, and deployments, but it needs to be negotiated and approved by the vendor. Because of such differences in the notion of power, there is a difference in the entities involved in the project's decision-making. As mentioned, the POs and their proxies are the primary decision-making entities in the Pigeon case. In contrast, a combination of vendor and client-side representatives (Squad POs) are the authoritative entities involved in the decision-making in the Bluebird case.

These contextual differences have an influence on the type of dependencies and their way of coordination observed in both cases, which are discussed in the following sections.

6.3 Comparison of key Dependencies

Comparing the key dependencies between the cases is made using two dimensions: 1) dependency relationship, 2) dependency type. There are two reasons for using these dimensions; firstly, it will match the framework used in the data analysis where dependencies are described using 'who is dependent on whom' (i.e., dependency relationship) and 'for what' (dependency type). Secondly, only understanding the type of dependencies would not give a complete picture of the dependencies and, in turn, would not be beneficial for the readers. For example, instead of saying '*developers in the team need details of the user stories (i.e., task knowledge) to perform task estimations*' if we say '*developers depend on the BA to get details of the user story to perform task estimations*' would provide a better picture of the dependency where the dependency relationship is between the developer and BA and the dependency is for user story details. This later statement clearly explains the dependency relationship (intra-team dependency) and dependency type (task knowledge).

The cross-case dependency comparison outcomes are summarised in Table 6.3, where key dependency refers to a dependency that has persistent risks associated with it impacting project's progress.

The table data shows that both cases have similar dependency relationships: inter-team dependency, intra-team dependency, technical dependency, and process dependencies. However, it is predictable that there is no vendor dependency in the Pigeon case as the product is not outsourced. The *Inter-team dependency* occurs when one development team relies on another to complete its work. In both cases, multiple specialised teams are involved in the software development process and need to coordinate with each other for different types of knowledge, activity, and resource. As observed, both the case projects have prevailing issues in managing the work-

output dependencies that have been critical for inter-team coordination. In Pigeon, the inter-team coordination issues were raised due to the uncertain and poorly coordinated work requests that impacted their work progress and team relationship. Whereas, in the Bluebird case, the issues were raised by the delayed response from the data hub squad, creating uncertainty in getting the required output and delay in work completion.

Table 6.3: Summary of Dependency Comparison between cases

Dependency Relationship	Key Dependencies in Pigeon	Key Dependencies in Bluebird	Facts
Inter-team	- Knowledge (Requirement, Expertise, Task) - Activity (Work-output) - Resource (Authority)	- Knowledge (Requirement, Expertise) - Activity (Work-output) - Resource (Authority)	Low distribution impact in Case-1. Case-2 has a higher knowledge dependency between teams.
Intra-team	- Knowledge (Requirement, Expertise, Task) - Activity (Work-output, Business Process) - Resource (Entity, Authority)	- Knowledge (Requirement, Expertise, Task) - Activity (Work-output) - Resource (Entity)	High distribution impact in Case-2. Case-1 has a High Dependency on a Single resource person. Case-2 has a high dependency on vendor-side developers.
Vendor-Client	Not Applicable	- Knowledge (Requirement, Expertise, Task) - Activity (Work-output, Business Process) - Resource (Entity, Authority)	Dependency is not applicable for Case-1. High distribution impact in Case-2. Case-2 has a high dependency on the vendor coordinator.
Technical	- Resource (Technical)	- Resource (Technical)	High distribution impact in Case-2.
Process	- Activity (Work-output, Business Process)	- Activity (Work-output, Business Process)	Low distribution impact in Case-2.

Considering the dependency types, knowledge dependency had a substantial impact on inter-team coordination, particularly in the Bluebird case. A possible explanation of these findings may be related to the immaturity of the product. Since the development teams in the Bluebird case are working on the customisation of a new COTS product to replace an existing system, they had to rely on other internal teams for the knowledge about the old product to replicate the features in the new system, and at the same time,

on the vendor development teams for their expertise and domain knowledge about the new product. Participants from the Pigeon case mentioned that the module teams were mainly independent in their daily knowledge dependency and only exchanged information related to the work-in-progress. It can thus be suggested that product maturity influences the knowledge dependency between development teams. However, none of the cases reported any significant impacts caused by the distribution of knowledge dependency and related issues. Therefore, the distribution of the teams might likely have a low influence on the type of inter-team dependencies and associated issues.

The *intra-team dependency* occurs when members of the development team depend on each other for their daily activities. Both the case findings highlighted similar types of dependencies throughout the development phases. In the Pigeon case, all the team members are co-located, and the majority of the dependencies and their issues are related to DM, who is the central knowledge and decision authority of the team. Extensive reliance on a single resource entity is potentially creating risks in intra-team coordination which is an administrative issue. While in the Bluebird case, the critical issues in the intra-team dependencies involve two vendor resources intermittently working from remote locations. Other team members rely heavily on their expertise and technical knowledge of the product, which was reportedly disturbed by their unavailability and poor responsiveness due to time differences. According to these findings, it can be inferred that the distribution of the team members strongly impacted the intra-team dependencies; however, it is not the primary reason for dependency issues in this relationship.

Vendor-related dependencies apply only to the Bluebird case in which several types of dependencies are located, such as: understanding the COTS-product features (knowledge), negotiating any additional customisation requirements (decision-making), and getting technical support and feedback (work-output). Due to the nature of the project, the success of this project was heavily reliant on the effective coordination of

these dependencies. As indicated in the findings chapter, the time zone difference and lack of mutual trust had been creating delays in managing these types of dependencies. Moreover, the management of these dependencies relied upon a thin channel through a dedicated liaison person, which was perceived as a challenge in fulfilling the temporal constraints.

Technical dependencies are core to the product performance, and ineffective coordination related to technical aspects may raise significant issues and risks in the project. In both cases, the inter-module and the cascading relationship between the product components were the primary types of technical dependencies. There were additional testing environment-related issues observed in the Bluebird case. As discussed in the Findings chapters, the higher the number of modules in a product, the more complex the technical dependencies are. Therefore, the product size strongly impacted the technical dependencies and, thus, considered an influential factor for this dependency type. Furthermore, in the Bluebird case, the physical dispersion of the team member's introduced new technical dependencies (e.g. remote access permission and setup), which is considered another influential factor for this dependency.

Several *process dependencies* exist in both case projects. For example, in the Bluebird case, while getting a vendor side work support, the client-side development teams should follow a certain procedure to lodge the request and wait until they get a response from them. The vendor support team is responsible for handle multiple client requests at a time, so they investigate the request based on their work priorities which may impact development teams' work progress. Due to the nature of the project and process, it is inevitable to get rid of this process guidelines. A similar type of process constraint identified in the Pigeon case where the existing customers of the product can lodge their issues via a dedicated software created for this service. All the customers should follow this procedure to get product specific support. However, there is some room for flexibility noticed in this case, and on several occasions, customers directly

emailed the development manager to avoid the queues and get urgent support which is contradictory to the other case findings.

Process dependency issues also relate to misalignment between the development, testing and QA cycle which has high impact on delay and quality. For example, in the Pigeon case, both the sprint and release progress highly relied on the test activity output (i.e., work-output dependency) and there was a certain process that was needed to be followed to perform the testing activities to align with the development activities. Likewise, there was a similar testing cycle management issue observed in the Bluebird case where the testing activities started 2 days before the end of sprint which was creating risks of bottleneck and quality concerns. Therefore, it is plausible to assume that project type and the process guidelines have strong influence on this type of dependency. In Pigeon, the process dependencies involved distributed QA teams, which created risks of delay and poor quality output. In contrast, process dependencies only involve distributed vendor teams in the Bluebird case, which have very low impacts on the coordination.

The cross-case comparison revealed some interesting facts about the dependencies. While both cases encountered dependency issues in relation to decision-making, there was difference in the way the element of power was generated. In the Pigeon case organisation, the issue was related to one person (i.e., DM) having the authority to make most of the work-related decisions of the team leading to potential bottleneck situation. This authority has been assigned to him according to the hierarchical power structure of the management which has potential to create bottleneck. If we consider the Bluebird case organisation, there was a clear layer of dependency related to the vendor in the decision-making authority. In a COTS-based customization project, this layer of authority is expected and there is no way to get away from this control. However, in the Pigeon case, it was not true since the central authority structure was artificially created and could be changed. In fact, the management made an effort to fix this issue and re-defined the responsibilities of the Development Manager based on the

feedback provided from this analysis. Additionally, the vendor-related dependencies found in the Bluebird case were not present in the other case, which supports the previous findings and thus can be articulated that in a COTS-customisation project, the decision-making entities and dependencies are strongly influenced by the vendor organisation. Therefore, the project type can be considered as an influential factor for the DASD project dependencies and relationships.

Secondly, the comparative analysis of the dependencies between the cases indicates a negative influence from the dispersion factors, i.e., temporal and geographical distribution. As presented in table 6.3, three (3) out of the five dependency relationships have high influence from teams' or member's dispersion and interestingly, all of them are associated with the Bluebird case. The remaining two relationships indicates low influence out of which one is associated to the Bluebird case of this study. This could be implied that COTS-based customisation projects having high dependency on the globally distributed vendor might have higher effects of dispersion, particularly temporal dispersion, then that of in Feature enhancement projects. However, with a small sample size, caution must be applied, as the findings might not be appropriate in other similar case projects.

Additionally, we have observed a team-related factor that has influenced on the type of dependencies faced in the process. Different teams would face different types of dependencies depending on their expertise and their involvement in the project. For example, the Program squad in the Bluebird case is the program team and typically relates to resource type dependencies, e.g. outsourcing and managing any special developers not in-house. So the majority of their daily dependencies would be related to resource type. Whereas the O&M squad, which has expertise in operations and asset management and customer billing modules, faces knowledge (e.g. understanding of customer billings services), activity (e.g. writing SQL scripts), technical (e.g. maintain the inter-module dependencies) dependencies. Therefore, team expertise can be

considered as an important dependency factor in this context as claimed by (Espinosa, Lerch, et al., 2002).

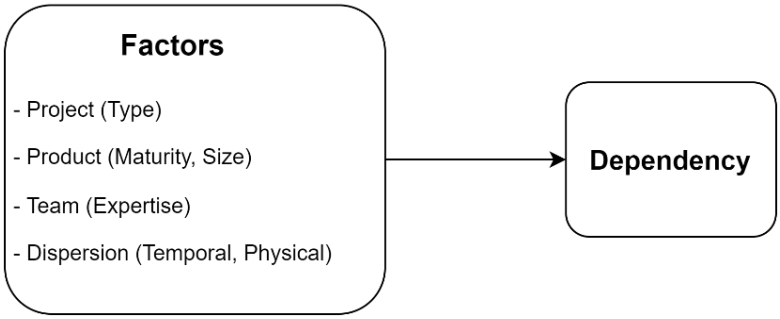

Figure 6.1: List of situational factors influencing the DASD dependencies

Overall, there are six (6) situational factors identified from the cross-case analysis that may have an influence on the dependency relationships, dependency types and the risks and impacts associated with them, those are: project type, product maturity and size, team expertise and physical dispersion. These factors are further grouped into four high-level categories based on their key properties, such as project type under *Project*, product maturity and product size under Product, Team expertise under Team and physical dispersion under Dispersion. Figure 6.1 illustrates the list of situational factors and their sub-types indicating their influence on the DASD project dependencies.

Table 6.4: Analysis of Dependency Risk impacts in both cases

	Dependency Risks	
	Delay	**Poor-Quality**
Pigeon (Case-1)	*High*	*High*
Bluebird (Case-2)	*High*	*Low*

As discussed in the findings chapters, both the projects are being exposed to significant vulnerabilities leading to risks of delay and poor-quality dependency outcomes. Table 6.4 highlights the severity of the dependency risks in the two study cases. However, the cross-case comparison indicates that the Pigeon case is susceptible to *Higher* impacts from both delay and poor-quality outputs. On the

contrary, the delay risks have *Higher* impacts than the quality risks in the Bluebird case. The reason is that the Pigeon case is working on a mature product which has live customers, therefore any delays in critical client issue fixing or any vulnerabilities created by a poor-quality work would have severe customer impacts. In case of the COTS Customisation project, there is some flexibility in the quality aspects, since the project is under development and there have been no customer impacts identified at this stage of the project. Based on this outcome, it is predictable that Mature product development projects would have higher impacts from the dependency risks than in under-development software projects.

6.4 Comparison of Coordination Mechanisms and their effectiveness

This section presents the comparison of the coordination mechanisms used to manage the dependencies identified from both the cases, except the vendor-related dependencies because it is not applicable to Pigeon case. A summary of the coordination activities, tools and artefacts and structure mechanisms for each dependency relationship, including the vendor dependency, is presented in Table 6.5 which are discussed in the following paragraphs.

6.4.1 Mechanisms for Inter-team dependency

While managing the inter-team dependencies, both the cases relied on the synchronised pre-scheduled meetings for their information exchange, dependent work negotiations and resolving issues. The only difference located in this activity is the way of conducting the meetings. In Pigeon, this pre-scheduled 'Tech Sync' meeting is conducted virtually via the conference phone call with screen sharing. Since multiple teams from different time zones are present in 'the same place', 'at the same time, this form of meeting is effective in coordinating inter-team dependencies. As mentioned by one of the SM, "…*the environment where they are all talking to each other, decisions come quickly*…" [PSM1]. But our observation of these meetings identified that the phone conference calls are challenging due to a lack of physical expressions and technical issues. Sharing photos of each other, when there are no options for video call

and they never met each other, could improve their impressions, as said, "*after he gave me the photo, I showed them a picture of [Philip]'s team, and he was thrilled*" and "*that was my way to make themselves comfortable*" [PSM].

In the Bluebird case, the pre-scheduled 'Scrum-of-Scrums' meetings were conducted face-to-face in front of the dependency board where all the key persons (i.e., SMs, POs, PM) are present to discuss the dependencies, their priorities and make decisions of their execution. Since all the teams and their representatives are collocated, this opportunity for face-to-face meetings had privileged to reduce the challenges faced in the former case context which the key difference in the way of coordination. Moreover, the always visible wall-posted dependency board was an effective information radiator of the inter-team dependencies and their status for the ongoing release. All the coordination activities identified in both cases were supported by similar types of tools and artefacts. Because of the global distribution of multiple teams, physical proximity between the teams was not possible in the Pigeon case and the Scrum Master and, if required, Product Manager acted as 'coordinator' role for all the boundary spanning activities. In the Bluebird case, high proximity between the development teams was achieved by placing them in the shared open workspace, and more effectively, putting highly interdependent teams adjacently in the same block.

Considering the effectiveness of the coordination mechanisms, the development teams in the Pigeon were struggling to coordinate the early identification and notification of their dependencies which was creating risks of delay and poor-quality outcomes. Though similar types of coordination mechanisms were practiced, these types of risks were not apparent in the Bluebird case. There were two likely causes contributing to this situation. Firstly, the inter-team dependencies in the Bluebird case were well managed through the joint release planning meeting which facilitates early identification of inter-team dependencies. And if there are any new dependencies identified during sprint, those are discussed and managed during the weekly Scrum-of-Scrums (SoS) meeting. During the SoS meeting, the squads discussed all the possible dependencies

for the current release cycle and those were distributed into sprints that are tracked using the dependency board. The development iterations (i.e., sprints) did not start until the dependencies were coordinated with other squads and tracked through the shared artefact. These triage sessions and coordination meetings were effective to reduce the chances of uncertain dependencies that could create block the release progress.

Secondly, the close proximity between the teams could be another contributing factor to eliminating this dependency risks. Since teams are collocated in the same open workspace, the PO's and SMs of the squads got opportunities for adhoc coordination to resolve urgent issues. In contrast, the two highly dependent teams in the Pigeon case were globally distributed, which was creating delays in resolving code issues on Adhoc basis. As stated by one of the developers,

"*If I know the respective team, e.g. in the last month I just jumped into the [X] team and fixed the bug. it really depends. But it's possible if the team is here or local*." [PDV1]

In this case, the code issues between the distributed teams took longer to be resolved due to the chain of communication, which could be avoided if the teams were in close proximity. Therefore, co-location of the highly dependent teams and joint planning meetings deemed effective to reduce the risks caused by unmanaged dependencies.

Table 6.5: Cross-case comparison of Coordination mechanisms

Dependency Relationship	Pigeon Case	Bluebird Case
Inter-team dependency	**Coordination activities:** Synchronous scheduled Virtual Inter-team coordination meetings (e.g. Tech-Sync meeting, QA meeting, Support Meeting) via conference call and screen sharing, Asynchronous communication via Email and Jira **Artefacts:** User story, Electronic Squad boards, Jira Issue Tickets **Tools:** Jira, Email, WebEx, Skype **Structure:** Temporally distributed and co-located interdependent Development teams, Scrum Master and Product Manager as boundary spanning activity coordinator	**Coordination activities:** Synchronised scheduled face-to-face Inter-team coordination meetings (i.e., Scrum-of-Scrums) in front of the Dependency board Asynchronous communication via Email and Jira **Artefacts:** Dependency Board, Risk and Issues board, SoS Board **Tools:** Jira, Email, WebEx **Structure:** Co-located inter-dependent development teams in Open space work environment, SM as Coordinator between teams, Highly Interdependent squads sit adjacently
Intra-team dependencies	**Coordination activities:** Agile prescribed face-to-face coordination meetings (e.g. Sprint planning, daily standup, Backlog grooming, sprint review) Asynchronous communication via Email, Jira **Artefacts:** User story, Backlogs, Electronic Story board, Software code, 'How to' docs **Tools:** Jira, Confluence, TFS, Slack **Structure:** Close proximity of the team members SM as Agile coach	**Coordination activities:** Agile prescribed virtual and face-to-face coordination meetings (e.g. release planning, Sprint planning, daily standup, Backlog grooming, sprint review and retrospective) Asynchronous communication via Email, Jira **Artefacts:** SoS Board, Physical and Online Squad board, User story, Epics, DoD, DoR, Release Backlog, Sprint Backlog **Tools:** Jira, Confluence, Email, Slack, WebEx **Structure:** Temporally distributed and co-located team members, Open communication channel for continuous availability of remote member SM as Agile Facilitator

Vendor-client dependencies	Not applicable	**Coordination activities:** Synchronised virtual meetings via conference calls, Scrum-of-Scrums meeting for coordinating with Product Manager Asynchronous communication via Email, Ticketing tool **Artefacts:** Online forum, Coding standards **Tools:** Jira, Ticketing tool, WebEx, Email **Structure:** Temporally distributed vendor teams, SM and PM as coordinator
Technical dependencies	**Coordination activities:** Tech-sync meeting, Q&A sessions during inter- and intra-team meetings, Triage Sessions, Adhoc meetings **Artefacts:** 'How to' docs, Software code **Tools:** Jira, TFS **Structure:** Close proximity between Co-located team members, Temporal dispersion between teams	**Coordination activities:** Scrum-of-Scrums meeting, Requirement discovery sessions for product level coordination, Agile prescribed team coordination meetings, Adhoc meetings **Artefacts:** Dependency Board, Online forum **Tools:** Jira, Email, Slack, WebEx **Structure:** Temporal dispersion between team members, Close proximity between Co-located inter-dependent development teams in Open space work environment, SM as Coordinator between teams
Process dependencies	**Coordination activities:** Separate iterations (i.e., Sprint) for Development, Testing and QA, Test Review meeting, QA meetings Adhoc communication via email **Artefacts:** Electronic Squad board (separate for testing and development) **Tools:** SMART tool, Jira, Email **Structure:** Development Manager as Point of Contact for customers' and QA queries, SM as iteration facilitator	**Coordination activities:** 2 days for testing at the end of each iteration, Synchronised virtual meetings via conference calls for vendor collaboration, Scrum-of-Scrums meeting for coordinating with Product Manager Asynchronous communication via Email, Ticketing tool **Artefacts:** Jira Tickets linked to vendor ticketing tool **Tools:** Jira, Ticketing tool, WebEx, Email **Structure:** Temporally distributed vendor teams,

		SM and PM as coordinator

6.4.2 Mechanisms for Intra-team dependency

There are a number of similarities and differences noticed while coordinating intra-team dependencies. Both cases have co-located team members working in the fortnightly sprints. While all the team members in the Pigeon case were co-located, in contrast, in the Bluebird case, there were two members within the participating team intermittently working from remote locations. In both cases, the development teams' coordination activities were based on Scrum and XP practices, for example, coordination meetings such as sprint planning, daily standup, backlog grooming, sprint review and those were supported by several shared artefacts such as squad boards, user stories, backlogs maintained in Jira and confluence. Emails and Slack were frequently used for both co-located and distributed member communication.

In both cases, majority of the coordination mechanisms applied for managing intra-team dependencies were effective. In Bluebird, all the coordination activities, tools, artefacts and structure mechanisms were supporting well to manage the collocated team members coordination needs. However, there was a lack of mechanism to support substitutability in the Pigeon case which had a risk of potential bottleneck. As discussed in the findings section in Chapter 4, the high availability and extraordinary performance of the DM was covering this lacking, but it is not sustainable for long term. On the contrary, although having two remote working members, there was no issues reported or analysed in the intra-team dependencies. Since the remote members have co-located working experience with other team members, they developed a good level of mutual understanding while working together and participating in socialization activities that mitigated the risks related to trust (Al-Ani & Redmiles, 2009; Brede & Šmite, 2012).

Moreover, the open communication channel using slack and WebEx facilitate better availability of the remote members as observed in the Bluebird case. As stated by the SM of the O&M squad, while two members staying adjacent rooms in the same hotel, and one was always available using the open communication channel and the other

one was not, which made a big difference in their coordination. It is analogous to the idea of continuous coordination that enables opportunities of informal communications based on mutual awareness of the activities performed by the remote member (Al-Ani & Redmiles, 2009; Redmiles et al., 2007). An implication of this is the possibility that the open communication channel could better support the continuous coordination between distributed team members.

6.4.3 Mechanisms for Technical dependency

Both the cases have indicated technical dependencies relate to strong inter-module relationships discussed in the respective finding's sections in chapter 4 & 5. One of the key mechanisms for coordinating this complex and cascading technical dependencies is to develop sound understanding of the inter-module relationships, including the product architecture, integrating modules, and inter-relationship between them. The primary benefit of this shared understanding is that developers will understand of the effects of the code changes in other parts of the system, and they will be "*lot more confident in (their) changes rather than going and changing a little bit and just gonna hoping it's not breaking anything*" [PDV2]. Another benefit includes the support for faster localization of the source of the bugs, i.e., whether the bug is created by other team's change or the local changes. Thus, Teams can start communicating with the respective team early which has been a persistent challenge for the Pigeon case.

In both cases, the coordination mechanisms addressed the technical dependencies in two levels: product and team level. The product-level understanding includes a shared understanding of '*which team is working in which part of the product*' and '*how my work is inter-dependent on other teams*'. In Pigeon, the weekly 'tech sync' meeting is the primary mechanism that supported the development of product level understanding. During this meeting, teams "*sync up with other development teams. Issues that we are having, to alert people about critical client issue, opportunity for [Team C] in US to list up issues [they] got… Not only concerns about the issues, but also for any new builds, any new changes to share… Heads up in things, e.g. investigating one issue that might*

come to the other teams" [PDM]. Similarly, the per release requirement discovery session and weekly SoS meetings are the primary mechanisms facilitating the product level understanding of the technical dependencies in the Bluebird case. However, due to the involvement of the distributed parties, there is a difference in the way of conducting these coordination meetings in the cases. In the Pigeon case, the 'tech sync' meetings were conducted over the phone conference with shared screen using WebEx as several of the teams were globally distributed. Whereas, in Bluebird, all the squads were co-located in the same building, the inter-team synchronisation meetings were conducted face-to-face standing in front of the dependency board.

The coordination mechanisms addressing the team-level understanding of the technical dependencies represent that the team members develop understanding about the product, its functionalities and the inter-relationship between the modules. Without this level of understanding, team members would not realise the impacts of their code changes or cause misunderstanding. In both cases, the coordination mechanisms supporting the team-level shared understanding were mostly agile prescribed meetings, such as planning poker, sprint planning, daily standup, code reviews. Besides, the triage meetings, formal and informal adhoc meetings also contributed to the team-level understanding of the technical dependencies. These meetings were conducted face-to-face in both the cases, expect for the duration when the two members of the Bluebird case were working from remote locations and the video calls were used to connect them in the team meetings.

6.4.4 Mechanisms for Process dependency

In both cases, the process-oriented dependencies were managed using some pre-established guidelines and tools. In Pigeon, there was a dedicated customer support team based in India which handled the first level support of the product. There were pre-scheduled synchronous meetings with that team to help and respond to their queries. Besides, the development team also received a lot of emails from the support team for information exchange in the form of queries and support tickets were created

in Jira for the issues they could not solve. Customers also had the flexibility to lodge their issues directly by email to avoid the normal support process. Similarly, in the Bluebird case, the vendor provided a support tool where the clients could create tickets for the issues or general support requests. Vendor support teams handled all the client requests as per their side work priorities. The SM coordinated the team specific vendor issues with the PM during the SoS meeting and PM acted as the coordinator to negotiate and manage the support issues with vendor teams.

To manage the testing process dependency relationship, development teams followed a pre-established testing cycle which were different in both cases. In the Pigeon case, the testing cycle is separated from the development cycle which is not ideal for test activity coordination and raises complexities such as delay in information exchange and task completion, poor performance. In the Bluebird case project, the testing activities started 2 days before the sprint end which was creating too much workload for the single tester and created a risk of poor-quality work-output. Therefore, in both case projects, the testing cycle and testing activities were not managed effectively to mitigate the risks of activity bottlenecks (delay) and poor-quality work output.

6.5 Comparison of Coordination challenges

Both cases encountered several coordination challenges related to communication, dispersion, organizational, people, process, and technical issues. A summary of the comparison of coordination challenges between the two cases is presented in Table 6.6 using the dependency structure (as presented in Table 6.3).

Under inter-team dependency, the Pigeon case project encountered several challenges when compared to the Bluebird case. For example, indifference in the choice of communication channel is resulting in miscommunication or delayed response from the remote person is related to the teams' distribution. Similarly, poor engagement in conversation or lack of trust between the team members located in different locations is also affected by language and cultural preferences. It is already established that communication is the key challenge in coordinating DASD projects (Alzoubi et al.,

2016; Dorairaj et al., 2011; J. D. Herbsleb & Mockus, 2003) which is also evident in this study.

Table 6.6: Summary of Cross-case comparison of Coordination Challenges

Dependency Relationship	Pigeon Case	Bluebird Case
Inter-team dependency	- Miscommunication or reduce response due to the choice of communication channel - Misunderstanding of requirements due to format - Continuous availability and responsiveness - Physical and Temporal dispersion - Lack of understanding at the customer level - Lack of engagement - Territorial Mindset - Lack of trust - Communication tools used	- Poor response to requests due to task complexity
Intra-team dependency	- Misunderstanding at the team level - Central Dependency structure - Added interference in the openwork environmental	- Miscommunication or reduce response due to the choice of communication channel - Continuous availability and responsiveness - Time difference - physical distance
Vendor dependency	N/A	- Insufficient channel for liaising with vendor - Time difference - Misalignment between vendor and client work priorities - Lack of engagement - Lack of trust
Technical dependency	- Inconsistencies in code Management	- Data Extraction complexity from old system
Process dependency	- Difficulty in managing process tools - Asymmetrical Responsibility distribution - Hierarchical Authority structure	- Management of Self-organising squads - Complex security concerns - Restricted permission in shared artefacts

In Pigeon, there are three different types of challenges reported in the intra-team dependency coordination. Misunderstanding in the team members due to lack of knowledge about the product, process, technology, and tools are communication related challenges. Other challenges include the central dependency on the DM that is caused by the team's knowledge dependency structure and work environment related challenge that is creating interference in the team's work process. In contrast, all the reported challenges in Bluebird case are linked to the distributed communication with the remote team members and similar to the distributed communication challenges found in the former case. The reasons for the differences in the two case findings are the distribution of the team members. As Bluebird case has some distributed members involved, their coordination challenges are primarily related to distributed communication. Whereas the Pigeon case is more challenged by knowledge acquisition, distribution and work environmental issues.

Since only the Bluebird case has significant vendor dependencies, it faced a number of challenges in this type of coordination. While coordinating via a single coordinator role, development teams struggled to get their supports and decisions. Besides, difference in vendor's location and organizational structure have contributed to number of challenges including, misaligned work priorities, lack of engagement and trust.

These coordination challenges need to be considered and can be resolved following the suggestion made by (Buchan et al., 2019). In their work, Buchan et al. proposed that to coordinate knowledge dependencies on vendor, a system is needed that would address the main challenges identified: (1) surfaces the squad's information needs in sufficient time for communication delays to be low impact on their current and near-term work; (2) provides a mechanism for communication with vendor resources where identification of the right knowledge source and knowledge negotiation interactions are low latency; (3) provides a mechanism for the squad team members to get to know and trust the appropriate vendor-side contacts; and (4) provides a presence and availability indicator for key people both in the squad and vendor teams.

Additionally, to coordinate process dependencies, it would be useful to have transparency in the progress of vendor work (e.g. bug fixing, product enhancement, infrastructure set-up) that impacts the squad's progress. It was "*not knowing*" that participants said was frustrating for their own planning. Misaligned work processes and priorities, the other challenges of process dependencies, could be improved by a practice that regularly promotes active understanding of each side's workflow practices and plans, and in particular the consequent sensitivity of work and information timeliness to future progress.

Under technical dependency, the challenges report in the case project are from two different dimensions. In the Pigeon case, the technical challenges were faced due to the lack of precise code management policy given by the organisation which resulted in the struggles for the development teams in identifying the source of bugs that were breaking their codes. In contrast, the Bluebird case suffered from the complicated data structure and system features of the old system which was creating complexities in the data extraction and exchange for the new COTS system.

The challenges reported in managing the process dependencies are primarily caused by the variations of the software development process followed in the projects. For example, the Pigeon project faced issues due to inappropriate adoption of agile methodology which resulted in asymmetric authority and unclear work distribution structure. The agile values and principles recommend self-organizing teams where the work distribution and decisions are made by the team without central supervision which was clearly lacked in the Pigeon case. Interestingly, the Bluebird case organisation reported issues while aligning the self-organizing teams for the inconsistencies in the iteration cycles. Therefore, it is advisable that the organisation encourage more internal autonomy for internal decision-making and work management. But at the same time, multiple inter-dependent self-organised teams working in the same project should adopt a '*Semi-Self-organised*' team to allow external influences on the iteration cycles between them (Kumlander, 2010). Another advantage of such Semi-Self-organised

team is that the manager role needs to be rotated which will solve asymmetrical responsibility distribution and hierarchical authority challenges. Other two process-oriented challenges reported in the Bluebird case are due to the security and restricted permissions. Since security is one of the big concerns for the organisation, it is not easy to get rid of the restrictions and organisational data access policies. But it is recommended that the organisation should lower the frictions as much as possible to alleviate the issues faced by the remote workers.

6.6 Summary

In summary, we have presented the cross-case analysis results based on the five key aspects relevant to this research. The contextual differences explored between the cases are primarily caused by the mismatch in the type of project and product maturity. Both the cases have similar forms of dependency relationships, except the vendor dependency found in Bluebird case that is exclusively linked to the project type. The cross-case dependency comparison outcomes indicate that four different types of situational factors (Project, Product, Dispersion, Process) impacts on the dependencies and associated relationships. There are two main types of risks and impacts associates with these types of dependencies: Delay and Poor Quality.

While coordinating the dependencies, a repertoire of coordination activities combined with tools and artefacts and structure mechanisms is used as coordination mechanism. The choice of the attributes and their combination varies based on the dependency relationship and distributed elements involved. The coordinator role plays an important part to facilitate boundary spanning activities with team's external elements. While evaluating the state of coordination i.e., coordination effectiveness, a combination of explicit and implicit components has been reported and their relationship to the dependency causes and impacts are established. A number of coordination challenges caused by communication, dispersion, organisational, people, process and technical related issues have been discussed with a number of possible solutions. The understandings obtained from these cross-case analysis is further used to develop a

theory of coordination in distributed agile software development context presented in the next chapter, Chapter 7.

Chapter 7 : Conceptual Model of Coordination in DASD

This Chapter proposes a general conceptual model of coordination in Distributed Agile Software Development settings based on the analysis of findings discussed in Chapters 4, 5, and 6.

The model defines four key concepts of coordination in distributed agile software development: dependency type, dependency risk, coordination strategy, and coordination effectiveness; related sub-concepts of dependency relationship and situational factor are also discussed. The relationships between the proposed concepts are discussed in detail. This model of coordination is the principal conceptual contribution of this research. As part of the practical implication, this study proposes several strategies to guide practitioners in achieving effective coordination in distributed development contexts.

The concepts of coordination presented in section 7.1 includes the concept of dependency and its antecedents and dependency risks. Then covers the coordination strategy concept, followed by a discussion of coordination effectiveness and its components. The boundaries, associations, and states of the constructs related to the coordination concepts are presented in section 7.2, followed by the prescription of effective coordination mechanisms in section 7.3. Finally, the chapter concludes with a summary in section 7.4.

7.1 Concepts of Coordination in DASD

Coordination in distributed agile software development consists of several inter-dependencies involving people of different expertise in different locations, and organisations. While coordination is defined as '*managing interdependencies between activities*' (Malone & Crowston, 1994), other dimensions such as people (individual or

group) involved in the process and their relationships are equally important to fully understand the coordination well. For example, if Task B relies on the output from Task A, then there is a sequential activity dependency between the two tasks. However, these tasks may be performed by different people or teams, and they need to coordinate themselves so that both parties have a shared understanding of what is required and when, what needs to be done to achieve that output, and how to deliver the output in a consumable form. Therefore, this research applies the dependency relationship and dependency types to form a holistic view of interdependencies and their coordination.

There may be several interdependencies that do not impose significant impacts on the project outcome. In contrast, others will have significant risks associated with them and cause severe impacts on the project's progress if they are not effectively coordinated. This study focuses on the latter type of dependencies which are labelled as '*key dependency*'. The concept of dependency will be discussed next followed by a discussion of the key dependencies identified in this study.

7.1.1 Dependency Concept

As per the framework of analysis used in this study, a dependency can be described by three key attributes: 'who', 'for what' and 'why'. The attributes 'who' and 'why' collectively describe the dependency relationship and the attribute 'for what' describes the type of dependency. This study identified different types of dependencies while analysing the cases, which were then compared with each case's findings to develop a generic concept of dependency.

The types of dependencies identified in the two cases are grouped into three high-level categories, *Knowledge*, *Activity* and *Resource*. Multiple primary types of dependencies identified in the cases are grouped together to form these high-level categories as illustrated in Figure 7.1. Each primary-level dependency category is discussed first, followed by a discussion of the three high-level categories.

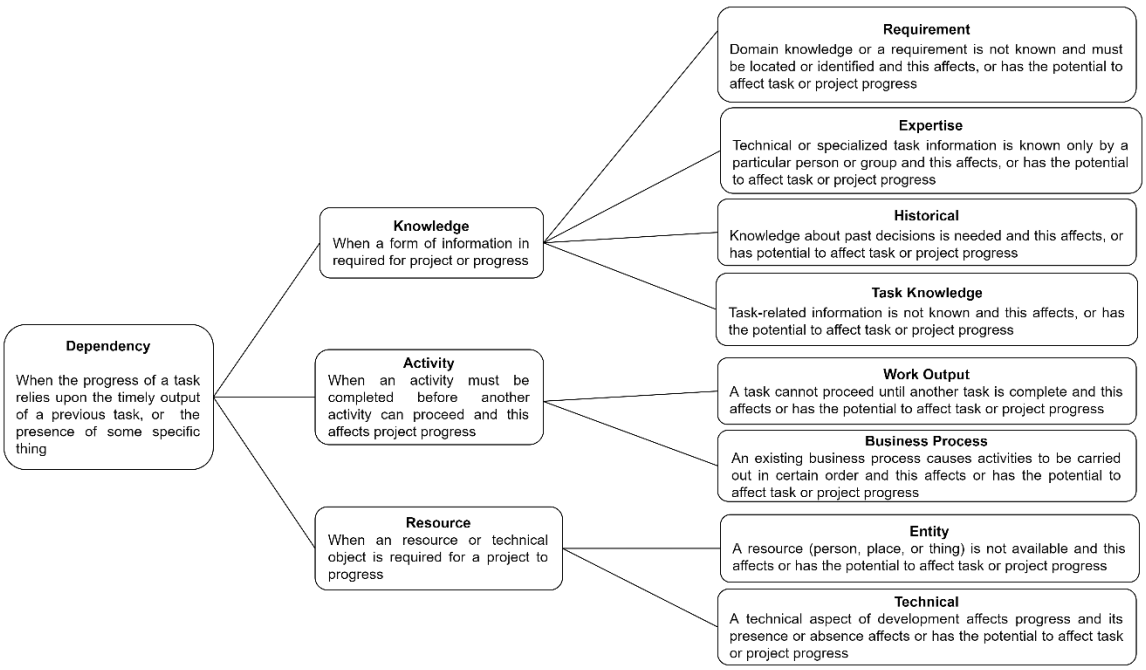

Figure 7.1: Categorisation of the dependencies in the DASD project

Requirement Dependency

Requirement dependency emerged as one of the key dependencies in both cases. In this type of dependency, "*domain knowledge or requirement is not known and must be located or identified and this affects, or has the potential to affect, project progress*" (D. E. Strode, 2016). The collection and analysis of the requirements, and developing a shared understanding of those requirements are important for the development and testing activities of the product which in turn affects the success of the project (Darnell, 2015). In a distributed agile software development project, people having the knowledge of requirements and domain expertise are physically and temporally dispersed which makes it critical to locate them, communicate with them, and collect the requirements in a form that can be converted to an outcome, e.g., new feature of a software product (Hussain et al., 2014).

In the Pigeon case, globally distributed Product Owners are the primary source of requirements and prioritizations that are acquired through synchronised group meetings. As mentioned by the SM,

"*So the [input] into the squad was product management that's our requirements, our prioritisation the whole [that are coming from the distributed stakeholders], what we are doing [in the sprint]. So, [distributed stakeholders] had a lot of focus, and they're all overseas, in different parts. We call them Product Owners*" [PSM1].

In the Bluebird case, the remote software vendor is an important source of customisation requirements for the new COTS product. "*The knowledge of the requirements comes from an understanding of the current system and from the vendor, which is then considered as Release goals*" [BBA]. A dedicated liaison role has been created to coordinate all the requirements-related to dependencies with the vendor. The requirements are recorded either as Epics or user stories that are the inputs for development teams for release and iteration planning and are managed using different shared artefacts and tools.

Besides, there are several requirements and domain knowledge dependencies that are linked to multiple dependency relationships. In both cases, different module and feature-specific teams are involved in the requirements discovery process, which reflects the *inter-team dependency relationships*. In the Pigeon case, "*[development teams] will come in [the meeting] with something like this feature was needed for this frontend feature that has to be in*" [PDM]. In the Bluebird case, the client development teams rely on each other to identify the customisation needs to implement the existing system features into the new COTS software system. The Product Owner (Bluebird) and the Development Manager (Pigeon) roles are the primary sources of requirement and domain-specific knowledge for the team which is part of the *intra-team dependency relationship*.

Since agile software methodology allows the adoption of new and changing requirements at any time of the project, coordinating these requirements during the ongoing development process is always a major challenge, particularly in GSD projects (Hussain et al., 2014). A lack of task requirement knowledge among the distributed

teams and tracking requirement changes across distributed sites are the primary challenges identified in the DASD coordination.

Expertise Dependency

In collaborative software development, a high level of coordination is required to manage people with specialized skills and successful integration of their knowledge and work to produce high-quality software (Kudaravalli et al., 2017). A combination of this type of specialized knowledge and skills is called *Expertise* dependency on which other parties rely to complete their work. In agile software development, it is defined as "*a situation wherein technical or task information is known only by a particular person or group, and this affects, or has the potential to affect, project progress*" (D. E. Strode, 2016). When applied to distributed agile software development contexts, expertise includes and relates to people who are distributed globally, which makes the coordination process even more critical due to the challenges associated with reduced availability and accessibility.

Both cases reported high exposure of *Inter-team* expertise dependency in several parts of the distributed software development process. "*Each team has an area of expertise*" [PDM] on which other teams, both co-located and distributed, rely and they need to coordinate with each other to avail the required expertise. In the Pigeon case, other teams communicate with Team-R via email for their expertise in replication service-related issues. In the Bluebird case, the SMEs are the experts of the current system and its expertise which other development teams rely on, therefore, include them in the intra-team coordination meetings. Besides, a high level of expertise dependency exists with the vendor teams on which all the other development teams heavily rely upon. Though this is specific to this case context, it suggests the potential significance of expertise dependency in a similar type of project.

Similarly, a number of *Intra-team* expertise knowledge dependencies occurred in both case projects, which requires appropriate mechanisms for effective coordination. For example, the bug-fixing activity bottleneck issue analysed in the Pigeon case was

primarily caused due to the confined expertise knowledge on the DM. A similar situation emerged in the Bluebird case due to a high number of expertise dependencies reported on the solo tester at the end of the sprint which led to a high risk of the testing bottleneck. Agile values urge the formation of a cross-functional team having overlapping expertise that can competently manage *intra-team* expertise dependency requests (Khalil et al., 2013).

In the case of large and complex systems, developing a higher level of expertise knowledge relies on strong knowledge about the system, its architecture and its functionalities, which is time-dependent. This is backed up by the statement of the senior developer of the Pigeon case, who says, "*I think the longer you go on, the more you realise how much there is to know. As technology improves, then there's so much to [learn] and do*" [PDV2]. Therefore, the longer a team member works in an organisation, the more depth of knowledge (i.e. expertise) they acquire about the system, which helps to manage the expertise needs within the team.

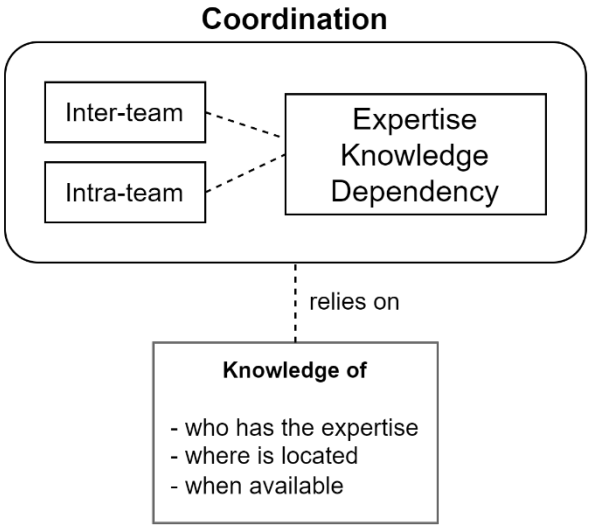

Figure 7.2: Factors linked to Expertise Knowledge Coordination

Overall, both the inter-and intra-team dependency relationships heavily rely on either co-located or distributed expertise; therefore, knowledge about the expertise knowledge sources, their locations and availability are important to effectively manage them as shown in Figure 7.2.

Task Knowledge Dependency

Task knowledge dependency was first mentioned in the early literature on coordination in software development (Kraut & Streeter, 1995), and late applied in other studies in distributed software development (Espinosa et al., 2007b, 2007a; Espinosa, Kraut, et al., 2002; Espinosa, Lerch, et al., 2002). In general, all the people working on the software development project should have a common understanding of the task, i.e. what they are building, what needs to be done, how they should organize themselves, and how their work outputs should fit together to meet the system requirements. This concept of task knowledge resembles the findings of the two studied cases. For example, the DM in the Pigeon case mentions,

"*When they do their break-down, I really wanting them to think what you are going to do to achieve this story. Because lot of the time we got this story, I've got a rough idea what this thing is, and I am just gonna stuck. When I am stuck then I realise three days later when I got only 1 more day to go, now I actually understand what it is*" [PDM].

This statement implies that the team members should have a good understanding of the task they are working on, i.e. what is going to achieve and how. Without this kind of task-specific understanding, there is a high probability of work progress delay.

Additionally, developing a shared understanding of the product, such as what are the different parts, and how they interact with each other, helps in gaining an understanding of how the changes in the code could potentially impact other parts of the system.

"*If you gonna doing something, and if you understand it [The system and its relationship], that makes the work lot more easier and you're lot more confident in your changes rather than going and changing a little bit and just gonna hoping it's not breaking anything*." [PDV3]

Such a shared understanding is beneficial in two ways: firstly, the developers will be more confident that their changes will not create any issues. Secondly, it will reduce the

chances of blocking other teams' work by creating uncertain bugs, and hence, will reduce possible delays in the project's progress. An understanding of the interdependencies, i.e. 'who else is dependent on us' and 'whom we are dependent on', would help the teams to identify the dependencies earlier, and thus has greater potential to notify and coordinate them effectively.

"*Before we don't know what other teams were doing until they have been blocked by us. it takes a while to figure out the problem, to then introduce the effects. A list can tell for ourselves we think we need to talk to these three teams because they may affect us*." [PSM1]

Based on these findings, task knowledge dependency is defined as '*a situation wherein task-related information is not known, and this affects, or has potential to affect the task or project progress'*.

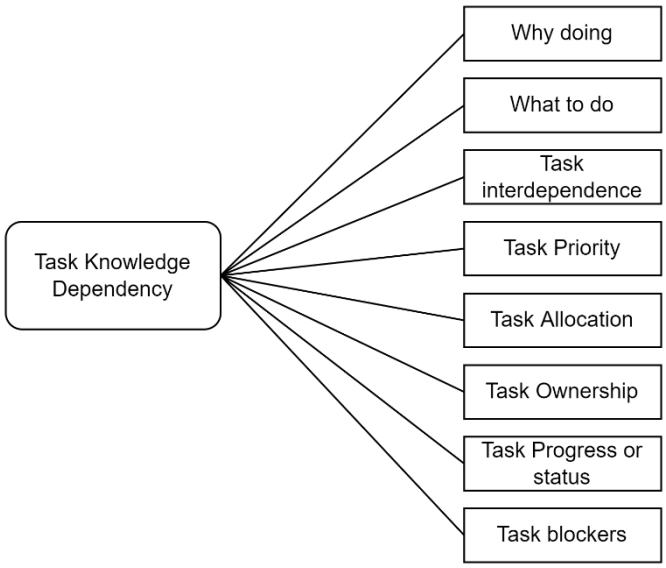

Figure 7.3: Components of Task Knowledge dependency identified in this study

Overall, this study identified a number of task-related knowledge (as shown in Figure 7.3) from both cases that play an important role in coordinating the *inter-team* and *intra-team* dependency relationships in the DASD context. The knowledge of task interdependencies, their priorities, task ownership, task status and potential blockers are the task knowledge dependencies that are related to *inter-team* dependency relationships. Whereas knowledge of why doing a task, what to do in a task, who is

doing what (task allocation) and knowing the priority of the tasks in the ongoing sprint are the task knowledge dependencies that are related to *intra-team* dependency relationship. From a software development context, task familiarity helps in reducing the subjective complexity of the task-in-action (Campbell, 1988), which enables improved coordination of tasks and their dependencies. Espinosa et al. further applied these findings in the DSD context and found a positive impact of task familiarity on 'team coordination complexity' (Espinosa et al., 2007a).

This study analysed that this type of dependency is dominant in the initial stages of the project, which require a high level of inter-and intra-team coordination, as explained by the tester in the Bluebird case, "*Sometimes as you go along, you are experienced, you don't need to ask them. But from the start when we started, I need to ask a lot*" [BTS]. As the project progresses, team members develop a cognitive understanding of the tasks, which reduces the need for overt coordination and mainly relies on implicit mechanisms, such as transactive memory (Marques-Quinteiro et al., 2013) and awareness (Steinmacher et al., 2013). This finding supports evidence from previous observations of (Espinosa, Lerch, et al., 2002) which showed that the intensity of the communication is higher at the beginning of the globally distributed software development project, and a mix of explicit and implicit coordination is required to manage them effectively.

Moreover, it is observed in both the case projects that, the majority of the task-related knowledge awareness is not present at the team level which has potential impacts on the output. Due to the temporal and physical dispersion, distributed teams and members get fewer opportunities for personal communication and mostly rely on the coordinating person for getting task knowledge which has the potential to create a knowledge bottleneck. Therefore, effective coordination mechanisms are required to improve the task knowledge at the team level to mitigate the potential impacts.

Historical Dependency

Historical dependency is defined as a situation wherein information about past decisions is required to perform a task. The absence of such information may affect or has the potential to affect the task or project progress (D. E. Strode, 2016). Historical knowledge dependency emerged as a critical dependency in both cases. In the Pigeon case, 'How to' documents were used to support historical dependencies, i.e. past work/feature knowledge. They were created for each new feature and stored in Confluence so that team members could go back and search in that document whenever they had any queries regarding that feature.

"*What we do, if it's a new feature, we create some 'How to' document and share it in Confluence. I think that is helpful, because if we develop a new feature in this sprint, and it will go to our QA (local tester) in the next sprint, it will take about one month to release, then it will be tested in another overseas team (QA testers) in the next month. So we may forget what we did in that feature. 'How to' docs work as a reminder of what we did. We always do this for the new features*." [PDV4]

In the Bluebird case, the COTS system customisation activities heavily relied on knowledge of the current system design and logic to implement the existing features into the new ERP system. For example, the team's BA had to consult with the subject matter experts (SMEs) to gain a clear understanding of the architecture and functionalities of the existing system (i.e. historical knowledge) to perform his activities effectively (e.g. to break down epics into user stories with the features). SMEs' knowledge of both business and product points of view was necessary to understand the requirements comprehensively. Additionally, developers gaining high-level design ideas from SMEs was a form of past decision that aids in developing POCs and seeking feedback before finalising the design. The O&M squad invited the SMEs to be part of the team by attending most of the team planning and coordination meetings, e.g. release planning, sprint planning, and daily standup meetings. Since these SMEs were co-located it was easy to frequently coordinate and ensure their participation in

regular meetings; however it would be difficult if the members were globally distributed impacting the task and project progress. Therefore, this type of past decisions is considered as historical dependencies that have potential impacts in the DASD projects coordination and need mechanisms to effectively manage them.

Work-output Dependency

As per the Coordination Theory (CT), interdependencies between the activities mostly involve a task-subtask relationship which is defined as '*a situation where one activity produces something that is used by another activity*' (Malone & Crowston, 1994). While analysing the inter-dependencies between software development activities, this study adopts this definition and categorises such a relationship as a work-output dependency. A significant number of work-output dependencies were identified in both cases which belonged to one of the following types:

- Output of concurrent tasks is required to complete a task-in-action
- Output from a previous task is required to start (prerequisite) or continue (concurrent) a task
- Outputs of independent tasks are required to collectively achieve a high-level goal

In the first situation, a task-in-action is waiting for output from another task and cannot progress until the output is received. In the Pigeon case, Team R could not deploy the changes until the bug was resolved and tested by Team C who are globally distributed,

"*Sometimes other teams' bug which causes our staff to break and then we get a lot of bugs, but it's all one cause of other teams' issue. Sometimes you can't deploy your fix*" [PDV1].

This type of work-output dependency is like the 'transfer' task-subtask dependency identified in coordination theory (Malone & Crowston, 1994) where the work-output needs to be transferred from the producer to consumer activity. The code management tool is being used to do the transport the output between the teams.

In the second situation, there is a specific sequence in which the tasks must be executed: a task cannot begin until a previous task is completed. A typical example of this type of work-output dependency is the ‘sprint planning session’ which cannot proceed until the epics are translated to user stories. As per the scrum framework, the user story creation task is the pre-requisite for other sequential tasks such as planning poker, self-task estimation, and team velocity calculation. In the Bluebird case, the BA is responsible for converting the Epics into user stories that are required for the sprint planning session; until the user stories are ready, the session cannot proceed. This type of dependency corroborates the ideas of sequential (Thompson, 1967) and pre-requisite constraints (Malone & Crowston, 1994) dependency.

Another sequential type of activity dependency occurs when the output from one task is required by another task but there is no chronological order of execution or task blockers involved in the process. For instance, in Bluebird, the deployment task of a team cannot proceed until the deployment instructions produced by the developers are received. The inter-relationship (i.e. two way) between the activities that is the driving force in this dependency type, and therefore is similar to the reciprocal type of dependency suggested by Thompson (1967).

The final type of work-output dependency occurs when the outputs produced by several teams and stakeholders are not directly dependent on each other; rather their work output contributes to the main overall goal, either short term or long term. In both cases, multiple development teams and other stakeholders work together to achieve a shared goal. In the Pigeon case, the shared goal is represented as the business requirement, whereas in Bluebird, it is represented as a release. In both cases, the teams are working to achieve a common objective (a task) and the success depends on the completion of the team’s activities (group of tasks). Similarly, the work-output of individual team members is required to achieve the sprint objective. This dependency type is like pooled interdependence as suggested by Thompson (1967).

In addition, the case data analysis indicates that the desirable characteristics of the work-output can be posited as: fit for purpose, shared understanding and timeliness (as shown in Figure 7.4). The fit for purpose characteristics defines that the work-output (i.e., the output of a previous or concurrent task) should be in a usable condition, meet the quality standard and expectations of the consumer task and be accurate. The timeliness characteristics are defined as the output being received within the expected timeline. A shared understanding defines that the output received is understood either before the task-in-action is started (pre-task and the need is anticipated), or with minimal delay after the dependency and coordination need is communicated to the appropriate resource (in-task, either anticipated or ad hoc and unanticipated).

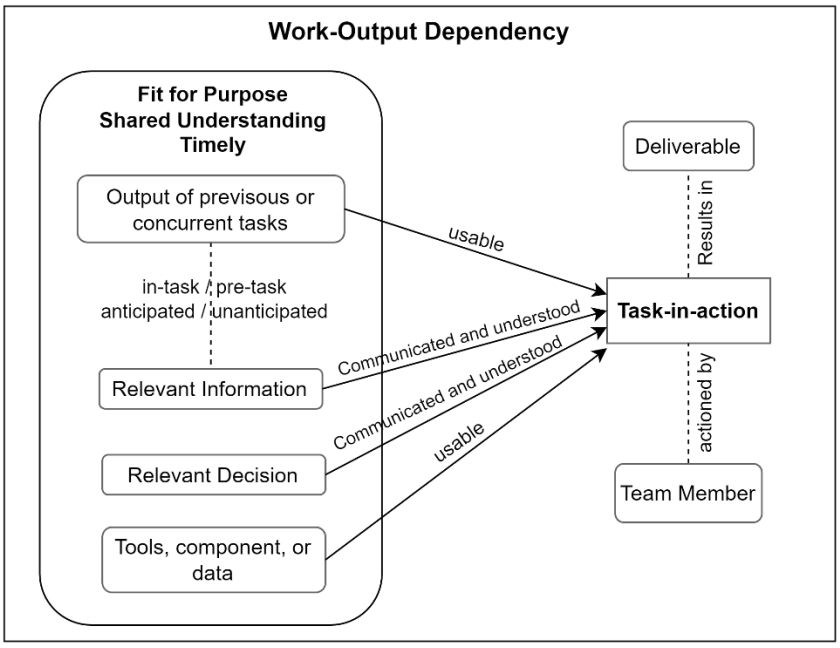

Figure 7.4: Characteristics of Work-output dependency Coordination

Overall, a work-output dependency could involve both co-located and distributed parties which have possible risks and impacts that might affect the project progress as analysed in both cases; therefore proper mechanisms need to be used to effectively manage them.

Business Process Dependency

A Business process dependency occurs when the execution of the activities is mandated by a predefined guideline or sequential order imposed by the organisation.

Certain business process, embodied within the existing technical system, could act like a dependency and needs to be managed properly to avoid potential impacts on the outcomes.

Several business process dependencies emerged in both the studied cases. For example, in the Pigeon case, globally distributed customers needed to follow a stipulated guideline to register their requests. All the customer support requests were reported through a dedicated tool, which were automatically assigned to the support team. The distributed support teams would first investigate the issues and provide support to the customer accordingly. When the support team failed, they had to transfer the requests to the development teams by creating a Jira ticket and assigning the appropriate team to investigate and resolve the issues. Though some flexibility was allowed, most of the client support activities needed to follow the above workflow defined by the business. Any discrepancy in the defined process could lead to a potential delay in resolving the issue, which might have an impact on the customer's business operations.

In the Bluebird case, a similar kind of business process dependency emerged between the client and global vendor activities. Since the COTS product implementation and management activities heavily relied on the vendor teams, there was a constant coordination need that existed across the development phases. To ease the logging and effective management of client requests, the vendor organisation defined a business process that the client side teams needed to follow. As per the process, the clients needed to log their requests through an online ticketing tool. These requests were automatically assigned to the respective vendor team based on the request type which were then prioritised and served in order. Besides, a dedicated liaison person was being assigned to coordinate the client requests and adhoc supports. Contrast to the former case, there was little room for flexibility allowed in the process. The purpose of this pre-defined workflow or guideline was to better organise and distribute the activities and reduce unnecessary delays that might impact the customer experience.

Since this type of dependency is specific to the organisation and its business process, this dependency is not commonly articulated in software development studies. Strode et al. (D. Strode, 2012) reported this type of dependency in the agile software development context. The findings of this study confirmed that the business process dependency is an important dependency type in the DASD project that needs to be managed properly; otherwise, it could have an impact on the DASD coordination (Sablis et al., 2021).

Entity Dependency

Software development is a knowledge-intensive work that involves several key resources that needs to be available and accessible in a timely manner. All these resources are broadly termed as *Entity* and the dependency on these entities is defined as a situation where a resource (person, place, or thing) is not available and this affects progress – it is a type of resource dependency (D. E. Strode, 2016). Entity dependency emerged as a common dependency in both cases.

In the Pigeon case, the critical dependency situation around the Development Manager (DM) is a good example of entity dependency. Being the key source of domain and knowledge expertise, the DM's absence or delayed response caused significant impact and knowledge acquisition bottlenecks on all parts of the development process. In the Bluebird case, the SMEs are the key resource entities on whom the development teams rely on for their knowledge of the existing system. Coordinating their presence was a key factor in translating the current system features into the new COTS system.

Besides, several entities emerged based on their authority in decision-making about the requirements, task priorities and their distributions. In the Pigeon case, the DM is the primary authority resource for making most important decisions of the team. In the Bluebird case, the Product Owner of each development team and the vendor liaising PM were the key authoritative resources. As discussed in previous Chapters, the availability and responsiveness of these decision-making entities had potential high impacts in both local and distributed coordination. In the Bluebird case, all the vendor

side communication were primarily liaised by the PM, for which his availability was crucial for organising and negotiating most of the customisation and support-related works and decisions. Similarly, the DM in Pigeon case was the primary point of contact for the work decisions coming from other teams; therefore his continuous availability was critical for the coordination performance.

In both cases, several local and online documents were deemed crucial for successful completion of task-in-action or overall project progress. For example, user stories are one of the key resources used to translate the user requirements as important development activities such as estimation, planning, testing and deployment are dependent on the user stories. In the Pigeon case, the ‘How to’ documents developed by the developers served as an important intra-team knowledge management resource for information on the system features and other searchable options. In the Bluebird case, the vendor side documentation on the system features, architecture, and coding standards served as key resources for the customisation development activities.

Overall, DASD projects face a significant amount of entity dependencies and it is impractical to achieve coordination success without the proper management of these types of dependencies. In a DASD project context, lack of knowledge of the location and availability of the required entities makes it difficult to ensure the on-time, and ready-to-use availability of the entities.

Technical Dependency

Technical dependency is in the heart of the software development process which can be defined as a situation where a technical aspect of software development process affects the progress of a task or project (D. E. Strode, 2016). Technical dependencies are dynamic in nature (Cataldo et al., 2008) and can have different dimensions depending on the product and organization. Coordination of technical dependency refers to the congruence among the technical components of the system (or system under development) and their relationships so that they all act together as a cohesive software system. Several types of technical dependencies were found as both cases

are involved in product development that consists of more than one module with complex relationships between them.

In both cases, the ownership of the software modules was distributed among different expert teams as such each module is owned by a single team and a single team may own more than one module. For example, Team R in the Pigeon case was responsible for a single module of the product, whereas the O&M squad in the Bluebird case was responsible for two product modules. At the same time, software modules have overlapping functional areas owned by different teams which potentially creates interdependencies between them. In the Pigeon case, globally distributed teams R and C had strong technical overlap between their modules. Whereas in Bluebird, two teams were placed together having strong technical dependencies between their modules.

Other technical dependency issues relate to the need for connection and synchronisation between multiple development environments configured for the developers, testers, QA, and production testing to ensure consistency of the developed solution.

"*Whenever a support ticket is raised, then you need to connect to the [environment] from where the ticket came from, they send us with the details of the connection [to the environment]. if we can't connect then we have to wait for another day until they reply back*". [Pigeon, Tester]

As per the statement, the connection to the appropriate environment was required to reproduce and understand the cause of the issues, which represented a technical dependency situation. In the Bluebird case, technical teams struggled to synchronise between the different sandbox environments established for development, QA, User Acceptance Testing (UAT) and production, which created additional work and increased frustrations between the teams. As stated by the PO of O&M squad,

"…*issues with having UAT when it first got stood up, they took a copy of production into it. Yeah. I think that we started trying to do some deployments into UAT and they*

realised that there was some stuff that hadn't been configured… It's refreshing from prod again, right? So start over" [BPO].

Remote access to the systems through an additional layer of security and configuration also contributed to issues related to managing technical dependency.

Overall, ineffective coordination of technical dependencies may result in inconsistencies between the technical environment that would create issues in continuing the software development activities and the project progress. The global distribution of the teams adds complexity to the coordination of the technical dependencies. These complexities could be reduced by developing strong understanding of the technical dependencies and their impacts, and by improving proactive and collaborative behaviour between the responsible teams.

The list of primary dependency categories derived from this study is summarised in Table 7.1 along with sample evidence from the data. The primary dependencies are grouped under three high-level categories: knowledge, activity, and resource dependencies, which are discussed in the following section.

Knowledge, Activity & Resource Dependencies

The dependencies discussed in the previous section are classified using three high-level categories based on the following decision questions adapted from Strode (D. E. Strode, 2016):

1. Is workflow at risk because there is something the team does not know?
2. Is workflow at risk because the team is waiting for a task to complete?
3. Is workflow at risk because the team is waiting for a resource to become available?

The questions are adapted based on the dependency risks concepts derived from the analysis framework discussed in 7.1.3. The primary dependencies are identified through the inductive analysis of the data, after which each dependency is passed through the above decision rules to categorise them into one of three high-level

categories. The knowledge, activity and resource dependency categories are discussed below with supporting evidence to justify their formation.

Knowledge dependency occurs when a team member needs information that is required to complete a particular task which can be in the form of requirement, expertise, historical or task knowledge. Whenever such information is needed, first its source is identified, and information is exchanged between the parties. For example, in agile software development, the sprint planning session is used to exchange information about the user stories (*requirement dependency*) needed to perform the estimation and planning of the tasks related to them.

Different types of (*expertise dependency*) are required to complete the tasks, some of which may exist within in the team while others may need to be acquired from other teams. Development team members must have specific knowledge about the task-in-action (*task knowledge dependency*) to identify other related tasks, their priorities, who needs to be involved and how to communicate them. Besides, they also need to know what might be blocking the task execution, when to expect output from other dependent tasks, and what is blocking them. They also need to know what has been done in the past (*historical knowledge dependency*) while performing any task-in-action. Each of these types of dependencies require knowing any pre-task or in-task information, either anticipated or unanticipated that affects or has potential to affect the progress of any task-in-action or deliverables from a group of tasks.

Both work-output and business process dependencies are categorised under the activity dependency, as they depict a situation where an activity cannot proceed until the completion of another activity. In the case of work-output dependency, the task-in-action needs output of previous or concurrent task must be completed in time. The work-output from a sequential task or group of tasks might also be required to complete a single task or achieve a high-level outcome. In a business process dependency, tasks must follow a process specified by the organisation or the methodology used in the process.

Table 7.1: Evidence for dependency types identified in both projects

Dependency Type	Evidence from data	Explanation	Additional Comments
Knowledge Dependency			
Requirement	"*So the going into the squad was product management that's our requirements, our prioritisation the whole, what we are doing. So they had a lot of focus, and they're all overseas, in different parts. We say them Product Owners*" [PSM1] "*He [Steve] goes for those meetings, and he would propose the things that we wanted like these are the technical improvements we need upgrade*." [PSM1]	The globally distributed product owners share their ideas and expectations, i.e. the requirements, and their prioritizations during the online meeting. These requirements directly go to the team that they work on depending on the shared priorities. The Product Manager in the team, who is the Proxy PO, would propose technical improvement (i.e. requirements) to get approval from the POs.	Both the PO and the team acts as source of requirements as identified in Pigeon case. The squads and the vendor are the primary source of requirements in Bluebird case.
Expertise	"…*part of it is very technical component, and that's why tend to be come through me to have a look at the problem, I break it down and then they go down to the team*." [PDM] "*The data comes from the data hub squad; they refine it which requires Business knowledge*" [BTS]	Team members relies on the DM for his expertise about the product and the frameworks that is required to break down the tasks. The Data hub squad has necessary expertise knowledge to process the old system data and share with other teams to be used in the new COTS system.	This type of dependency has been observed both in inter-team and intra-team dependency relationships.
Task	"*When they do their break-down they really wanting them to think what you are going to do to achieve this story because lot of the time we got this story, I've got a rough idea what this thing is, and I am just gonna stuck. When I am stuck then I realise three days later when I got only 1 more day to go, now I actually understand what it is*" [PDM] "*Meetings are the opportunities to sync where we are up to, what we are doing, status, the progress, sync understanding and coordinate the understanding*" [BTD2]	The team get together in the sprint planning session where members get the details about the task, why doing it, what needs to be done; without this understanding there is high potential to face problems later in the process. Agile prescribed team coordination meetings are the key practices to develop shared understanding about what is going on and when, who is doing what, what's the progress etc. Shared artefact such as Squad board acts as continuous information radiator about the tasks, their allocation, and status.	Task-specific information dependency is noticed in all types of dependency relationship and have high impacts in the dependency outcome. Agile prescribed frequent team coordination meetings and shared artefacts are the primary coordination mechanisms identified in both cases.

Historical	"The SME's have knowledge about the solution based on the current system domain, which is a high-level design" [BTS]	The SME's have strong technical knowledge about the current system domain which is required to get the design and solution decision of the existing system that is required for developing Proof of Concept (PoC)	Historical (knowledge) dependency is found in Pigeon only and are related to inter-team dependency relationships since they are from another team.
Activity Dependency			
Work-output	"…*whenever anyone's got any problem or they want something done, you suddenly get something someone contact you or [Robin] would say*" [PSM1] The integration Squad works for all the value streams and each squad is dependent on their support for integration for testing purpose [BTS]	The work of one specialized development team is required by another development to complete their work (e.g. data replication, connection setup, deployment instructions).	Both the cases have strong work-output dependencies in all the dependency relationships.
Process	"*Usual process is that they will raise support ticket using the system called "SMART". The support team will then do investigation, they will create a Jira support ticket, [and then] assigned it to appropriate development team"* [PDM]	The client support activities need to follow a pre-established process to better manage the activity dependencies.	Inter-team and vendor dependency relationships mostly face this type of dependency.
Resource Dependency			
Entity	"*So he's [Robin] the main gate, I suppose you could say. and everything comes in and he will filter out all the crap and condense what we need to look at"* [PDV2] "*If we plan for directions, I give options like tickets and the output and ask his [PO] opinion… what to do or what to do with the tech debts or impact users*" [BBA]	Team member relies on specific roles, e.g. development manager, Product owner, for most of work-specific decisions such as work priorities, bug confirmation and distribution.	This type of dependency mostly observed in intra-team dependency relationship.
Technical	"*We do it on top of framework, and Team C I guess takes the other half... of the backend stuffs*" [PD01] While working remotely, members need to use remote connections (dependency) to access the development environments that needs to be setup and permission. [BTD1]	Inter-module dependencies are forcing multiple development teams to rely on each other and there are cascading effects of the work-outputs produced by the teams. Team members need to get access through connection setup and permission provided while they are working from remote locations.	Technical dependencies are primarily related to technical dependency relationships.

Resource dependency defines a situation wherein a resource entity or a technical object is required for the progress of the task-in-action. An entity resource could be a person who needs to be present to perform any activity or have authority to make decisions required to perform activities. Other forms of entity resources could be physical or electronic documents required to perform any task. Similarly, the necessity of cohesive interrelation between the system components, without which the system would be unstable, are considered as technical resources. If any part of the system is not integrated or functioning properly, it would make the system unstable. Since availability of required resource is the primary concern for both the entity and technical dependencies, they are categorised as resource dependency.

7.1.2 Antecedents of Dependency types

Two types of antecedents were found to influence the dependency types: 1) Situational factors: project type, product maturity and size, team expertise and dispersion and 2) Dependency relationships: Inter-team, intra-team, technical, and process dependency relationships.

Situational factors

Project type was found to influence the type of dependencies present in DASD Projects. For example, the vendor dependency relationship exists only in the COTS-product customisation project in which knowledge, activity and resource dependencies occurred. Project type was also found to influence specific dependency types. Projects followed different process guidelines that had an impact on the process dependency relationship where each followed the execution of specific set of activities. For instance, the outsourced product customisation project in Bluebird case encountered differences in the workflow and order of task execution in comparison with the other (case). Moreover, vendors had to organise their process flow to facilitate efficient responses to clients' requests – such business process constraints forced the clients to adhere to specific guidelines to get faster response and technical support from the vendor.

Product size and its maturity was found to significantly influence the complexity and amount of the technical dependencies as they increased proportionally with the number of modules of the product. Such high complexity also increased the probability of risks and issues related to coordinating those dependencies. In the Pigeon case, the product under development was quite large which made it difficult for teams and their members to be expert in all aspects; hence interdependencies increased.

"*It's very hard to be an expert of everything, especially a company like that the software gets huge. and you can't be expert on everything" [PDV2]*

Product maturity was also found to strongly influence inter-team knowledge dependency. For example, when the product is mature, the development teams would have gained sound knowledge about the product and its features which reduced the inter-team knowledge dependencies. On the contrary, when teams were working on an immature product, they depended more on external knowledge sources and expertise for their knowledge requirements which necessitated frequent communication and coordination between the knowledge teams. For example, in Bluebird case, because they were working on a new product which was outsourced from a global vendor, they had to heavily rely on team's external knowledge resource (e.g. SMEs) to understand how the system features worked. Likewise, teams had to coordinate with other teams to identify the customisation requirements which they did during the requirement discovery sessions.

The differentiated teams' expertise is another factor that influences the inter-team dependency relationship and the types of dependency faced throughout the project. Different teams would face different types of dependencies depending on their expertise and their involvement in the project. As Espinosa et al. (2002) showed that different teams working on a project would have different expertise that would define in which part of the system they would work in, thus, the nature of their work and their dependencies would vary. Since the feature or module allocation defines the inter-team dependencies, it is indirectly influenced by the team's expertise. In the Pigeon case,

based on their expertise, Teams C and Team R owned two different functional features with strong overlapping between them which created strong inter-team dependencies. Similarly, in Bluebird, the O&M squad owned the customisation of two product features that had high dependencies with other features of the new outsourced ERP software that resulted in a high number of *inter-team dependencies*.

Physical dispersion was found to influence the technical and process dependencies in an *intra-team dependency* relationship. Additional technical configuration and access controls are imposed for the remote team members. And based on the distribution nature, development activities need to be planned and performed differently than that of the co-located members. Similarly, the temporal dispersion of the team members influenced the intra-team dependencies. For example, due to the temporal dispersion of the two distributed technical members found in the Bluebird case, the O&M squad had to alter their daily meetings to adjust to the remote members and give them priority in all the team meetings over the local members. Moreover, the time zone differences have influenced members' availability and response time, which would require adjustment in the work process. For example, the SM had altered the schedule of the sprint activities to sync with the remote members' availability.

Dependency relationship

A dependency relationship exists when two entities (person, team, software module, part of the process) rely on each other for a particular type of dependency. Four primary types of dependency relationships were identified: *Inter-team*, *Intra-team*, *Technical* and *Process* dependency relationships as shown in Figure 7.5. The vendor dependency relationship identified in the Bluebird case is not included because it is specific to the COTS-based product customisation project.

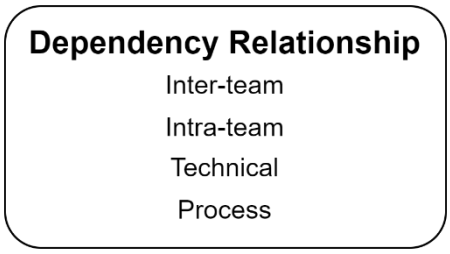

Figure 7.5: Types of Dependency Relationship

An *inter-team dependency relationship* requires coordination of dependencies between two or multiple teams. A typical example of an inter-team dependency could be the work of the O&M squad that relies on the data to be provided by the Datahub squad for completing their work. Without the data, the work of O&M would not be completed which will block their sprint progress.

An *intra-team dependency relationship* occurs when the team members rely on each other to complete their work. As seen in the Pigeon case, members of Team-R heavily rely on DM for their work and his unavailability and late response could impact the work completion.

The *technical dependency relationship* emerges when the software modules depend on each other to behave like a system as a whole. Any interruption or faults exposed in the integration of the modules may result in inconsistencies or software failure. The synchronisation between different software development environments is another example of technical dependencies identified in the project and any inconsistencies between them could cause software performance issues.

The *process dependency relationship* can be defined as the synchronisation between different parts of the software development process. A common example of such a relationship is the synchronisation required between the coding and testing phase in any development cycle. In agile development methods, complete integration of testing and development is strongly recommended (Talby et al., 2006) which has been a challenge in both projects.

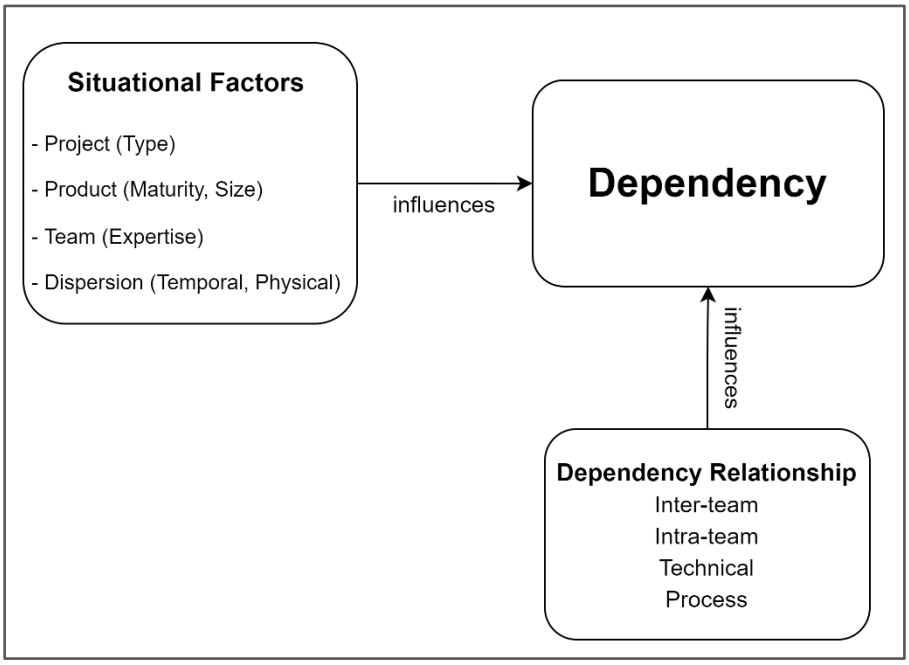

Figure 7.6: DASD Project Dependency and its Antecedents

Overall, the types of dependencies faced in the DASD projects are influenced by different types of dependency relationships and situational factors as illustrated in Figure 7.6.

7.1.3 Dependency Risk Concept

In a software project, the risk is measured by the relationship between the vulnerabilities and threats that exist in the project activities (Dahbur et al., 2011). In software development, while a '*Vulnerability'* refers to a software, hardware, or procedural flaw or weakness that may lead to an issue, a '*Threat'* refers to a potential impact caused by exploiting an existing vulnerability in the system or process. Finally, a '*Risk'* is the potential impact caused by the likelihood of a threat being exposed by a vulnerability. Thus, a dependency is at risk if an associated vulnerability leads to a potential threat that will have an impact on the dependency outcome as shown in figure 7.7. The higher the likelihood of the threat, the higher the potential risk will be.

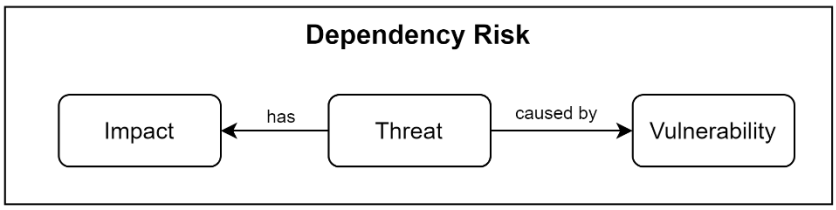

Figure 7.7: Dependency Risk, Threat, Vulnerability, Impact relationship

A dependency issue is a form of threat resulting from a vulnerability that has a high potential to impact the task or project progress. Considering a risk scenario in the Pigeon case, the lack of an appropriate mechanism (*vulnerability*) for early detection and timely notification to address the work-output dependency between Teams R and Team C has the potential to cause a delay in achieving the desired outcome (*threat*). Thus, this work-output dependency is exposed to a dependency risk, if not managed properly, will have a strong impact on the timely completion of the task.

The dependency risk concept emerged while developing the framework of analysis as part of the research design. A dependency is identified as a key dependency if it has several risks associated with it and thus has a high potential to affect the dependency outcome and cause dependency issues. Though all dependency risks were initially categorized under *Delay (e.g.* work completion delay, delay due to misunderstanding, rework, additional communication, technical debts, and poor-quality output - see Figure 3.6, Chapter 3), this was later revised based on a deeper understanding of the dependency issues during data analysis. Thus, dependency risks are classified under two broad categories of *Delay* and *Poor-Quality* as shown in Figure 7.8.

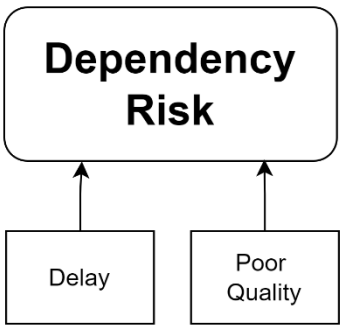

Figure 7.8: Dependency Risk and associated components

The dependency issues identified in both cases are grouped under one of the categories based on their homogenous properties. For example, there is a high potential of task completion delay if the progress of any task is blocked because of the uncertainty around the expected time of completion of a prerequisite or concurrent task. Likewise, if any additional work is required due to a misunderstanding about any task requirements or delay in acquiring key information, it could potentially delay the task completion. The delay can be prolonged and potentially interrupt the release schedule if an excessive amount of additional work is needed.

By contrast, under-delivery of the planned released deliverables, any sort of degrading in the output, or failure in meeting the time commitment is considered as a risk that can impact the quality aspect of the work deliverables. For example, the high knowledge and work-output dependency situation on the Development Manager (DM) highlighted in the Pigeon case has a high potential to impact the quality of both individual and team's work outputs. Besides, the poor relationship between the teams and degraded individual team members' performance is also considered as a risk linked to the quality of individual and inter-team performance. For instance, in the Bluebird case, all the developers' tasks are "*funneling down to a single test resource*" that has the potential to create a testing bottleneck situation that not only impacts the performance of the tester but also the quality of the work outputs done by the team as indicated by the PO, "*So as the quality goes down, what stuff [work] just doesn't get done*" [BPO].

Vulnerabilities causes that emerged from both cases are categorized into three groups: knowledge, work, and process management issues. The knowledge management category includes the issues caused due to discordant identification, communication, information exchange, and shared understanding that has the potential to cause dependency risk. For example, reduced availability and responsiveness of the remote member caused by the time zone difference is a primary cause of knowledge-related issues that could foster incorrect presumptions which in turn may cause delay. Similarly, any work management issues would lead to delay and quality threats that would impact the dependency and its coordination. The overwhelming workload on a single work resource due to lack of clarity in the workload distribution is an example of a work management issue that can result in poor quality work output leading to bad technical debts. And the process management issue category includes all types of causes that can create dependency issues and risks. Misalignment and ineffective work cycles in a different part of the software development process are examples of such issues that have the potential to create delay and quality issues.

Thus dependency risk can be defined as '*a state or property of a dependency, which if ignored, will increase the likelihood of causing dependency issues that could adversely affect the task or project progress'*. The dependency risks associated with the dependency types and their causes are summarised in Table 7.2.

7.1.4 Coordination Strategy Concept

A coordination strategy is a combination of coordination mechanisms that manage dependencies in a situation (D. Strode, 2012) These mechanisms are consciously selected by the project stakeholders based on the dependency needs. Our findings are consistent with Strode's coordination strategy concept. This study found that several mechanisms are used to manage the same dependency in different parts of the software development process, while one mechanism can be used to address the coordination of many dependencies. Moreover, the mechanisms varied depending upon the distribution of the teams. The coordination strategy concept (i.e. the

coordination mechanisms in a situation) comprises three components: coordination activity, tools and artifacts, and structure. The following sections define and discuss each of these components.

Table 7.2: Summary of Dependency risks, and their causes related to dependency types

Dependency Type	Risks	Causes
Knowledge	**Delay:** - Presumptions - Delay in work completion - Delay in project schedule -High risk of Knowledge acquisition bottleneck - Delay in work completion due to unavailability or late response - Delay caused by additional work resulting from incorrect assumptions	**Knowledge Management:** - Reduced availability and responsiveness due to time zone difference - Lack of knowledge about remote expertise - Increased risk of misunderstanding from lack of shared understanding - Language and cultural difference - Most of the task and expertise knowledge is confined to single resource person - Unavailability or late response **Process Management:** Inadequate opportunity for adhoc communication between squads
Activity	**Poor-Quality:** - Poor Quality Work Output may lead to technical debt and rework - Frustrations between teams - High chance of Under-delivery of release deliverables - Poor quality work output from tester and QA - High risk of deteriorated performance of DM due to burnout - Probable risk of detriment to team performance	**Work Management:** - Differentiated/Competing work priorities. - Uncertain work progress. - No mechanism for early identification and notification of dependent work - Exceptionally burdened with responsibilities associated with multiple roles - Lack of clarity in development resource distribution - Overwhelming amount of workload at a time on tester and QA

	Delay: Delay in getting work-output - Interruption in scheduled works and under-delivery of release goals - High risk of work completion delay in testing and QA cycle - high risk of activity bottleneck at tester	**Process Management:** - Misalignment of work style or process. - Ineffective work cycle for development, testing and QA activities **Knowledge Management:** - Additional effort and time required to coordinate with remote member.
Resource	**Delay:** - Delay in work completion - High risk of Single point of failure - delay in critical client issue fixing due to unavailability or low responsiveness - High potential to block work progress of development teams - Potential to interrupt release schedule or under-delivery of release goals - Prolonging client-side interruptions **Poor-Quality:** - Poor quality work output - Potential risk of failure in time commitment	**Knowledge Management:** - Reduced availability and responsiveness. - Lack of shared understanding of the technical dependency - Lack of awareness of possible impacts of code **Process Management:** - Testing becomes a bottleneck. - Firewall and security create barrier. - Solo decision-making and task allocation authority for certain type of work

Coordination Activity

A coordination activity is the primary act involved in managing a dependency and can be defined as '*an action performed individually or in a group to manage a particular dependency*'. Both synchronous and asynchronous coordination activities emerged as a common mechanism that involved both local and global entities at varying frequencies. The activities can be performed in two different modes: *Synchronous* and *Asynchronous*. In synchronous coordination activities, participants are either physically co-located, or a combination of co-located and globally distributed participants. For example, in Bluebird, the Scrum-of-Scrums meeting was a synchronous coordination activity where all the client development teams met in the same place to coordinate inter-team dependencies. Whereas in Pigeon, the 'Tech sync' meeting was a synchronous inter-team coordination activity that included both co-located and distributed team members. In both projects, synchronous activities occurred in different frequencies: per release, per sprint, weekly, daily, and on-demand. For example, in Bluebird, the synchronous 'Release planning' meeting occurred once per release, but the scrum-of-scrums meetings were held weekly to coordinate existing and new dependencies within the current release. The sprint planning session, sprint review, and retrospective meetings are examples of synchronous coordination meetings that occur per sprint, either in the beginning or at the end. The daily standup meetings are regular team-level synchronous coordination activities where team members get "*to check what is going on, what happened yesterday, is there are any blockers*" [PDM]. Most synchronous coordination activities are pre-scheduled to facilitate distributed teams and members' participation. In some cases, these activities occur on an ad-hoc basis, particularly when team members are working remotely or there is an urgent coordination need that requires other teams' participation.

Coordination activities in *Asynchronous mode* are performed without real-time interaction between the members and the communication and information exchange are delayed over a period. This means that the participants need not be physically

present and there would be a minimum delay between the information exchange. Email and messaging via Slack were the most common type of asynchronous coordination activities used in both cases. Asynchronous activities mostly occurred on-demand at any time without any schedule or notification and therefore could happen at any frequency. For instance, Team R members log their activities and work progress of the day by answering three questions in a Slack channel that are used in the next day's standup meeting. "*Answering the questions helps them to check that he [team member] is in the right direction in terms of what have done so far*" [PDM]

A coordination activity can apply a combination of synchronous and asynchronous modes for managing a dependency situation. For example, in Bluebird, while working remotely the Functional Developer (FD) of the O&M squad maintained a constant communication channel by using an open WebEx channel for synchronous communication and a slack channel for asynchronous communication. This hybrid communication mode served as an effective coordination mechanism by reducing the dispersion effects. Figure 7.9 portrays the coordination activity concept along with their frequency modes.

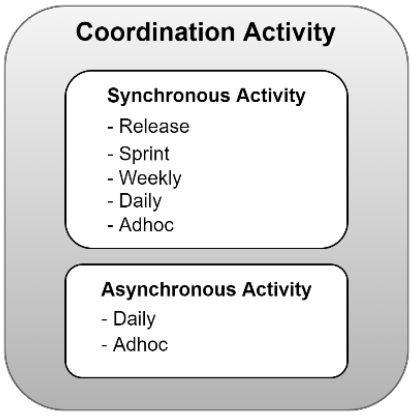

Figure 7.9: Coordination Activity and its modes

Coordination tools and artefacts

While coordination tools facilitate the execution of the coordination activities, coordination artefacts contain information that is used during coordination activities. in both cases, tools such as WebEx and Skype were used to facilitate synchronous

activities and participation of dispersed team members, and Email, Slack to facilitate asynchronous coordination activities. Jira was used to support synchronous knowledge sharing in group meetings and facilitate asynchronous coordination by maintaining records of all project-related tasks, e.g. development, testing, bugs, deployment, and progress-tracking. Jira was also used to facilitate asynchronous communication via descriptions ad comments made in the issue tickets.

Coordination Artefacts contain information which is either produced or consumed by project stakeholders to accomplish their tasks. In both cases, several artefacts were used in either physical or electronic format to store information about the project, its requirements (e.g. epics, user stories, product backlogs), and work outputs (e.g. software code). They were used in task management activities such as tasks breakdown and estimation, and in tasked related decision-making such as task prioritisation, and task assignment. Some artefacts were publicly visible (e.g. dependency board, Squad board) or some were stored in a shared place with restricted permission (e.g. project information, Jira tickets, how-to docs). In the Pigeon case, most artefacts were electronically stored and accessible only by team members. Whereas, in Bluebird, artefacts such as Physical wallboards and squad boards were maintained both physically and electronically. The physical boards were visible for the co-located members radiating information about the project, teams, and tasks that support coordination activities, whereas electronic boards were useful for the distributed members.

In summary, a coordination mechanism includes coordination activities supported by coordination tools and artefacts that manage the dependencies in a situation. Figure 7.10 illustrates the coordination tools and artefacts and their relationship with coordination activity.

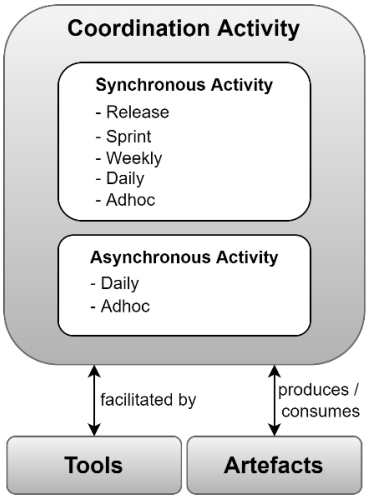

Figure 7.10: Coordination tools and artefacts and their relationship with coordination activities

Structure

Structure represents the arrangement of the teams, and the relations between them - it consists of three primary components: *Availability*, *Proximity* and *Substitutability*. (D. Strode, 2012).

Availability represents the continuous presence and responsiveness of a resource entity for information acquisition, decision-making, or participation in project activities. In global software development contexts, teams are dispersed both spatially and temporally with reduced availability. Both cases encountered several coordination issues due to a lack of availability due to their globally distributed nature. In the Pigeon case, since Team C members were distributed in different US time zones, the availability issues were managed using pre-scheduled weekly synchronisation meetings.

"*We have touchpoints in terms of the Tech Sync Meeting every Thursday, that syncs up with other development teams… opportunity for the team in the US to list up issues they got, then need coordination between a lot of teams*" [PDM]

Proximity refers to the state of the closeness of a resource entity in distance or time. While agile software development methods recommend close proximity between the team members, it is structurally not feasible in a distributed software development

environment. In general, the more physical distance between people, the less they communicate and share knowledge, thus negatively affecting coordination.

There are two dimensions of proximity: spatial and temporal. In distributed contexts, it is not possible to achieve spatial proximity, but temporal proximity (i.e. time-zone proximity - overlap in working hours between time zones) could be achieved by three types of mechanisms: *mechanistic*, *organic*, and *implicit* (Prikladnicki & Carmel, 2013). In a mechanistic mechanism, team activities or meetings are formally organised by adjusting their time zones to ensure the presence of all team members, e.g., pre-organised team meetings. Organic mechanisms rely on unscheduled and unstructured communication between the team members without any prior arrangement, e.g., informal communication over chat messengers or phone conversations. Implicit mechanisms such as shared mental model, transactive memory, shared knowledge about the task, people and their expertise, presence and activities can promote familiarity between the team members. And team familiarity increases as proximity between the members increases, which can improve coordination (Espinosa et al., 2007a). As each of these three mechanisms individually is not sufficient to achieve effective time-zone proximity, it is important to form a strategy by combining these mechanisms (Prikladnicki & Carmel, 2013).

Both cases used several organic mechanisms to achieve temporal proximity such as informal face-to-face and virtual meetings. In the Bluebird case, the scrum-prescribed agile coordination meetings were used to coordinate between the local and distributed team members. Whereas, in Pigeon, team members relied on their cognitive understanding of tasks to identify who had the expertise about that task and used their personal relation to that person while looking for answers to their queries.

"*it's more cognitive, e.g. it's a build issue, I will talk to [Michael]. Tends to be who you know in terms of what is your personal relationship, and how you trust someone. Mahesh might be nice to talk to the person or might just give a quick answer*" [PDM]

Achieving continuous availability and proximity between the distributed and the local teams was perceived as one of the major challenges by participants in both cases. The current body of knowledge suggests that a good level of awareness support could improve the team cognition which will increase coordination effectiveness (Gutwin & Greenberg, 2004; Smite & Dingsøyr, 2012). Gutwin et al. (1996) proposed a framework for awareness support consisting of four types of awareness: workspace, informal, social and group structural.

Workspace awareness includes knowledge about other parties and their interactions within the workspace that helps predict each other's activities and actions that can guide the coordination. There are several elements of workspace awareness, such as identity (i.e. who is responsible for an action), change indication (i.e. past and present changes made in the artefacts), conflict notification (i.e. potential or current conflicts resulting from a change in the artefact), activity (i.e. actions performed in an artefact), location, abilities, and intentions of the team members (Steinmacher et al., 2013). Several of these elements emerged in the case findings. For example, in Pigeon, issues were raised due to lack of awareness of changes made in other parts of the code.

"*Sometimes other teams' bug which causes our staff to break and then we get a lot of bugs, but it's all one cause of other teams' issue. Sometimes you can't deploy your fix*" [PDV1]

Similarly, lack of awareness of other teams' expertise and activities had an impact on the work progress of the development teams,

"…*people globally are checking pieces of stuff in the codebase which was causing the build fail and we didn't know what's part causes that problem... it quite frustrating as we don't really know what's going on outside of here, who's doing what all over the world*" [PDV2]

Informal awareness indicates the knowledge of who is around (i.e. presence) and what they are doing (i.e. work status). It is easy to develop such awareness if the members are co-located, but difficult while they are distributed. In the Bluebird case, several participants identified difficulties in tracking the presence of the remote members and lack of awareness of presence was perceived as the key barrier to effective communication between the members. Although the presence of remote members was tracked by their status in the Slack channel, but this was not always effective because in some cases even if their status was 'available', there was no response as they "*may be not available or in another call*" [BTD2]. On several occasions, the person was busy with some work and did not respond since accepting the call "*may interrupt the flow of work*" [BTD2].

Social awareness represents knowledge about the availability of other members and their preferences that supports the coordination mechanisms. Developing a cognitive mind about availability and personal preferences are the key factors that create better approachability while communicating with distributed members (Steinmacher et al., 2013). The Tester of the O&M squad in the Bluebird case noted that the effectiveness of the communication and willingness of the participation of the remote member depended on "*the person and his/her preferred channel of communication*". As it is not always explicit what is the best way of communicating with a person or what channel should be used, cognitive knowledge about the person helps to develop a shared mental model about the preferred way of communication.

Finally, group-structural awareness indicates the level of understanding of the roles and responsibilities of other people and the activities they are involved in. This awareness was found to support the coordination of inter-team dependencies in both cases. While the development teams in Pigeon faced coordination issues due to a lack of knowledge of other teams, their expertise, roles and responsibilities in the project, there was no such issue observed in the Bluebird case. Several mechanisms could be used to develop group-structural awareness, such as finding out experts and their

locations from historical data and contextual information, and making teams aware of roles assigned to other teams or developers in the project.

"*Sometimes you don't know who to call or you don't know who you are talking to. I don't know how these guys know that I had done this. Maybe they looked in TFS [Code version tool]*" [PDV2]

Substitutability is another dimension of the structural element which refers to a situation when a team member has the expertise to perform the tasks (e.g. development, testing) of another. Substitutability can reduce the impacts of delay and provide support in maintaining project schedules. In the Pigeon case, substitutability worked well in development activities but was not very effective in providing knowledge and decision-making alternatives.

"*It depends on workload, if it's heavy then [DM] assign to another developer but if someone is willing to take or willing to learn a different thing, he will take that as well… we previously had one tester and he left it and we did not have any tester at that time. turn by turn we did some testing. He is the only one and so much bug to do for him. So we do help him" [PDV3]*

Therefore, as discussed in Chapter 4, there is a high risk of knowledge and decision-making bottleneck due to not having any substitute for the team's Development manager. The development manager has skills and authority that have little crossover with other team members' expertise. Though the senior developer has the same expertise as the DM and knows most of the product features, he lacks decision-making capability. Such a person could be a potential substitute for the DM and his decision-making capabilities could be improved by introducing pair-programming and joint decision-making sessions with the DM.

"…*whenever the boss is way, I take over the standup meetings, things like that and just make sure they've got stuff to do. I think the biggest blocker I see anyway is a lack of understanding not knowing what to do, that's the biggest blocker in my opinion*" [PDV2]

In the Bluebird case, Substitutability was achieved by developers participating in the testing activities and thus sharing the workload of a single tester to reduce the risk of the testing bottleneck. In agile software development methodology, substitutability promotes flexibility in work management and consensus decision-making that fulfils the cross-functionality and self-organising team behavior (Hollenbeck et al., 2012) which can be considered as part of coordination strategy. In the Pigeon case, development teams had a positive mindset of taking over the tasks of another team even if that was beyond their expertise which is a core agile culture and effective strategy for inter-team dependency coordination (Marthe Berntzen et al., 2021).

"*Each team has an area of expertise, but sometimes when another project comes up it might be who's got bandwidth to do it. So sometimes we take on things outside of our area of expertise*" [PDM]

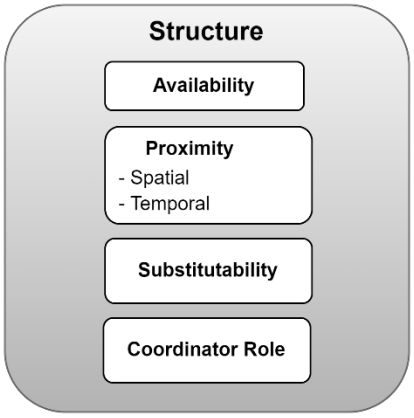

Figure 7.11: Structure mechanism and its components

Similar to the concept of the "*Coordinator Role*" designated for boundary-spanning activities (D. Strode, 2012), there were several roles defined by both case organisations that served as coordination mechanisms while managing inter-and intra-team dependency relationships. As discussed in Chapter 6, several coordinator roles emerged from both case findings that were effective to manage the inter-and intra-team dependencies. For example, the Scrum Master (SM), Development Manager (DM) and Product Manager (PM) were the key coordinator roles in Pigeon. Whereas, in Bluebird, the vendor liaising Project Manager (PM) along with SM and PO were the roles acted

as coordinator for effectively coordinating the distributed teams and members-related dependencies. Figure 7.11 illustrates the structure mechanism including the four components associated with this mechanism.

In summary, several types of coordination mechanisms were identified in both cases depending on the dependency relationship, type, distribution and organisation of the teams. For managing inter-team dependencies, synchronised group coordination meetings (face-to-face or virtual) were the key activity found in both cases. These meetings were supported by several shared artefacts such as dependency board, squad board, and tools such as Jira, and WebEx. Several key roles such as Scrum Master, Product Manager, Product Owner were identified. Finally, several organisation structures such as proximity, continuous availability and substitutability play an important role in coordinating the dependency relationships between co-located and distributed teams.

Three antecedent factors were found to influence the selection of the mechanisms forming the strategy, dispersion, dependency relationship and process. The physical and temporal dispersion of the teams and members influenced to either apply different mechanisms or improvise the existing ones to facilitate distributed coordination as analysed in both cases. For example, in Pigeon, to adjust the time-zone differences, Team R had to adjust their working hours, either by coming early or staying late, depending on the location of the remote member.

"*Time-Zone is a challenge, [when] we are going to talk with the US person, we need to come early, [but] if we need to talk with [the person in] India, then we have to stay late*" [PTS]

In the Bluebird case, the remote member always got priority to share their status or opinions during the team meetings to minimise the impacts of interruption due to poor connection. Besides, in both cases, the group-mode coordination activities that included distributed participants mostly relied on technology-mediated communication using voice or video calls, screen sharing, slack and electronic artefacts (e.g. Jira,

Confluence) which was different from the co-located participants' meetings. For example, in Bluebird, when all the members were present in the NZ location, regular coordination meetings such as daily standup, and sprint planning meetings were conducted face-to-face. Whereas the same meetings were conducted using video calls when two of the members were working from remote locations. Therefore, the selection of the coordination mechanisms depended on the distribution of the members.

The selection of the coordination mechanism is also influenced by the dependency relationship. For example, the coordination mechanisms used for inter-team coordination differs from the mechanism of intra-team coordination. The pre-scheduled inter-team meetings were the primary mechanisms used in both cases for inter-team coordination. In the Pigeon case, these meetings were called as 'Tech sync' meetings where distributed teams coordinated their dependencies. Similarly, in Bluebird, the inter-team Scrum-of-Scrums meetings were used to synchronise and coordinate their inter-dependent activities. In contrast, agile prescribed meetings, such as sprint planning meetings, daily standups, code reviews, sprint reviews and retrospective meetings, were the primary mechanisms used in both cases for managing intra-team dependencies. Thus, the dependency relationships influenced the formation of coordination strategies in DASD projects.

Both case participants reported issues with the overwhelming workload for the solo tester in the team leading to testing bottlenecks and performance hazards. Inappropriate and inefficient management of the testing activities in the agile software development process was the primary source analysed for this issue. For example, while in Pigeon, the development and testing cycle was managed in separate sprints, the testing activities were only performed in the last 2 days of the sprint in the Bluebird case. Both these approaches were prone to delay and poor-quality outputs which was contrasting the motivation of producing shippable products from each sprint cycle. Therefore, it is evident that any inconsistencies in the software development process,

particularly the methodology e.g. waterfall, agile or hybrid, would have an influence on the coordination mechanisms.

Previous literature findings also confirm that proper distribution and implementation of testing activities throughout the sprint iteration is one of the key challenges in adopting agile methods (Talby et al., 2006; Wohlin & Petersen, 2010) which is apparent in this study. Following a Test Driven Development (TDD) can be a probable solution to effectively manage the testing bottlenecks and coordination overhead in a distributed agile software development context (Morken, 2014b; Sangwan & LaPlante, 2006). Figure 7.12 presents the components of coordination strategy and its antecedents in managing the dependencies in DASD projects.

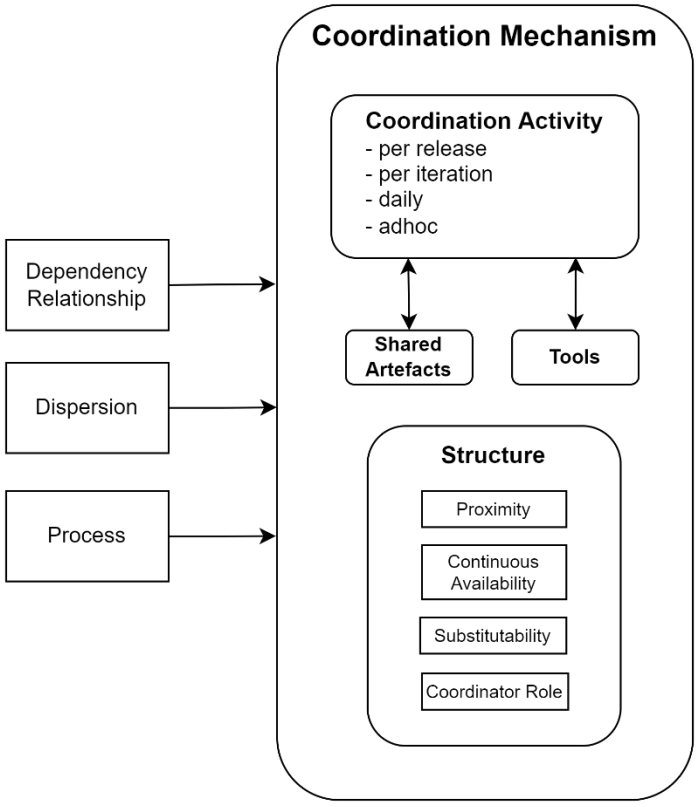

Figure 7.12: Coordination strategy components and its antecedents

7.1.5 Coordination Effectiveness Concept

A preliminary model of Coordination Effectiveness (CE) is presented in Chapter 2. Strode introduced the concept of CE in the context of co-located agile software development (D. E. Strode et al., 2011). According to this concept, coordination

effectiveness is achieved by a combination of explicit and implicit components. There are three explicit components: *right thing*, *right place* and *right time*, and five implicit components: *know why*, *know what is going on and when*, *know what to do and when*, *know who is doing what* and *know who knows what*. According to Espinosa et al. (2002) who proposed a similar concept in the context of a globally distributed software development, coordination outcome can be measured by the '*state of coordination*' which is defined by the degree to which the dependencies are effectively managed. If the dependencies are well-managed, the state of coordination is 'highly effective' and has a positive influence on the team performance. To achieve a highly effective state of coordination, an effective coordination strategy is required made up of a combination of explicit and implicit mechanisms considering the task, time, and distance factors.

The findings from both cases conform to both the concepts of CE which indicates that an effective coordination strategy would positively reduce the possibility of dependency issues (e.g. form of delay or quality issues) which would positively affect the coordination outcome. For example, if a resource person is physically or virtually available to respond to developers' queries or take decisions, it will reduce the chances of the presumption that could cause delays. It means that if the *right person* is present at the *right place* at the *right time*, the coordinating strategy would be effective to reduce the possibility of delays in decision-making or information acquisition.

Likewise, the coordination strategy involves a mix of mechanisms to effectively coordinate the dependencies. Some of these mechanisms could be explicit (e.g., pre-schedule meetings) and some are implicit (e.g., know who is available) and there is a correlation between these mechanisms to effectively manage dependencies. For example, while coordinating dependencies between local and distributed development teams, a recurrent group meeting was found to be an effective explicit coordination mechanism where all the team's representatives were present to discuss their dependencies. These regular meetings facilitated a shared understanding of '*what is going on*' in terms of release goals and '*who is doing what*' to achieve the release

goals. This cognitive (i.e., implicit) understanding helped in identifying possible work dependencies with other teams or '*know what to do*' to resolve the same issue faced by other teams.

"*We have touchpoints in terms of the 'Tech Sync Meeting' on every Thursday, that syncs up with other development teams. Issues that we are having, to alert people about critical client issues… Not only concerns about the issues, but also for any new builds, any new changes to share, and then email all the developers to know what is happening. Heads up in things, e.g. investigating one issue that might come to the other teams as well*" [PDM]

The regular inter-team synchronisation meeting above is an example of explicit mechanism that fosters the development of cognitive understanding (e.g. '*who is doing what*', '*what is going on*', '*how to solve*' an issue) which is helpful to coordinate current and future inter-team dependencies. Without this knowledge, teams may face delays in identifying which team to contact for an issue, or which team's work may cause the bug, and hence, would increase the dependency risk. Therefore, a coordination effectiveness model should consider both explicit and implicit components of the coordination strategy.

Explicit coordination mechanisms involve overt activities that include direct interaction and information exchange between multiple parties for managing interdependencies. Examples of such mechanisms are task distribution, pre-arranged meetings, information, and instruction exchange via written documents or communications (Espinosa, Lerch, et al., 2002; Faraj & Sproull, 2000; Kanaparan & Strode, 2021). Several explicit coordination activities, tools and artefacts emerged from both cases: while agile coordination meetings (e.g. sprint planning, daily standup, sprint review, code review) helped in managing the team's internal dependencies, pre-scheduled meetings helped in managing inter-team dependencies. In the Pigeon case, tech sync meetings, support meetings, and QA meetings were used to manage dependencies between distributed development, support, and QA teams respectively. Conversely, the

Scrum-of-scrums was the primary explicit coordination activity used to coordinate dependencies between the collocated teams. Several tools and artefacts were also used as explicit mechanisms: for example, Email, Jira, WebEx, Slack, and Confluence were the common coordination tools used in both case projects. In the Bluebird case, a number of physical posters and boards, e.g. dependency board, SoS board, Definition of Done and Definition of Ready posters were used to coordinate between the teams. In the Pigeon case, most shared artefacts were electronically stored and accessed by members of the teams.

To be effective, these mechanisms must ensure that the required resource or information is present and accessible when it is needed and in a form that can be used or understood by the involved parties. If the information or resource is either not available or not in the form that is readily consumable, then it will not be effective to manage the dependencies. For example, while sharing the difficulties in coordination with a distributed product owner, one of the scrum masters of Pigeon says,

"*To be specific, she's not technical like me. So I put her in discussion and then she says, 'I don't know, I need to ask someone to come back to you'. And then they go, she doesn't know very much, it's just a waste of time, right? So I had to structure it so that the person she needs to go to was in the same meeting, which means that we got an answer straight away*" [PSM1]

In the above scenario, initially, the resource person was not available in the meeting which created delays in receiving responses and thus both sides were frustrated. This mechanism is complex and requires a back-and-forth dialogue until the required information is acquired. But as soon as the resource person is included in the conversation, this complexity is resolved and information exchange becomes faster and easier. It illustrates that having the *right thing*, at the *right place*, at the *right time* is important for effective coordination; therefore, these three explicit components are included in the model of coordination effectiveness as shown in Figure 7.13.

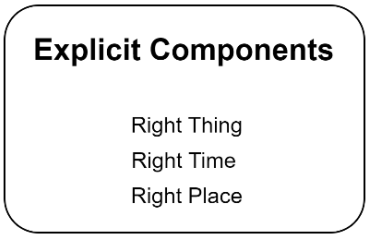

Figure 7.13: Explicit Components of Coordination Effectiveness

Though explicit mechanisms are supportive for effective coordination, it has limited bandwidth and is prone to availability (Nasroullahi & Tumer, 2012), particularly when teams and members are distributed globally. In contrast, implicit mechanisms involve actions based on the cognitive understanding of the team members about the task and the team that helps them to manage their interdependencies without any overt instructions or communication (Espinosa, Lerch, et al., 2002). While interacting with other members and participating in different software development activities, team members develop a cognitive understanding of the task and the team which helps them implicitly to coordinate their dependencies. Since the teams and members are not co-located for informal face-to-face communication, implicit mechanisms are crucial in distributed software development.

Similarly, several implicit components of coordination effectiveness associated with the dependency risks were identified in both cases. They are:

- Know how interdependent
- Know who is available
- Know who knows what and where
- Know what is going on and when
- Know what to do and when

The '*know how interdependent*' component indicates that a shared understanding of the task-in-action and its interdependencies on other tasks contributes to the timely coordination of related dependencies. As identified in the Pigeon case project, development teams encountered major challenges in coordinating their work dependencies. Since there was a lack of appropriate mechanisms to identify

dependencies early, teams did not know how their work was interdependent on other teams' work. As a result, it led to delays which was impacting their work quality and relationships.

"*you're trying to do what are you trying to do some feature that you haven't talked to us about? It's due in three weeks, and [now] you need our help? And also now it's not gonna be delivered because of us? Well, that's not really fair, because you didn't tell us about it upfront*" [PSM1]

The above example illustrates that there is a lack of knowledge about how one team's work is dependent on another team's work which had an impact on their work coordination. Therefore, a lack of shared understanding about the inter-dependencies involved in the task-in-action is an important cause of delays. Similarly, the lack of mechanisms for early identification and notification of task dependencies leads to potential quality risks. It suggests that having a shared understanding of task and related dependencies is needed for effective coordination of inter-team task dependencies.

The '*know who is available*' component indicates the awareness about the availability of a resource entity. This implicit awareness is critical for communication and coordination with remote team members for information and decision exchange as stated by the BA in Bluebird case,

"*It's important to know if he [the remote person] is available or not, e.g., try calling but* ***may not be available*** *or may be in another call or may interrupt flow of work*" [BBA]

This evidence indicated that the lack of availability or presence awareness is barrier in distributed communication. As discussed in the literature section, this aspect of implicit coordination is important for coordination with distributed team members. While team members are co-located, it is easy to find out whether the person is available or not, and thus can just turn around and start communicating, "*I'll just turn around and say, hey, do you know anything about this?*" [PDV1].

Another instance of this awareness includes knowledge about the individual's preference which has an impact on the effectiveness of the coordination. While communicating with a remote person, knowing *what they are working on*, *how they work*, *how they prefer others to work* and *what's their preferred way of communication* supports coordination. For example:

"*Tends to be who you know in terms of what is your personal relationship, and how you trust someone. [Michael] might be* ***nice to talk to the person*** *or* ***might just give a quick answer***." [PDM]

Based on this evidence, '*know who is available*' was included as an implicit component of coordination effectiveness.

The *know who knows what and where* component indicates the awareness about the expertise available both inside and outside the team and their locations. It is evident from the data that team members must know about the skills and expertise of the team members, so that they can seek advice, on any issues or clarification and feedback on their work.

"*But sometimes I am not familiar and sometimes I know what the issue is, but I don't know how to solve it…* ***to make it quick****, you know sometimes he [Development Manager]* ***might say that doesn't matter or you can shortcut***" [PDV1]

The above example from the Pigeon case indicates that the team member knows that the development manager has the necessary knowledge and expertise to guide him to investigate or solve any issue which might take longer otherwise.

Similarly, knowing which developer to contact when testers are facing issues helps in resolving the problem faster, "*He [tester] will come to ask if he's got any problem, or how to run this*" [PDV2]. As mentioned by the tester in Pigeon, Team's 'Agile board' records information on which developer worked on a particular task and has the required knowledge to help the testers.

While it is comparatively easier to develop such awareness in co-located teams, but can be challenging to know about distributed teams, and their expertise, hence may face difficulties to identify their location.

"*Sometimes you don't know who to call or you don't know who you are taking to. I don't know how these guys know that I had done this. May be they looked in TFS*" [PDV2]

From the developer's perspective, it is difficult to know whom to contact when they are globally distributed. While shared artefacts such as code repositories may be helpful in identifying the required expertise, in Pigeon, most members were highly dependent on the DM who identified the distributed knowledge source and their location.

"*In terms of who to go for information, your first contact would be Dev. Manager and he would probably point you*" [PDV4]

Though the team members initially did not have this knowledge and were dependent on the DM, however, they eventually developed a cognitive understanding that provided support for future coordination. Therefore, knowing who knows what and where was included as a component of coordination effectiveness.

The *know what is going on and when* component reflects the shared understanding about the activities of other members within the team, as well as, those performed by other teams. Such shared understanding helps in identifying the dependencies earlier, i.e. results in proactive coordination rather than reactive coordination. In the Pigeon case, Teams R and C were struggling to resolve some of their uncertain dependencies. Team C was lacking a shared understanding of the task interdependencies and was not interested in participating early until being blocked which resulted in delays.

"…s*o if* ***they're changing there that may affect us****, so we can contact them proactively. And ask if there's a problem, right? Ask if you're going, can you think about this? Before we didn't know what they [Team C] were doing until they have been blocked by us. it takes a while to figure out the problem to then introduce effects*" [PSM1]

In the Bluebird case, the development teams were conducting a release planning session at the beginning of each release and recording the inter-team dependencies in the dependency board. Any further dependencies were further added and coordinated during weekly scrum-of-scrums sessions. During these sessions, teams developed an overall understanding about *what is going on* in terms of the project, what other teams are doing, how they may affect us. Based on this understanding, interdependent teams started coordinating early to avoid blocking each other's work progress. Therefore, knowing *what is going and when* was found to be an important component of coordination effectiveness for effective coordination between local and distributed teams.

The *know what to do and when* component indicates that every individual involved in the project has a shared understanding of their tasks and the timing of their execution in relation to all other tasks in the current work cycle. Lack of this understanding leads to misaligned work priorities and uncertainty in work completion which has the potential impact of causing delays and poor quality. The following examples from (include case name) illustrate how the team members developed a shared understanding of their tasks (i.e. what to do) and priorities through agile practices.

"*It is very easy for the developer to be head-down the blink is on, you can't see anything else when you are very focused on what you are doing before you reach to the end of the sprint and find that it needs to be re-engineered. For me it's [daily standup] a way to check what is going on, what happened yesterday, is there are any blockers*" [PDM]

"*The backlog grooming happens with the product owner, and at that point we kinda looking it what are the priorities, and we are making sure whenever we are planning on or working on is the priority*" [PDM]

In agile methodology, the project goals are executed in Sprints where each team decides what they will do (i.e. sprint goal) in relation to overall project or release goals; thus ideally knows what to do and when as a team. Similarly, the teamwise sprint goals

are broken into small tasks that are estimated and self-assigned by the team members during sprint planning sessions and further synchronised during backlog grooming, daily standup sessions. The scrum-of-scrums session used in Bluebird is an effective way to know what each team needs to do to collectively achieve the release deliverables. Therefore, it is reasonable that knowing what to do and when is supportive for effective coordination and therefore was included as a component of coordination effectiveness.

Additionally, two implicit components '*know why*' and '*know who is doing what*' of the coordination effectiveness concept first introduced by Strode et al. in co-located agile software development, were found important in the context of DASD coordination.

The *know why* component indicates a common understanding about the collective goal of the project and how a particular task contributes to the achievement of that goal. In the Pigeon case, several participants highlighted that understanding the background of the task, how it contributes to the feature under development is important for both successfully planning and developing the features.

"S*ort of information I like to give is the background of the problem. then* ***explains the problem and why we doing it*** *and overview of the solution. It's done during the planning poker. Stories not really practical until it has these things*." [PDM]

Lack of correct understanding or incomplete understanding resulted in issues that would require additional time and effort to complete their work.

"*He might come and say, it doesn't work. Because* ***he might not have understood the reason of doing that****, or run something wrong, or hasn't apply to some data properly*" [PDV2]

In the Bluebird case, this shared understanding was developed while creating the 'Definition of Done' and 'Definition of Ready' through team consensus. These artefacts were posted around the team's workspace to make it continuously visible for self-

checking, which implicitly helped to avoid misunderstanding; hence required fewer coordination efforts.

The *know who is doing what* component indicates a shared understanding of the activities of the other individuals in the project, either as a group or individually. This concept emerged in Pigeon. Members of Team R encountered conflicts with Team C due to not knowing what the other team was doing which had a significant impact on their work progress.

"*We don't know what they're doing, and they don't know what we're doing. But it's listening to what the activity that they're trying to perform to figure out how, where it is that we come in"* [PSM1]

Therefore, *knowing who is doing what* was found to be an important component of coordination effectiveness. All these implicit components contributing to the CE in the DASD project are presented in Figure 7.14.

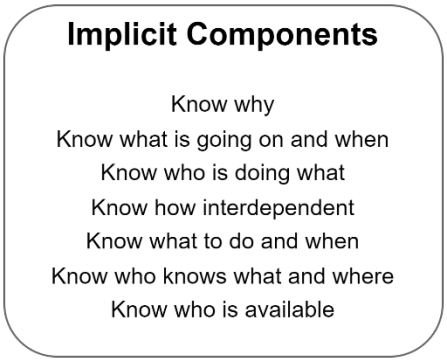

Figure 7.14: Implicit components of Coordination Effectiveness

Overall, based on the findings discussed above, coordination effectiveness in distributed agile software development is conceptualized as,

"*a state of coordination in which all the right things are available and fit for purpose in the right place, at the right time. And all the local and distributed teams and their members have a shared understanding of the project goals and associated tasks, their priorities, know what is going on and when, who is doing what and how their works fit*

with other team's work, know what to do and when, who knows what and where, who is available in different time zones".

The coordination effectiveness concept illustrating the explicit and implicit components are presented in Figure 7.15.

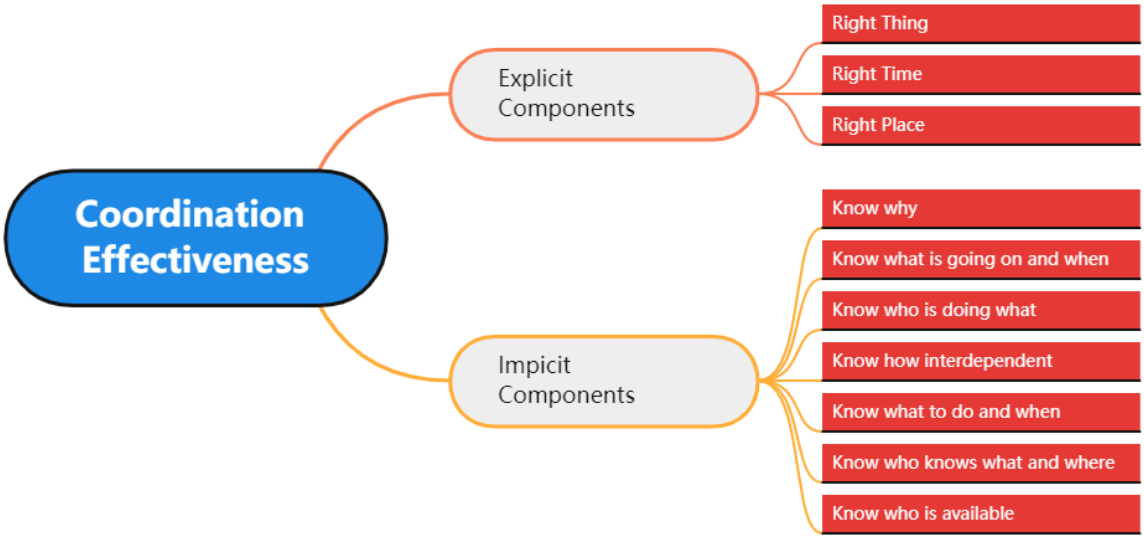

Figure 7.15: Coordination Effectiveness concept and its components

7.2 Boundaries, Associations and States of the concepts

While developing a conceptual model, it is vital to define the boundaries within which the concepts, their states and associations are valid. As this study focused on understanding the coordination in Distributed Agile software development projects, the boundary of the proposed theoretical model is the same as the study's context, i.e. DASD projects based on their merit in the current state of knowledge. The data collection and analysis methods were designed, and the case projects and participants were selected to meet these boundary conditions. However, the final boundaries to which the model applies were specifically determined by the researcher at the data analysis phase; therefore, all the concepts proposed in this model should meet the following boundary conditions:

1. The software development project under investigation has adopted either an agile methodology (e.g. Scrum, XP, Kanban), or a hybrid approach by combining practices from different agile-based frameworks (e.g. Scrumban,

ScrumXP), or a combination of contemporary agile and traditional plan-driven development methodologies.

2. There are several distributed parties involved in the project either distributed locally with few hours or no time difference or globally distributed with a substantial amount of time difference. There are several dependencies between the distributed parties for the successful completion of the project deliverables.
3. The customers are external to the project teams and there should be at least one or more customer proxy (e.g. PO) present as part of the project members.
4. The project should be a software development project either in the development phase of a new product with a minimal customer impact or in the feature enhancement phase of a mature product with minimal to severe customer impact.

The proposed model has a total of four primary and two auxiliary associations between the six constructs of this model (as shown in Figure 7.16). The associations outline the relationships between the proposed concepts were derived from the analysis of the two case findings. The proposed associations between the constructs are:

Primary Associations:

1. The dependency types are associated with Dependency risks. A dependency is termed a key dependency when it has the potential to lead to a dependency risk comprising delay and poor quality.
2. The dependency types are associated with coordination strategy. Different types of coordination strategies, consisting of coordination activities, tools and artefacts and structure mechanisms, are used to manage different types of dependencies.
3. Coordination strategy is associated with coordination effectiveness and a well-formed coordination strategy leads to a high level of coordination effectiveness conforming to both explicit and implicit components.

4. Dependency risk is associated with Coordination effectiveness, and a dependency risk indicates a lack of engagement of the components (explicit or implicit) that impact coordination effectiveness.

Auxiliary Associations:

1. The type of dependency is associated with the dependency relationship. Dependency relationships can be inter-team, intra-team, technical and process specific that may influence the different dependency types.
2. The type of dependency is associated with situational factors: project, product, team, and dispersion factors.

Representing all the possible states of the concepts and their components in different conditions is vital for any model development. This study has identified three different states of the coordination strategy and their effectiveness depending on the distribution of the teams and their members.

1. When teams are distributed:
    a. an effective team coordination strategy relies more on group-structural component for their Adhoc coordination, and coordinator role for all the boundary-spanning activities awareness to reduce impacts of low availability and proximity.
    b. achieving substitutability is difficult, and in most cases impossible, since each team has their own specialisation, and the division of labour restricts other teams to develop skills on features assigned to other teams.
2. When team members are distributed, the effective coordination strategy relies more on informal and social awareness among the team members for Adhoc coordination activities.
3. In both scenarios, coordination activities heavily rely on the tools and shared artefacts.

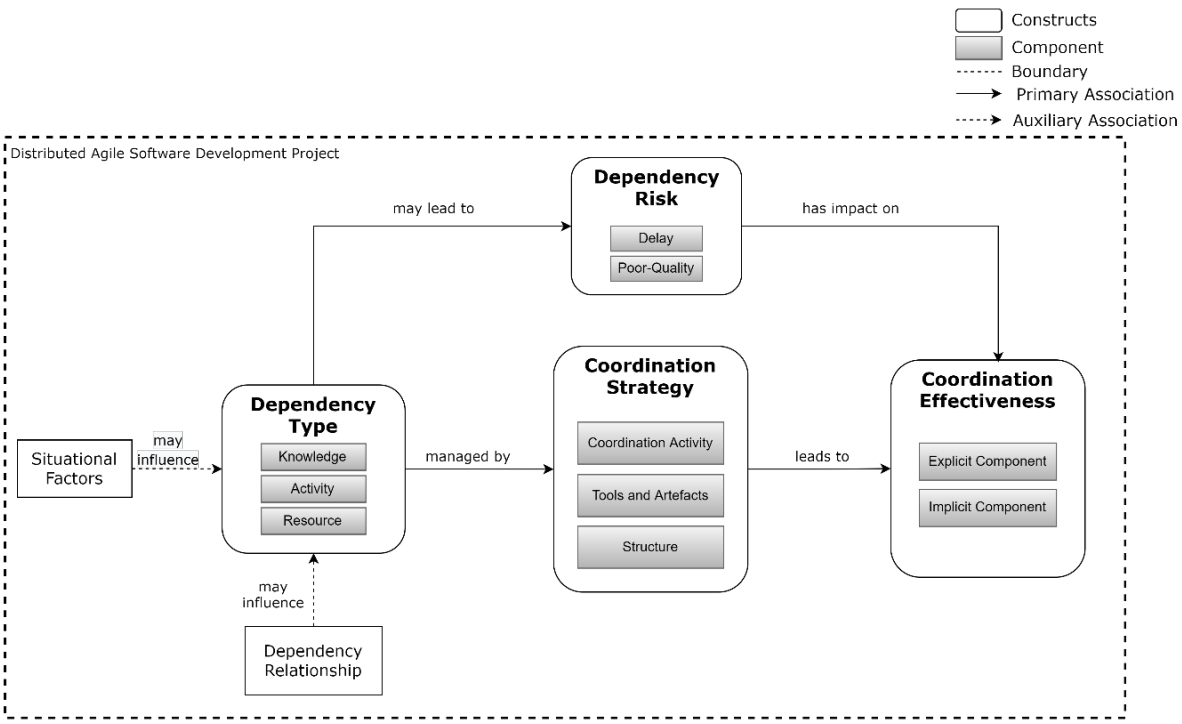

Figure 7.16: Proposed model of Coordination for Distributed Agile Software Development Projects

The proposed model of coordination is portrayed in figure 7.16 illustrating each of the parts with their components. However, a detailed view of the model is provided in the Appendix C. The primary and auxiliary types of associations among the concepts are respectively drawn using solid and dotted lines. The dotted box surrounding the model indicates the boundary within which these concepts are applicable.

7.2.1 Summary of the model

The process of coordination begins with the identification of the dependencies. The model describes that there could be three different types of dependencies identified in DASD projects. While identifying the dependencies, this study suggests answering three primitive questions: who is dependent, for what and why. The answers to these questions would help the practitioners characterise the dependency relationship, the dependency type, and the purpose of the dependency (i.e. the reason for the dependency), respectively. The model proposes that any task currently being worked on (i.e. task-in-action) is dependent on one or more of (1) shared information from a shared artefact or knowledge resource, (2) the output from a previous or concurrent task from another work resource, (3) a decision from an appropriate resource with

authority or a technical artefact needed being available and usable. In terms of dependency relationship, the model proposes that the dependencies would be between multiple teams or members of the same team, or the technical components, or process activities.

Once the dependency and its relationship are identified, it is important to analyse whether the dependency involves any dependency risk. A dependency risk would involve one or more vulnerabilities associated with any dependency that could potentially lead to a threat that could impact either by creating delay or poor-quality outcome. This study suggests that the dependencies and their associated risks need to be identified early and managed by a coordination strategy that would be effective enough to avoid or mitigate those risks.

A coordination strategy consists of mechanisms involving various coordination activities, tools and artefacts and structure mechanisms. In addition, the desirable characteristics of the output of the coordination strategy are posited as (1) fit for purpose –this includes meeting quality standards and being accurate; (2) there exists a shared understanding of the relevance and impact of the coordination output on the task-in-action; (3) promotes awareness about the parties involved, their roles and responsibilities, availability and preferences, and their interactions in the workspace; and (4) the output is received and understood either before the task-in-action is started (pre-task and the need is anticipated), or with minimal delay after the dependency and coordination, need is communicated to the appropriate resource (in-task, either anticipated or ad hoc and unanticipated).

Finally, the model proposes that a state of coordination (i.e. coordination effectiveness) could be achieved when the coordination strategy meets the above desirable characteristics which eventually helps in alleviating the dependency risks.

7.3 Prescriptions for practical implementation

'*In theory, theory and practice are the same. In practice, they are not*.' - Benjamin Brewster (1882)

A primary characteristic of a good theoretical model is that it can be implemented in practice (Lewin, 1951) and the 'goodness' of a model depends on its usefulness for practice. Therefore, this section will focus on the practical implementation of the proposed model in distributed agile software development projects. One of the significant characteristics of the proposed DASD coordination model is that it has the potential to support practitioners in understanding the dependency risks that might affect effective coordination and apply appropriate coordination strategies to mitigate the dependency risks. A list of mechanisms that practitioners can adopt to mitigate potential dependency issues is presented in this section. However, the efficiency of this model and the prescribed mechanisms is yet to be tested by the practitioners.

Fourteen types of dependency issues drawn from the findings of the two cases are categorized into three high-level categories: *knowledge management*, *work management* and *process management*. A repertoire of coordination mechanisms to resolve these types of issues are discussed in the following sections.

7.3.1 Practices for resolving Knowledge Management issues

All the dependency issues that are linked to knowledge acquisition, distribution and shared understanding are grouped under this category. The main mechanisms for addressing these dependency issues are the following:

- A constant open communication channel, by setting up a continuing conference call with a shared screen and slack channel, creates '*visual cues*' that increase awareness of the presence and availability of the remote members. Additionally, an adjustment to working hours in all the locations would increase the overlapped working hours, which would help reduce time zone impacts on remote members' availability and communication frequencies.

- For collocated teams, open workspace, adjacent seating arrangement for highly interdependent teams, and participating in social activities (e.g. BBQ parties, year-end parties) improve inter-team relationships and communication.

- For distributed teams, conducting regular synchronization team meetings and maintaining virtual social interactions (e.g. sharing photos of distributed team members, informal discussion sessions) are effective mechanisms to avoid delays in getting responses and feedback.

- Conducting stakeholder analysis per release and sprint helps in the early detection of teams' collaboration needs. Automated dependency analysis techniques (e.g. logical coupling analysis) and visualization tools (e.g. Dependency Map for Jira) can be used to identify and track inter-team task dependencies.

- A combination of traditional and agile approaches is effective for inter-team coordination. Highly interdependent teams should align work styles and perform joint planning and review activities to improve dependency awareness between them. Inter-team continuous coordination techniques, such as Scrum-of-Scrums meetings, Communities of practices (CoPs), open fishbowl sessions, are supportive for collective shared understanding about other teams' activities, their expertise and effectively manage their interdependencies.

- Agile practices are effective for creating redundancy of knowledge resources that could potentially eliminate knowledge acquisition bottlenecks. Joint problem solving and knowledge sharing sessions e.g. pair programming, mob programming, code review, and group knowledge sharing and discussion sessions e.g. sprint planning sessions, daily standup, Q&A sessions create opportunities for spontaneous knowledge sharing in the team.

- Pairing asynchronous communication channels (e.g. sending emails prior to the meeting sharing a list of items to be discussed) with synchronous

communication (e.g. scrum-of-scrums meetings, support team meetings) can be effective to reduce opportunities of misunderstandings due to language and cultural gaps in distributed teams' coordination.

Overall, a combination of traditional and agile mechanisms is effective to reduce knowledge management issues. Additionally, the effectiveness of these mechanisms depends on the successful integration of high-end technologies and shared artefacts for improved knowledge distribution and awareness support.

7.3.2 Practices for resolving Work Management issues

The following mechanisms and practices are recommended to mitigate the four types of work management issues identified in both cases:

- Test-Driven-Development (TDD) is the recommended method to create a balance between development and testing by shifting the mindset from delivering features to delivering high-quality features. Adopting a TDD approach in both the team and organisation level will encourage the developers to participate in testing that would reduce the potential risk of a testing bottleneck due to solo testers.
- In case of mature product development, feature enhancement work streams should be separated from the bug fixing and support stream for better workload management and resource distribution. Additionally, using teams could apply different kinds of various metrics to perform an analysis of incoming workload, throughput and performance factors will improve Quality of Service (QoS).
- Adopting the TDD methodology could radically reduce the bug detection and resolution time which will ensure the QoS; therefore, it is the recommended methodology for mature product development projects.
- A combination of formal and informal mechanisms that promotes dependency awareness should be established both at the team (e.g. stakeholder analysis, triage sessions, logical coupling analysis) and project level (e.g. joint planning,

review and retrospective sessions, scrum-of-scrums) to ensure early dependency detection.

- Visual dependency boards serve as good information radiators for tracking inter-team dependent works progress. For co-located teams, a physical dependency board conveniently placed in the open workspace and conducting regular team synchronisation meetings in front of that board is an effective mechanism for task coordination. A virtual dependency board aided by a project management tool (e.g. Jira, Active Collab) can be effective in coordinating globally distributed development teams.

7.3.3 Practices for resolving Process Management Issues

The following list of mechanisms is recommended to mitigate the three types of process management issues that could potentially hinder the coordination effectiveness:

- Agile values and practices endorse a self-organising, cross-functional team notion where all the work-related decisions and task allocations are made by the team instead of a single authority. Agile coordination practices facilitate opportunities for frequent communication that can be effective in the teams' autonomous decision-making.
- Adopting TDD methods creates a balance between the development and testing phases that can mitigate ineffective work cycles. TDD-based work cycle helps to quickly pinpoint code issues and resolve them. It ensures effective work cycles in both mature and non-mature product development.
- Aligning the work cycles between highly interdependent teams will facilitate improved dependency awareness, which in turn will help in resolving the inter-team work priority conflicts. Likewise, increased opportunities for Adhoc communication using regular inter-team synchronisation meetings with shared artefacts, and high proximity seating arrangements for co-located teams are examples of practices that would improve shared work prioritisation.

Overall, adopting an agile mindset and implementing agile recommended practices will resolve many issues related to process management. Promoting shared leadership in the team will facilitate collective decision-making that will potentially resolve issues related to the hierarchical decision-making structure. Organisations must be cautious to adopt mechanisms that create a balance between formality that supports teamwork alignment and agility to promote self-organisation (Berntzen & Wong, 2021).

7.4 Summary

In summary, this chapter presented the essential contributions of this thesis by proposing a theoretical model of coordination. The model and its elements are based on the evidence from the analysis of two cases studied in this research. We have further proposed recommendations for the practitioners by highlighting practices that could guide them to achieve coordination effectiveness.

Chapter 8 : Discussion

This chapter answers the research questions presented in Chapter 1, followed by a discussion of the study's findings as they relate to the extant literature. Next, the quality of the case study research methodology is evaluated against its adherence to meeting the criteria of validity, reliability, rigor and relevance using the widely used frameworks of Dubé & Paré (2003) and Fossey et al. (2002).

8.1 Answers to the Research Questions

This study begins by asking an overarching question that focuses on understanding the effective coordination process of the key dependencies in DASD projects. This primary research question has guided this research that is being answered with the data from two selected case projects. The formation process of this research question is presented in Chapter 2. The following sections answer each of the sub-questions separately based on the study findings and then address the primary research question of this study.

RQ 1.1 What are the key dependencies in DASD?

Dependency types and antecedents

A dependency represents a bi-directional, logical and time-dependent correlation between the activities it involves. Dependencies are logical as there is a cause-and-effect correlation between the activity's start and finish. Similarly, a time constraint exists in this correlation that has a potential or actual impact on the activities' start, execution and completion. In this research, a dependency is considered a '*key dependency*', which has the potential to cause a high impact on the activity outcomes.

There are two distinct antecedents that directly influence the types of dependencies found in DASD projects. Firstly, there are four types of dependency relationships found

in this study's cases: *inter-team*, *intra-team*, *technical* and *process*. Each dependency relationship contains one or more types of dependency grouped under three high-level categories: knowledge, activity and resource. All the high-level dependency types per dependency relationship are summarised in Table 8.1.

The *inter-team* and *intra-team* dependency relationships are the most important categories of dependency relationships since they were involved with all three types of dependencies. The dependency relationships and high-level dependency types were defined during the within-case analysis discussed in Chapters 4 & 5 and are summarised in Chapter 7 as part of the model development.

Table 8.1: Summary of Dependency Types per Dependency Relationship

Dependency Relationships	Dependency types
Inter-team	• Knowledge • Activity • Resource
Intra-team	• Knowledge • Activity • Resource
Technical	• Resource
Process	• Activity

Besides, several situational factors influence the dependency types and their occurrences. The dependency analysis identified four situational factors: *Dispersion*, *Project*, *Product* and *Team*. The dispersion factor includes the temporal and spatial distances, which were the critical selection criteria for the cases of this study. The two cases of this study presented three dimensions of distribution that might exist in any DASD project, those are:

1. Development/Technical teams are globally distributed
2. Team members are globally distributed
3. Customers or vendors are globally distributed

Depending on the project context, these dimensions may arise individually or in combination. Irrespective of the distributed dimension observed in the projects, they

affect the dependencies and their coordination. For example, the pigeon case inhabited development and testing teams that are globally distributed, which involved the mode of communication and collaboration with NZ-based teams. In contrast, in Bluebird, while all the development teams were co-located, some team members intermittently worked from distributed locations in different project phases which created issues. In particular, the distribution effects were observed in the team members' activity and knowledge dependency.

The type of the project influences the kind of dependencies observed in the DASD projects. The vendor-related dependencies in the Bluebird case were due to the project type. In contrast, no such dependency was identified in the Pigeon case as the project was focused on developing its own product, and no vendor relationship existed in the project activities. The maturity and size of the product under development also influenced the dependencies and their occurrences. In Bluebird, the project activities were related to the customisation of the newly outsourced COTS product, which created a lot of knowledge dependency on the vendor teams and internal teams to understand and transfer the existing system features to the new one. In Pigeon, the project activities focused on a mature product's feature enhancement and bug fixing. Therefore module teams were primarily independent of knowledge dependencies compared to the other case. However, for both the case projects, the product's size created complexity in developing a shared understanding of the task knowledge, which impacted the inter-and intra-team dependency coordination.

Finally, the specialisation or expertise of the teams involved in the project impacted the type of dependencies faced daily. In Bluebird, several teams engaged in the COTS customisation project were not involved in the development activities. The types of dependencies these teams faced were not similar to the ones faced by the technical development teams. Therefore, the types of dependencies are created and influenced by the team's expertise which further dictates the coordination process.

In terms of occurrence, knowledge and activity dependencies are the most frequently occurring between the teams, as identified in both cases. The Bluebird case had a higher amount of knowledge dependency between the teams than the Pigeon case. One of the possible explanations for this high degree of knowledge dependency is the maturity of the product. Since the Pigeon case was working on a mature product, teams already had good knowledge about the product and its functionality. On the contrary, the software under development in the Bluebird case was immature. Therefore development teams had a high degree of domain and expertise knowledge dependency on other teams. Nonetheless, task knowledge dependency was vital in both projects. A lack of shared understanding of the task knowledge impacted the coordination of this dependency and the activity dependencies, which was highly noticed in the Pigeon case. The inter-team resource dependency was infrequent compared to the other two types of dependencies. It was mainly found when there was a need for collaborative decision-making between the teams.

In terms of impact, knowledge and resource dependencies are the most impactful types of dependencies found in the intra-team relationship. In Pigeon, most of the knowledge dependencies were concentrated on the development manager, who possessed most of the product and domain-specific knowledge. Besides, the same person was the team's utmost decision-making authority, which caused an unhealthy dependency structure. A similar situation occurred in the Bluebird case, where the two intermittently distributed vendor-resourced developers analysed significant knowledge and resource dependency. Activity dependencies were the most common type of dependency in both case projects, which the project members and higher authorities acknowledged well. Hence, all but the dependency on the testing and QA activities were managed well.

The technical dependency relationship is mainly associated with both cases' inter-module or product feature dependencies. In both case projects, there were dependencies on the testing environments, which were considered technical

dependencies. This form of technical dependency was extreme in the Bluebird case as there was a pipeline of technical environments for development, testing, QA, UAT and production, which required consistent environment configuration.

The activity-related constraints found in both cases were considered business process dependencies. The criticality associated with this dependency type was the sequential bug-fixing activities flow and strict time-bounded deadline defined in the Pigeon case. In contrast, the pre-defined request and feedback process imposed by the vendor formed a process dependency that all client teams had to follow. Since the process and activity flow were well-established, there was a low incidence of such dependencies in the Bluebird case.

Dependency Risk

While considering the impacts of the key dependencies, the dependency risk concept has emerged. The dependency analysis indicates that several dependencies have associated risks that could cause high coordination impacts. A dependency risk is a potential vulnerability within the dependency that can become a threat leading to the possible effects on the dependency outcome if not managed well. Delay and Poor-quality are the two types of dependency risks that emerged from the data, as discussed in the preceding chapters. In Pigeon, the delay and poor-quality dependency risks had a high potential impact. In contrast, the Bluebird case dependencies were mostly exposed to the increased effects of delay risks. Since the product was still under-development, there was a low risk of poor-quality impacts, which justifies the low impact occurrence for the Bluebird case.

Among these two types, the dependency risks potentially leading to delays were the most prominent in both cases. All the key dependency types were associated with risks of delay impacting the project's progress. In contrast, the activity and resource dependency types only had associated poor-quality dependency risks. The root causes of dependency risks were grouped into knowledge, work and process management issues. The knowledge and process management issues were the most common in all

dependencies. These two issues are causing all the dependency risks associated with knowledge and resource dependency. Since work management issues primarily caused the risks associated with activity dependencies, it was unsurprising not to find any impacts of such issues on other dependency types. Additionally, this study identified dependency risks as more intense during the project's requirements gathering, scheduling, and feature development phases. The primary cause of this criticality was the high involvement of the distributed members in those phases.

Answer to RQ1.1

The key dependencies in distributed agile software development are the knowledge, activity and resource dependency. The findings related to these key dependencies can be summarised as follows:

- Knowledge dependencies are requirements, expertise, task and historical dependencies.
- Activity dependencies are composed of work-output and business process dependencies.
- Resource dependencies are composed of the entity and technical dependencies.
- Dependency types are influenced by inter-team, intra-team, technical and process dependency relationships.
- Dependency types are influenced by Dispersion, Project, Product and team situational factors.

Since the key dependencies have high dependency risks involved, It is significant to find the key dependencies and coordinate them effectively to mitigate the risks observed in the DASD projects.

RQ 1.2 How are they coordinated?

The coordination mechanisms applied to manage all types of dependencies are grouped into three high-level categories: Coordination activities, Coordination tools and artefacts and Structure. Multiple coordination mechanisms are selected to manage the dependencies; they form a coordination strategy that varies depending on the dependency characteristics and its antecedents. For example, synchronised group

meetings via conference calls and screen sharing were the primary mechanisms applied in both case projects to coordinate knowledge and activity dependencies between the local and distributed teams. At the same time, an asynchronous way of coordination was preferred to coordinate with distributed clients (Pigeon) and vendors (Bluebird). Though the distributed aspect was common in both scenarios, coordination strategy varied for coordinating the dependencies based on the dependency relationship. A summary of the coordination strategies used in the cases and their association with agile practices are presented in Table 8.2.

Coordination Activity

Coordination activities are the primary activities involved in managing the dependencies for all dependency relationships and can be either synchronous or asynchronous. In a synchronous coordination activity, the information exchange, discussion, negotiation and decisions are made synchronously where the participants are either present in person or over the virtual media. These activities are the most common coordination activities found in both cases. For example, sprint planning, code review, sprint review and retrospective meetings are synchronous coordination activities used to manage intra-team dependencies. Several other mechanisms not recognised as agile practices were also used as coordination mechanisms. For example, in the daily meetings, the inclusion of external team members (e.g. SMEs) and measuring the team's confidence to meet the sprint goals using Fist-to-Five are not recommended in any agile frameworks but applied in the Bluebird with the existing agile practices to make it effective.

Whereas the asynchronous coordination activity does not involve face-to-face communication or instant information or decision exchange. Several asynchronous coordination activities are commonly used for inter- and intra-team coordination needs in the DASD projects. For example, sending emails was the preferred mode of communication between the developers and the tester in the Pigeon case. In contrast,

support teams and clients primarily relied on emails to communicate with Team R members for Adhoc and urgent requests.

Tools and Artefacts

To coordinate with the local and distributed parties, various tools and artefacts support the synchronous and asynchronous coordination activities. In both case projects, group meetings were conducted over the conference call using Skype and WebEx to connect the local and distributed members virtually. JIRA and Confluence supported most forms of task tracking and knowledge sharing involved in most coordination activities. Physical or electronic mediums containing different information and knowledge supported the coordination activities in every stage of the DASD process. Artefacts such as product or requirements backlog for requirement tracking and sharing, epics and user stories to hold user and story details, squad boards, wall-mounted posters and boards for inter-team dependency and issues tracking, burndown charts for sprint progress tracking, retrospective boards for team improvement goals sharing are the key artefacts found in both cases. The format and the use of these artefacts are depended on the form and purpose of the coordination activities; therefore, the same coordination activity might use different artefacts in different phases of the project.

Structure

High Proximity to local and distributed teams and members, continuous availability of knowledge and resource persons, readily available substitutes of key resource personnel and coordinator role for facilitating boundary spanning activities are the components of the Structure mechanism. Co-location of team members and key expertise and authority resources such as subject matter experts, PM, and Development Managers improved the proximity of the resources necessary. Continuous open communication channels, Slack status, and wall clocks showing the local times of distributed members' locations were helpful mechanisms to achieve availability. Intra-team substitutability was achieved by having redundant skills and expertise in the teams, whereas inter-team substitutability was achieved by taking over

the tasks beyond expertise. Several designated roles, such as Scrum Master, Proxy PO, and Product and Project manager, were acting as coordinators to support inter-team coordination between the local and distributed teams.

There are noticeable differences in the coordination mechanisms used for dependency relationships which are further influenced by the dispersion factors. While only formal group meetings facilitated by a coordinator role were the primary mechanism for inter-team dependencies, informal and formal coordination strategies were used to coordinate dependencies frequently within the team. If the teams or members were distributed, the meetings turned into virtual meetings with the help of technologies to connect all the participants.

Answer to RQ1.2

The coordination mechanisms used to coordinate three types of DASD dependencies in the DASD projects form three distinctive categories:

- A group of synchronous or asynchronous activities are the primary coordination activities for managing the dependencies supported by a repertoire of tools and shared artefacts.
- Structure mechanisms ensure that the required resources, within the team or external, are available in reasonable proximity. There should be substitutes available for both inter-and intra-team levels to take over each other's tasks.
- A coordinator role plays an important part which is considered as part of the structural mechanism that ensures the boundary spanning activities between local and distributed parties.
- The selection and application of these coordination mechanisms are affected by the dependency relationships, dispersion and development process

Table 8.2: Summary of Coordination Strategies from the cases

Coordination Strategy	Coordination Mechanisms	Pigeon	Bluebird	Agile Practice
Synchronous coordination activities	Group Meeting (Formal & Adhoc)	√	√	√
	Daily standup	√	√	√
	Sprint planning	√	√	√
	code review	√		√
	QA Demo	√		√
	Informal F2F meetings	√	√	
	Sprint Review	√	√	√
	Sprint Retrospective	√	√	√
	Voice Conference Call	√		
	Video Conference Call	√	√	
	Screen Sharing	√	√	
	2-week Requirement Discovery		√	
	Release planning meeting		√	√
	Scrum-of-Scrums meeting		√	√
	Fist to Five		√	
	Backlog refinement		√	√
Asynchronous Coordination Activities	2-Week Sprint	√	√	√
	Email Communication	√	√	
	Comments and Descriptions of Tickets	√		
Tools	JIRA	√	√	
	WebEx	√	√	
	Skype	√	√	
	Confluence	√		
	Slack	√	√	
	Code Versioning tool (TFS)	√	√	
	Retrospective tool	√		
	Smartphone with gimble stabiliser		√	
	SMART tool	√		
Artefacts	User story	√	√	√
	Electronic Squad board / Storyboard	√	√	
	Release backlog	√	√	√
	‘How to’ docs	√		
	Software code	√	√	
	Sprint backlog	√	√	√
	Product Backlog	√		√
	Online Forum		√	
	Physical Dependency Board		√	
	Physical Risks and Issues Board		√	
	Requirements backlog		√	√
	Epics with Personas		√	√
	Physical Squad Board		√	√
	Definition of Done (DoD) poster		√	√

	Definition of Ready (DoR) poster		√	√
	Physical Retrospective Board		√	√
	Burndown chart	√	√	√
Availability	Continuously available Resource Member	√		
	Open communication Channel for instant communication		√	
	Status in Slack	√	√	
	Multiple Wall clocks for different time zones		√	
Proximity	Co-located team members	√	√	√
	Co-located key resource personnel	√	√	√
	Shared knowledge of roles and responsibilities of other teams		√	
Substitutability	Redundant skills and expertise in the team	√	√	√
	Task sharing between teams	√		
Coordinator role	SM	√	√	√
	DM	√		
	Proxy PO	√		√
	Product Manager	√		
	Co-located POs		√	√
	Project Manager		√	

RQ 1.3 Are the coordination mechanisms effective?

The third sub-question of this research asks about the effectiveness of the coordination mechanisms, which is answered using the Coordination Effectiveness (CE) concept. The formal definition of coordination effectiveness was developed based on the analyses of case evidence specific to the DASD context,

"*a state of coordination in which all the right things are available and fit for purpose in the right place, at the right time. And all the local and distributed teams and their members have a shared understanding of the project goals and associated tasks, their priorities, know what is going on and when, who is doing what and how their works fit with other team's work and know what to do and when, who knows what and where who is available in different time zones*".

It is also drawn from related research on coordination in agile software development (D. E. Strode et al., 2011). It suggests that CE is better understood by two components: explicit and implicit components.

CE Components

Explicit components confirm the availability of the right thing, at the right place, at the right time. Implicit components are shared awareness of the goal, project activities, priorities, allocations, inter-dependencies, and location and availability of the local and distributed resources. This study adopted the above components while forming an understanding of coordination effectiveness in the DASD context.

In addition, we have developed our own criteria to evaluate the effectiveness based on our analysis framework. In accordance with that criterion, the mechanism used in coordinating a dependency can be considered *effective* if it either aids in eliminating or reducing the risks associated with that dependency. This evaluation criterion directly links the dependency risks and the coordination effectiveness.

The case findings have confirmed the accuracy of the relationship. For example, several participants in the Pigeon case indicated that they have coordination issues with one of the remote teams, which have strong functional overlapping. One of the primary reasons for this inter-team coordination issue was uncertain high-priority work requests, potentially creating complexities on both sides.

We analysed this as an issue of not having a shared understanding of inter-team dependencies and the mechanism of early detection of dependent works that will allow sufficient lead time to the processing team. A shared understanding of inter-team dependencies is a cognitive idea of how one's work may impact others' work or vice versa. Without having this understanding, teams would wait until the last moment when they finally reach a stage when they urgently need some work from other teams, otherwise will fail to complete the work within the deadline (probable case of delay). These issues also have quality impacts, such as under-delivery and frustrations between teams that will impact their relationship. Therefore, a *poor or lack of shared understanding* of inter-team dependencies *negatively* impacts the dependency and leads to dependency risks. Therefore, 'know-how interdependent' can be considered

an implicit component of coordination effectiveness, which is directly related to dependency risks, as illustrated in Figure 8.1.

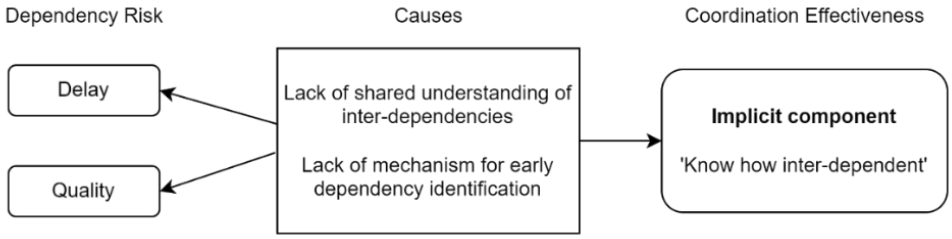

Figure 8.1: Example of extracting Coordination Effectiveness Component

Similarly, this study analysed all the dependency issues and causes identified in both cases to link to a component of coordination effectiveness, as illustrated in Figure 8.2. There is a total of eight components emerged from the data. Three of them are categorised as the explicit component: *right thing*, *right place*, *right time*, and the rest five are categorised as implicit components: *know-how interdependent*, *know who is available*, *know who knows what and where located*, *know what is going on and when*, *know what to do and when*.

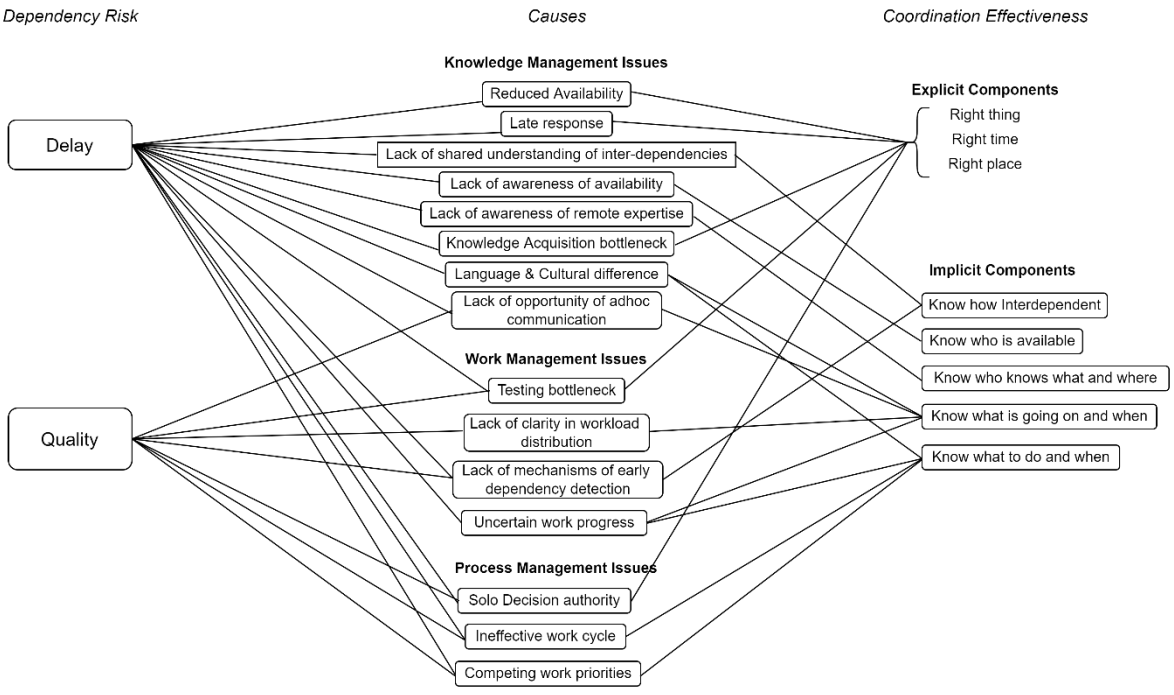

Figure 8.2: Relationship between Dependency risk vs Coordination Effectiveness

Answer to RQ1.3

The absence or lack of a coordination mechanism to alleviate the dependency risks would lead to a poor state of coordination effectiveness. An effective coordination strategy combining appropriate coordination mechanisms would eliminate or reduce dependency risks, leading to a high state of coordination effectiveness.

Achieving coordination effectiveness is essential for avoiding the risks related to the dependency outcomes, which two types of components could operationalise:

- Explicit components are achieved by confirming the right thing is present in a usable state at the right place, at the right time.
- Implicit components are achieved by developing cognitive awareness among the participants. This awareness facilitates shared knowledge about the task such as *knowing why doing the task*, *knowing what to do and when*, and *knowing how tasks are interdependent*. Additionally, this awareness supports teams' external knowledge, such as *knowing who knows what and where*, *knowing who is available*, *know who is doing what*, *know what is going on and when* in terms of the overall project and release.
- A mix of coordination mechanisms fulfilling the explicit and implicit components might be required to manage the dependencies effectively.

RQ 1.4 What are the challenges and factors associated with the coordination mechanisms?

Six high-level categories of coordination challenges were identified: (i) dispersion, (ii) team communication, (iii) vendor-client coordination, (iv) process, (v) organisation and (vi) technical.

Challenges

Due to the temporal and physical distances between the teams and members of the team, coordination between these distributed members was challenging. The communication challenges were caused by an inappropriate choice of communication channel, lack of engagement, and reduced availability and lack of responsiveness of the participants. Insufficient bandwidth of communication through the solo liaison

person, misalignment of the work process, and lack of trust due to lack of familiarity between the vendor and client team members were the primary challenges in vendor-client coordination. The process-related challenges resulted from the misalignment of the code management practices between teams, lack of shared understanding and misunderstandings about the system, product and process, and complexity in the data extraction from the legacy systems. Imbalanced responsibility and authority distribution, disturbance due to the work environment and misalignment between the self-organising teams were primary causes of organisational challenges. Finally, problematic tools and added layers of security and access permission created technical challenges in distributed coordination.

This research analysed the root causes linked to the coordination challenges and identified that the most critical coordination challenges are related to temporal and geographical dispersion, which is one of the primary challenges of DASD coordination (Talukder et al., 2017). The second most-critical coordination challenge is linked to the communication issues that are also well-acknowledged in the DASD literature (Alzoubi & Gill, 2014). The analysis showed that some challenges are unavoidable and need alternate mechanisms to reduce the effects of challenges. For example, dispersion is the standard and integral factor associated with the DASD project coordination; therefore, removing this challenge is inevitable, and the impacts can only be relegated by applying alternate coordination mechanisms, such as open communication channels to improve availability, improve familiarity between the distributed members to develop trust and mutual relationship. On the contrary, several coordination challenges are comparatively easy to resolve. For example, the key organisational challenges, such as hierarchical authority and central dependency structure, could be resolved by following the industry's best practices (Alzoubi et al., 2016; Alzoubi & Gill, 2020; Bick et al., 2017; Marthe Berntzen et al., 2021). Emphasising horizontal decision-making by encouraging self-organising teams could resolve the challenges with authority structure

(Hoda et al., 2010, 2012). This research proposed a list of mechanisms based on the case data analysis to reduce or eliminate the coordination challenges' impacts.

Factors

This research also highlighted three types of confounding factors (contextual, organisational and technological) that are associated with these issues. A better understanding of these factors and their relationship with the challenges would help the selection and adoption of appropriate mechanisms in alleviating the coordination challenges.

Answer to RQ1.4

The challenges associated with the DASD project coordination are resulting from

- temporal and physical dispersion, inter-team communication, vendor-client communication, process, organisation and technical issues.
- And these issues are contributed from contextual, organisational and technological factors.

Answer to RQ1

This research seeks to answer the primary research question to understand how the key dependencies are effectively coordinated in the DASD project. Based on the answers to the sub-questions, this study has understood the following:

- Knowledge, activity and resource dependencies are the key dependency types in distributed agile software development projects. These types may vary depending on the dependency relationships and situational factors.
- A coordination mechanism combining different activities, tools, artefacts and structural mechanisms is used to manage the dependencies. A coordination strategy is required to carefully select the mechanisms depending on the type of dispersion, dependency relationship and process.
- The effectiveness of the coordination relies on how well the risks associated with the dependencies are mitigated. An effective coordination strategy, a combination of explicit and implicit components, is required to mitigate the probability of delay and poor-quality outcomes.

- Six types of coordination challenges that need to be mitigated to reduce their impacts on the coordination while achieving coordination effectiveness.

8.2 Discussion

The initial conceptual model presented in Chapter 2 was based on two prominent works of coordination in agile software development and distributed software development (Espinosa, Lerch, et al., 2002; D. Strode, 2012). The model was continually refined and adapted, and the components of the final proposed model of coordination in DASD are drawn from both the findings of this research and relevant concepts from the contemporary literature.

The dependency types identified in this study are similar to the dependency taxonomy of agile software development projects as presented by (D. E. Strode, 2016). The conceptual model of this study includes a dependency taxonomy for the DASD projects, which is a revised version of Strode's work on coordination in agile software development projects. The following are the differences between Strode's (D. E. Strode, 2016) and the findings of this study, (i) *task knowledge dependency*, (ii) *process dependency*.

This study determined that the proposed '*task allocation*' knowledge in Strode's taxonomy is only a subtype of '*task knowledge*' dependency, and there are several other types of task-specific knowledge DASD context as discussed in Chapter 7 (see figure 7.3). This study introduced task knowledge dependency as a subtype of knowledge dependency, which redefined the task allocation dependency type of Strode's taxonomy. In a DASD setting, task-related knowledge creates a number of dependencies that can lead to several impacts if not managed effectively. Due to the physical and temporal dispersion, disseminating different forms of task knowledge among the teams and members is challenging, which requires proper coordination strategy to effectively manage this type of dependency.

Similarly, the '*Process dependency*' type of Strode's taxonomy was redefined as '*Activity dependency*', a dependency relationship that influences the dependency types.

At the same time, activity dependency in the taxonomy of this study represents interdependencies between activities, which can be characterised as work-output or business process dependency. The differences between this study's dependency taxonomy to Strode's taxonomy are highlighted in Figure 8.3 (yellow for revised, green for new).

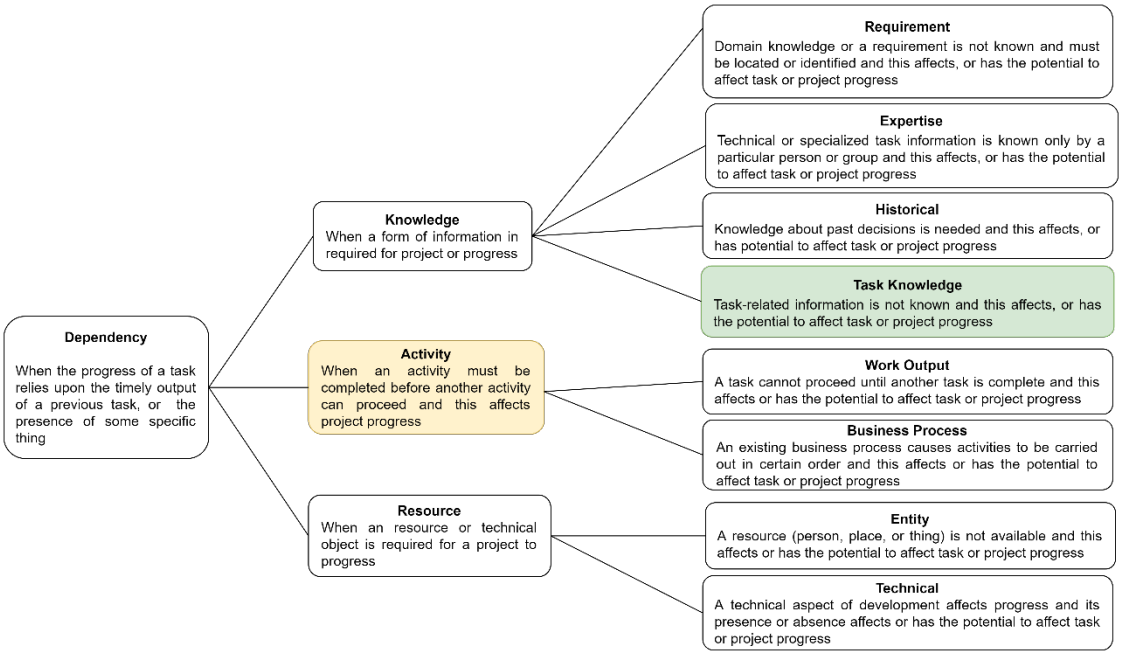

Figure 8.3: Revised and New components of Dependency Taxonomy

The coordination strategy concept of the model presented in this study supports findings from relevant studies on agile software development and distributed coordination in the extant literature. Strode (2012) proposed the term coordination strategy in an agile software development context by identifying the synchronisation, structure and boundary-spanning components. Espinosa et al. (Espinosa, Lerch, et al., 2002) identified that a strategy was needed to achieve a highly coordinated state by utilising a combination of implicit and explicit mechanisms in global software development contexts. This study defines coordination strategy in DASD as a combination of mechanisms that are carefully selected and applied to effectively manage the dependencies, which aligns well with the concept of coordination strategy proposed by both Strode and Espinosa. Moreover, this study found/ determined that an effective coordination strategy should be altered depending on the situation and need,

and the same mechanism might not be effective in all scenarios ‘one size does not fit all’ (Espinosa, Lerch, et al., 2002).

Since most coordination activities involving distributed participants are forms of boundary-spanning activities and also synchronous in nature, we found that there is a strong dependence on a coordination role to facilitate the coordination. This conceptualisation differs from Strode’s (2012) coordination strategy components, in which coordinator roles are only not involved in the synchronisation activities. Espinosa et al. (Espinosa, Lerch, et al., 2002) conceptualised coordination strategy as a mix of explicit and implicit mechanisms, where explicit mechanisms include task organisation and team communication, and implicit components consists of various forms of team cognition (e.g. shared mental model, transactive memory, group mind and task awareness). The conceptual model proposed in this study does not specifically include the combination of explicit and implicit mechanisms, as a specific mechanism can serve as both an explicit and implicit mechanism at the same time, and therefore difficult to characterise the explicit and implicit behaviour of a coordination activity.

The proposed model of this study presents an elaborated view of the coordination strategy for the DASD context as shown in Figure 8.4. The components differing from Strode and Espinosa’s work are highlighted (yellow for adapted, green for new). The model outlines that an effective coordination strategy in a distributed agile software development project is a combination of synchronous and asynchronous coordination activities which are supported by tools, shared artefacts and structure mechanisms. This conceptualisation supports the taxonomy of the coordination proposed by (Berntzen et al., 2022) who identified different forms of Meetings, pre-defined roles, and tools and artefacts as key elements of inter-team coordination in large-scale agile software development, including distributed teams.

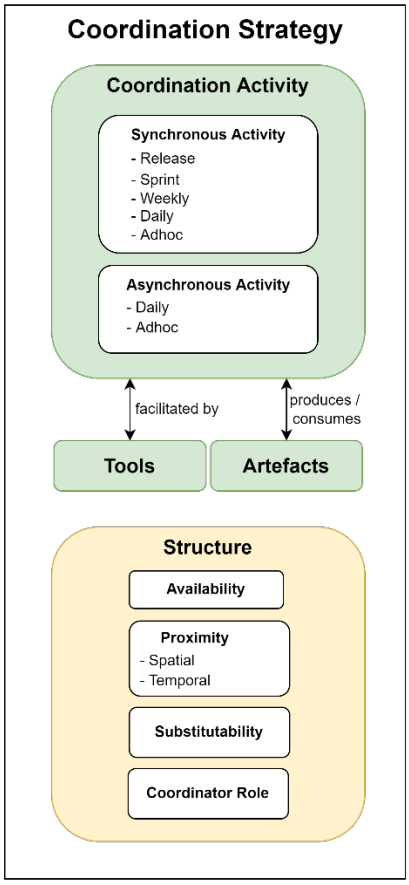

Figure 8.4: Components differing from Strode and Espinosa et al.'s work

The findings of this study concur with most of the coordination effectiveness components (highlighted as yellow in Figure 8.5) suggested by Strode et al. (D. E. Strode et al., 2011). In addition, two important implicit components were identified for developing a shared cognitive understanding between the local and distributed parties. The two new components, '*know how dependent*' and *'know who is available*' are highlighted as green in Figure 8.5. The '*know how dependent*' component indicates the level of shared understanding of different types of dependencies that helps to proactively identify, notify, negotiate and manage the interdependencies with distributed teams. The '*know who is available*' indicates the level of shared awareness of the availability of distributed resources and their preferences. Without having such awareness, people might not know when the required person will be available or might use a communication channel that is not the preferred way to get a quick response. This finding is consistent with that of Espinosa et al. (Espinosa, Lerch, et al., 2002) who defined this form of awareness as knowledge of 'who is around' to support coordination

in distributed software development. Therefore, this component was added to the preliminary model presented in Chapter 2 and further supported by the study's findings.

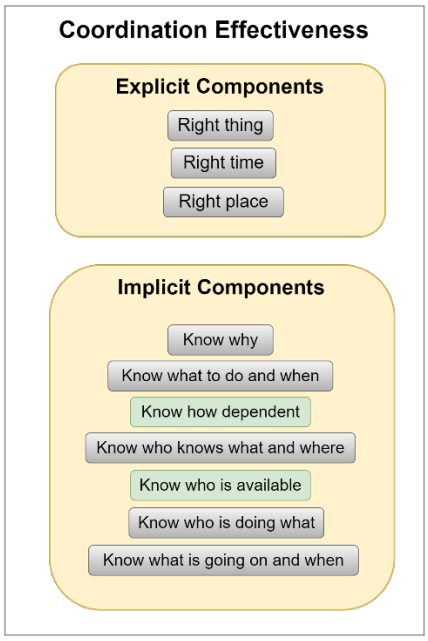

Figure 8.5: New components of Coordination Effectiveness

The situational factors presented in the model of coordination are supported by the work of Espinosa et al. (Espinosa, Lerch, et al., 2002). Espinosa proposed three situational factors: task, team and context, that influence the dependencies in the GSD context. While the findings of this study concur with Espinosa's study, there are some differences. Based on the findings from this study, four situational factors that affect the dependencies in a DASD context are identified. They are Project, Product, Team, and Dispersion. Similar to Espinosa et al.'s work, this study found that team and dispersion (i.e., context) factors influence the dependency types. While there is no direct impact or influence on the dependencies due to task factors, the project type may have an indirect relationship with the task factors, which in turn might influence the dependency on the DASD project. For example, depending on the type of project, the tasks might vary which would result in different kinds of dependencies. Therefore, the task factor can be considered a subtype of project factors that indirectly influences the project dependencies.

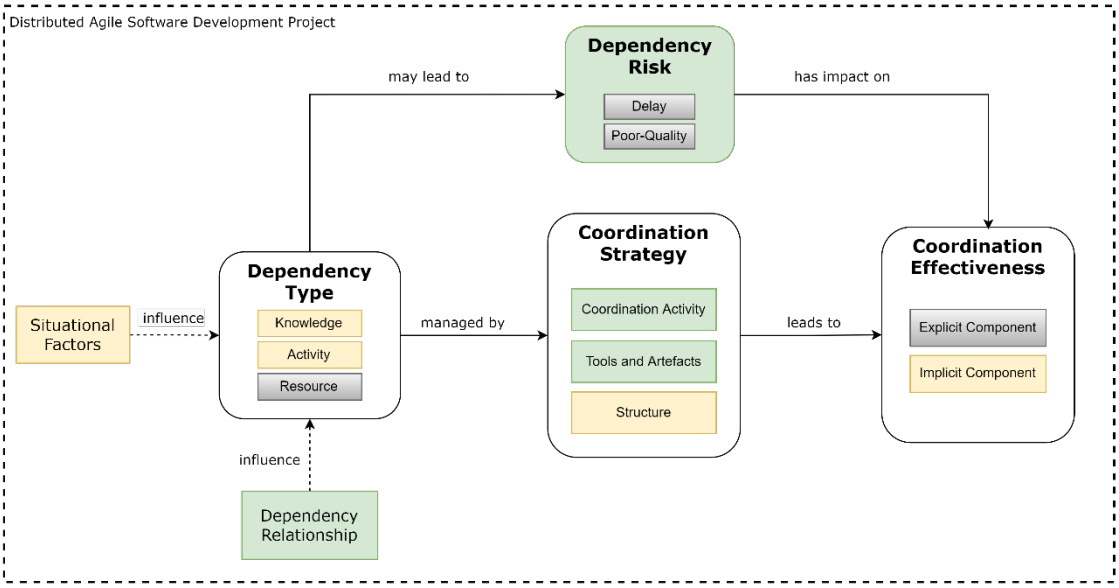

Figure 8.6: Distinctive contributions in the proposed model of coordination

The proposed model makes a unique contribution to the literature on coordination in the DASD context as presented in Figure 8.6 highlighting new (green) and adapted (yellow) components. Firstly, the proposed model sheds new light on the four different types of dependency relationships, which strongly influence the types of dependencies and their coordination in the DASD context. While most extant literature on DASD coordination either focuses on inter-team or team-level dependency relationships, there is no formal definition of dependency relationships or their implications on the dependency types presented in those studies. Secondly, this study determines two high-level categories of dependency risks, *delay* and *poor quality*, that could potentially hamper the coordination and project performance. This thesis provides a deeper insight into each type of dependency risk and its impacts on different parts of the DASD process. Moreover, this model highlights that different project and contextual factors can influence different types of dependencies, which could result in dependency risks. Finally, this model presents the relationship between dependency types, risks and coordination effectiveness. The study findings indicate that the vulnerabilities associated with the dependencies may lead to dependency risks and the risks could impact the coordination effectiveness if not appropriately managed.

8.3 Lessons learned and Recommendations

A number of important learnings can be noted to stimulate the expected participation of the remote worker in the activities related to their Scrum-based agile development process (e.g. daily Scrum meetings, Sprint planning, Sprint reviews, Sprint retrospectives).

1. The mediating technologies needed to be low friction to use. This would take some experimenting with both at the remote worker's site and the squad's work area. Criteria related to this include:
    - It is the responsibility of a squad member to make sure that the video equipment is available and working and that the remote worker would be available, prior to any planned team meeting.
    - The signup process to join the video conference should be straightforward for the remote worker.
    - The video screen should be large enough and suitably positioned for the entire team to see and hear the remote worker, so the squad has a constant sense of their presence.
    - The squad microphone should be sensitive enough so that as squad members speak, they can be heard by the remote worker. This proved challenging at times because the squad was sometimes spatially spread out in a large room. The squad resorted to passing a microphone around to members who were speaking, which could be a bit disruptive to the normal flow of conversation.
    - From the remote worker's perspective, they emphasised that the technology needed to be simple to be set up and use. One remote squad member left the technology set up at all times in a dedicated space to avoid the setup overhead and lower the risk of it not functioning when needed.
2. A highly visible clock set to the time in the remote worker's time zone increased squad empathy for the remote worker's situation during a squad meeting (generally working at an inconvenient or undesirable time from the remote perspective). The clock also helped to coordinate squad members' expectations regarding response times or availability of the remote worker.

3. A number of participants indicated that it would have been useful to have an indicator of the presence and availability of the remote worker. This would help with managing ad hoc requests and communications and the uncertainty of response times or availability. The remote workers interviewed felt this could be too obtrusive unless they were in control of the presence indicator. These observations align well with the findings of (Steinmacher et al., 2013) who investigate the need for awareness support in distributed software development.

4. One of the possible ways to mitigate this testing bottleneck could be to start the testing activities as early as possible in the sprint which will give enough time for both the tester and developers to coordinate and resolve defects.

5. Since dependencies between team members were quite dense (knowledge, roles and work), the need for spontaneous, unplanned coordination was quite common, usually short face-to-face interactions. Time zone differences and equipment set-up delays made this type of short spontaneous interaction challenging. This was largely obviated by one remote worker who synchronised his time at work with the team and had a video link always on while he was at work, in a dedicated space. While this may not suit all remote worker's situations, it certainly increased coordination opportunities and reduced delays in workflow.

6. Forming self-organising team behaviour will be a key to mitigating the single point of knowledge and decision-making bottleneck. Additionally, redefining the roles of the team members and empowering them to self-assign all the team tasks will reduce the imbalanced work distribution.

7. Remote workers reported that they missed the situational awareness of co-location with the rest of the squad. From a coordination perspective, this included:

    - Over-hearing spontaneous interactions that surfaced a coordination need or misunderstanding that turned out to be relevant to their own work or they were able to contribute based on their own experience and knowledge.

    - Visibility of the many artefacts in the work environment that acted as information radiators to coordinate knowledge about previous decisions, principles and information that should be kept front-of-mind. These points relate well to the role of situational awareness in coordination (Endsley, 1995) as well as the importance of workplace ambience (Mishra et al., 2012).

8. Remote workers reported delays in their work because they depended on remote access to certain databases and product features behind a firewall. They

suggested that this be coordinated prior to them travelling to the remote work location.

9. Lack of mutual trust in the distributed team (Brede & Šmite, 2012) was not an issue in this case since the remote worker had also spent considerable time.

8.4 Research Evaluation

This research signifies that coordination is a contemporary challenge in software development which is further complicated in a distributed context. This research uncovered from a systematic literature review of the empirical literature that there is a gap in the current body of knowledge supporting effective coordination in DASD. The systematic review process adopted in this study followed the established guidelines and procedures proposed in the literature (Dybå & Dingsøyr, 2008; Kitchenham & Charters, 2007b; Okoli & Schabram, 2010). Following this finding, the main body of this thesis focused on presenting the key aims and outcomes of two empirical case studies analysis.

While evaluating qualitative research, there is a recognition that the activities for conducting the study should follow the standards of good practice (i.e. methodological rigor), as well as the findings and interpretations of the findings, should meet the criteria of trustworthiness (Fossey et al., 2002). This research adopted a mix of positivist and interpretivist paradigms that combined knowledge from the extant literature (positivist) and interpretation of empirical case study findings to conceptualise coordination in the DASD project. This research applied the four aspects of quality assessment suggested by Yin (2002), construct validity, internal validity, external validity and reliability. The application of these aspects in this study are discussed in the following paragraphs.

Construct Validity

Construct validity concerns that the instruments used in the study should be designed in a way that supports the researcher's preconceived notions of the 'subjective' judgements (Yin, 2002). Achieving construct validity in any research that applies

qualitative methods is quite challenging. As the definition suggests, there should be a strong correlation between the operational measures used to collect data and the study's specified constructs. This research focuses on the following key constructs: dependencies, coordination mechanisms and strategies, coordination effectiveness and coordination challenges. We have developed a well-defined interview protocol that has been reviewed by the experts (i.e. members of AUTEC) and has been used throughout the data collection. Following the recommendation of (Yin, 2018) and (Dubé & Paré, 2003), we have used multiple sources of evidence to achieve data triangulation and established a chain of evidence using a predefined research protocol. Finally, the draft reports of the study have been reviewed by representatives of the participants. To maintain a 'chain of evidence', we have followed the case study protocol presented in Chapter 3 and asked the participants to review the case descriptions. Taken together, these measures have reduced the construct validity concerns by establishing a chain between the theoretical constructs of interest with the data collected from multiple sources.

Internal Validity

Internal validity primarily concerns causal relationships between the resulting concepts (Yin, 2018). This type of causal relationship building is useful for explanatory types of studies. For an exploratory case study similar to current research, such causal links are not a key focus. However, our preliminary model portrays assumed causal relationships between the concepts, dependencies, coordination mechanisms and coordination effectiveness. An additional layer to causal relations is identified while developing the framework of analysis. The framework shows probable relationships between key dependencies and dependencies, which further have an impact on coordination effectiveness. Since the framework has been developed based on preliminary data analysis, it addresses the internal validity concerns. Likewise, we have investigated the interview transcripts, filed notes and observational data to identify evidence to confirm the causal link among those concepts.

External Validity

External validity emphasises the generalizability of the research findings to a broader community. While doing multiple-case study research, Yin (2018) suggested that replication logic can be applied to avoid such validity threats. As mentioned in the design section (Chapter 3, section 3.2), cases have been selected based on specific criteria that will support replication. While performing the cross-case analysis, the researcher explicitly searched for cross-case patterns matching the within-case findings. These patterns support the generalisation of the case findings. However, generalizing any research outcome for all scenarios is challenging. The researcher has explained the outcomes with relevant examples that might be helpful for readers to understand their applicability in similar contexts.

Reliability

Reliability is concerned with the repetition of the same process by different researchers to generate similar outputs (Dubé & Paré, 2003). The primary intention behind reliability is to ensure that the data sources are correct and free from researcher bias. One of the requirements for allowing others to repeat the same process is by documenting the data collection process. Yin (2018) recommended the use of a defined case study protocol and maintaining a case study database to ensure reliability. In this study, we have developed and followed a case study protocol that has been well-discussed in the data collection section. For maintaining the case study database, an electronic file-based database has been used to store all relevant data and artefacts. Each case data are organised in separate folders and each folder is named using a unique case identification code to help data management. While reporting the findings, we have used both direct and indirect references from the case data to present the logical chain of evidence (Benbasat et al., 1987). We believe that it will help the reader to judge the reliability of the information.

Table 8.3: Attributes used for Methodological Rigor in this research adapted from Dubé & Paré

Attribute	Purpose	Application in this study
Research Design		
Clear research question	• Links the study's theoretical and practical contributions • Addresses reliability concerns (Miles & Huberman, 1994b)	• A key research question is formed, with three supporting questions
A priori specification of constructs	• Shapes the initial research design showing constructs of interest	• A preliminary model of coordination is developed grounded on relevant literatures and researcher's professional experience
Clean theoretical slate	• Reduce research bias while analysis • Limits researcher's interference on limiting the findings	• The conceptual model does not detail the concepts and relationships as priori. Only guides to form a high-level specification • A framework of analysis is further formed combined with initial data analysis that serves as data analysis framework
Multiple-case design	• Produce robust and more generalised results than single case • Enable cross-examination of case results to maximise theoretical coverage	• Two similar cases are selected satisfying the selection criteria.
Replication logic in multiple-case design	• Case selection based on substantive significance or theoretical relevance • Addresses external validity concerns (Yin, 2002)	• Both the cases are selected to provide literal replication
Unit of analysis	• Define a clear boundary within the case • Guide the data collection and applicability of theoretical contributions	• The Unit of analysis is the coordination process in DASD • Developed theory is applicable for coordination in DASD
Context of the case study	• Describe the contextual elements in which the research was conducted • Enhance the credibility	• Project contexts are clearly described for each case • Cross-case analysis based on contextual analysis outcomes
Data Collection		
Elucidation of the data collection process	• Elucidate the data collection process including o what data has been collected o how those data are collected o what sources are used and why o how collected data contributes to the findings • Enhance reliability and validity (Benbasat et al., 1987)	• Data collection section of this chapter elaborates the data collection process by depicting the types of data collected, their sources, and how we have preserved the chain of evidence
Multiple data collection methods	• Collect evidence from multiple sources to provide rich insight of the investigating phenomenon • Ensure construct validity (Yin, 2002)	• Data collected from multiple sources, such as individual participant interview, participants observations, field notes taken by multiple person, photographs
Data triangulation	• Converge evidence from	• Data triangulation is performed in two ways:

	multiple data sources to strengthen research outcomes • Enhance construct validity (Yin, 2002)	○ Data collected from multiple person of different roles ○ Data collected from multiple data sources • Findings are verified from multiple source of evidence whenever possible
Case study protocol	• Documenting details of the research procedures • Ensure the research process can be repeated by other investigators and reach to similar conclusion • Enhance research validity and reliability (Yin, 2002)	• A clear case study protocol has been developed including all the required instruments (see appendix for list of study instruments)
Case study database	• Organise and manage study documents • Enhance research reliability (Yin, 2002)	• A Cloud-based electronic case study database is maintained to store each case documents for ease of use
Data Analysis		
Elucidation of data analysis process	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability and reduce researcher bias	• Details of the data analysis process is discussed in Data analysis section (section 3.6)
Field notes	• Supporting evidence for other data collection methods	• Notes have been taken by two interviewers during the interview and observation sessions.
Coding and reliability check	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability and reduce researcher bias	• Content Analysis techniques are followed (Braun & Clarke, 2006; Fereday & Muir-Cochrane, 2006b) • Codes are verified by another research team member for validity
Data displays	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability	• Tabular presentation of key findings accompanying the data source
Logical chain of evidence	• Demonstrate how the final outcomes are drawn from the data • Enhance reliability and construct validity (Yin, 2002)	• Each participant is titled with a code while transcribing, codes are used to present their links to the findings • Documents are stored in a case database including case details, and participant codes.
Explanation building	• Demonstrate how data are interpreted to textually explain the phenomenon	• Delay diagrams, Social Network analysis is used to explain the criticality of the dependencies.
Searching for cross-case patterns	• Look for similarities and differences between cases using categories and dimensions	• Each within-case finding are used to perform cross-case analysis and search for patterns to develop high-level themes
Quotes (evidence)	• Supporting evidence to develop theoretical concepts and show their relationships	• Direct and indirect quotes are used to present case findings, that act as a chain of evidence to the theory
Comparison with extant literature	• Strengthen study findings by showing supportive or contradictory literatures	• Existing literatures are used to support the findings and theory development

Additionally, this study adopted the attributes from Dubé & Paré (2003) to achieve methodological rigor (as discussed in Chapter 3) and how the attributes were applied in

this study presented in Table 8.3. The attributes covered the criteria for evaluating rigor and relevance in this research design, data collection and data analysis process.

A multiple case study design was selected to produce a robust and more generalised result. However, the primary intention of this research was not to generalise the findings but rather to develop a clear and in-depth understanding of coordination to develop theoretical insights. Two literal replicated case projects were selected to address the external validity concerns (Yin, 2002). The contextual elements of each case were described before presenting the analysis outcomes to enhance credibility. A clearly defined unit of analysis provided a boundary within the case that further guided the data collection and analysis. As described in Chapter 3, multiple data sources were utilised to achieve data triangulation and findings were verified from considerable data evidence to enhance the construct validity. The predefined case study protocol with ethically approved instruments provided a well-structured guideline for improved repeatability of the research activities, thus strengthening the validity and reliability of this research. Finally, multiple content analysis techniques (i.e. activity analysis and coordination analysis) were applied using the activity-based framework (Korpela et al., 2002) and a priori-developed coordination analysis framework, respectively. The analytic framework, data analysis process, findings and interpretations were presented separately to demonstrate how the outcomes were drawn, creating a logical chain of evidence to contribute to the reliability and construct validity (Yin, 2002).

In summary, this research followed well-established guidelines to ensure the quality of the outcomes. Besides, this study maintained a clear chain of evidence to help the readers in following the data analysis and interpretation process. All the research questions are being answered based on the study findings and discussed to indicate this study's contributions grounded on the literature review. Finally, an evaluation of the research is presented to establish methodological and interpretive rigor achieved in this study.

Table 8.4: Summary of the criteria used for evaluating the quality of the research findings

Criteria	Considerations	Application in this research
Authenticity	To what extent the findings and interpretations are presented, including participants' views, range of voices and perspectives (including dissenting views)	Participants' own voices and views were interspersed, either verbatim or partial, throughout the textual description Avoided unnecessary explanations or interpretation of participant's actions while describing the findings; instead, emphasised interpreting the motives of the actions to correlate with the purpose of coordination Obtained first-hand information from all the participants in person using semi-structured interviews and direct observation Gained familiarity with the participants' day-to-day activity while taking interviews and other forms of data collection to develop Total of 17 face-to-face interviews were conducted, and ten coordination meetings were observed in the two cases
Coherence	To what extent the presentation of findings and interpretations are plausible (i.e., fits the raw data and verified from multiple sources)	Data analysis followed a well-defined and elucidated process for reliability (discussed in Chapter 3) Maintain the chain of evidence by following (Yin, 2018) for construct validity Direct and indirect references were used from the case data to present the logical chain of evidence Activity analysis followed the activity theory-based framework (Korpela et al., 2002) (discussed in Chapter 3) Developed and applied a framework, supported by extant literature and preliminary data analysis, for analysing the coordination process (discussed in Chapter 3) Methodological rigor is achieved by adapting the guidelines (Dubé & Paré, 2003) (discussed in Chapter 3) Used diagrams, tables, figures and other schematics for a better visual representation of the context and concepts Findings and their interpretations were presented separately to remove the researcher's interpretation bias Developed a comprehensive understanding of coordination in the DASD project, informed by context and content
Reciprocity	To what extent are the participants involved in the data analysis, findings and interpretations	All the interview participants reviewed case statements Initial data findings were shared with the case representatives Pigeon case organisation approved recommendations, and immediate changes were made to resolve the coordination issue

Typicality	To what extent the claims of this research are generalisable to other bodies of knowledge, populations, or contexts/settings	Multiple case study designs for robust and generalisable outcomes. Performed within-case analysis for both case projects, followed by cross-case comparison for generalizability Key research findings were evaluated using the up-to-date body of knowledge and agile community for broader acceptability Contextual factors were considered during analysis to establish the applicability of the findings in similar contexts Clearly demonstrated the relevance of the analysis to contemporary agile practices used for DASD coordination The generalizability was achieved by generalising the data to the conceptual model of coordination by thick description and interpretations of the phenomena of interest (A. S. Lee & Baskerville, 2003)
Permeability of the researcher's intentions, engagement, interpretations	To what extent does the researcher's perspective (e.g., role, preconceptions, personal experience) contributes to the findings and interpretations	A Preliminary model of coordination and coordination effectiveness were presented based on the researcher's understanding of existing literature on coordination A high-level conceptual model was used initially to guide the data collection process. However, the final model and associated concepts emerged from the case data analysis limiting the researcher's interference Practical implications were suggested based on the researcher and his team's experience, recommendations from the active participation of the agile community in NZ (Agile Meetup) Whenever needed, the researcher's perception and experience were used to clarify the criticality of the situations using mock-up scenarios matching with the DASD process

Chapter 9 : Conclusion

"*Coming together is a beginning; keeping together is progress; working together is success.*" — Henry Ford

The above quotation of Henry Ford best describes the rule for the success of coordination. Working together in a way where every person involved knows what is going to be built, what to do and when to achieve that goal and can act accordingly. In doing so, they need to understand how they depend on others and how to properly manage those dependencies to achieve the collective goal by minimising associated issues and their impacts. This understanding of coordination has had a profound influence on this research.

This concluding chapter summarises this study by revisiting the objectives of this research and how those objectives have been achieved. Following this is a discussion highlighting the contributions of this research toward Information System (IS) and Software Engineering (SE) literature and practitioners. The final sections of this chapter then acknowledge the limitations of this study leading to future research recommendations.

9.1 Revisiting the Research Objectives

This thesis was motivated by the relevance of effective coordination in Distributed Agile Software Development (DASD). The key objective of this research was to develop a deeper understanding of the key *concepts of coordination* and identify the *means of coordination* that could support *effective coordination* in DASD projects grounded on empirical evidence. Another objective of this study was to address the apparent lack of theoretical models of coordination in the DASD context.

While fulfilling the research objectives, a preliminary coordination model was developed based on systematic reviews of the existing literature, which guided further research

activities. This study adopted an unconventional approach by mixing the positivist and interpretivist paradigms, which, together with the research objectives, influenced the selection of the multiple case study research methodology. This study adopted a well-established case study protocol to direct the research activities, outlining the study design, data collection, data analysis, model development, and validation activities. Two cases were selected, fulfilling this research's selection criteria, followed by data collection from multiple sources to achieve data triangulation. The data analysis followed the widely used content analysis technique to create high-level themes. The two-phased within-case analysis was conducted using the activity theory-based framework and the preliminary framework of analysis developed for this research. A cross-case analysis followed this to conceptualise coordination in the DASD context.

The case studies were conducted sequentially, i.e., the data collection and first rounds of data analysis for the first case were completed before continuing to the second case. Several instruments were used while conducting the analysis, such as coordination theory for diagnosing the dependencies and delay diagrams for analysing dependency issues and impacts. The findings of the within-case and cross-case analysis were discussed separately, and then the conceptual coordination model was presented. While doing so, the research pursued to understand better coordination and its integral parts in DASD projects (i.e., the dependency types and their antecedents, dependency risks, coordination strategy), how these parts could impact the coordination (i.e., the relationship between these key concepts), and how the coordination needs could be managed to achieve a well-coordinated state (i.e., coordination effectiveness) which is called '*the success of coordination*'.

Finally, the answers to the research questions were presented based on the analysis findings, which were further examined from the literature perspective to outline the research contributions. An evaluation of the research design, activities and data interpretations was presented to establish the robustness of this study's design and findings.

9.2 Contributions to Knowledge

This research has made several novel contributions to the IS literature covering the substantive and conceptual domains of Brinberg and McGrath's validation schema (Brinberg & McGrath, 1983).

The substantive domain includes the substance of the deliverables that this research has produced, such as insights, methodologies, and artefacts. The literature review established that there is a lack of knowledge in understanding the coordination process in the DASD context. In response, this research has addressed this gap by exploring and summarising the coordination process of two distributed agile software development projects in their natural settings. This understanding of coordination encompasses an understanding of the types of dependencies that are found within the process, including the factors and risks associated with them (answered by RQ1.1), what coordination strategy can be used to manage the dependencies effectively (answered by RQ1.2 and RQ1.3) and the challenges and their cofounding factors (answered by RQ1.4).

This study has explored the coordination activities for three different dimensions of distribution that may present in DASD projects. This study also identified four distinct forms of relationships that influence the characteristics of the dependencies and their coordination strategies. The study findings highlighted that while achieving effective coordination in DASD, a mix of coordination mechanisms, both agile and non-agile activities and artefacts supported by various tools and structures, must be carefully selected to form a coordination strategy. This result endorses the findings of the extant literature (D. E. Strode et al., 2012; Xu & Cao, 2006), which is further extended to fit the DASD project characteristics.

One of the contributions to the IS research is that this work extended Espinosa et al.'s (Espinosa, Lerch, et al., 2002) work to address agile contexts. The conceptualisation and framework of coordination presented in their work focused only on the globally distributed software development context, which did not include agile practices and

their contributions to effective coordination. This study identified that several agile values and practices support effective coordination in the DASD context.

Additionally, this study extends Strode and her colleagues' work (D. Strode, 2012; D. E. Strode et al., 2011), focusing on co-located agile teams, to the distributed agile context. While doing so, this research has revisited the notion of dependency taxonomy and proposed a revised version applicable to DASD projects. This new taxonomy version includes task knowledge dependency, which is not present in the existing dependency taxonomy of agile software development. Additionally, the concept of coordination effectiveness, initially proposed by Strode et al. (2011) for agile SD projects, is being extended to accommodate additional implicit components applicable to DASD projects. The coordination model of this research complements Strode et al.'s (2011) coordination effectiveness concept, which states that both explicit and implicit components are essential for achieving effective coordination. However, this study underlines that the implicit components play a more vital role than the explicit components in achieving coordination effectiveness in DASD. This model also contributes to addressing the limitations of Strode et al.'s model by extending the applicability of their model in distributed teams.

Another substantive domain contribution of this study is the articulation of the primary selection criteria to form an effective coordination strategy. There is no process or guidelines found in the extant literature on achieving effective coordination (Moe et al., 2010; D. E. Strode et al., 2012; Yu & Petter, 2014), particularly in DASD projects. This study proposes the dependency risk concept, which strongly influences the coordination effectiveness of DASD projects. This study draws the relationship between dependency risk and coordination effectiveness and proposes that this relationship has the potential to guide the formation of effective coordination strategies for DASD projects. This relationship could be simplified as the coordination effectiveness will be higher for a particular dependency if the coordination mechanisms applied effectively reduce or eliminate the dependency risks associated with that

dependency. We used this business rule to evaluate the coordination effectiveness for both projects discussed in the preceding chapters. This relationship and the criteria for evaluating coordination effectiveness is an important contribution to this research. Since Strode's model of coordination effectiveness was incomplete in defining such a relationship, it was important to draw the connection to support evaluating the effectiveness of the coordination.

The conceptual domain contributions include interpreting the observations in the form of concepts, novel patterns, frameworks, abstraction and theories. This study presents a conceptual model of coordination for DASD based on empirical evidence from two case studies. The model integrates relevant theories from core research streams such as coordination and distributed agile software development to provide a deeper understanding of dependencies and measures for effective coordination. The model has profiled three distinct concepts: dependency, coordination strategy and effectiveness. However, there are distinct differences in each of the components that are specifically applicable to the DASD projects, as discussed in Chapter 7.

Additionally, this model profiled the dependency risk dimension that has not been articulated in the DASD coordination research as far as our knowledge is concerned. This research delineated that there are two distinct types of dependency risks, delay and poor quality, linked to the key dependencies that strongly influence effective coordination in DASD projects. Notably, this finding may differ in other project contexts. However, the results offer a basic understanding of the dependency risks and their causes associated with each dependency category that may be anticipated in similar contexts.

Another conceptual domain contribution is the framework of analysis developed in this research, as discussed in Chapter 3, to support the analysis of the coordination process in DASD projects. Although the framework was used along with the activity-based framework (Korpela et al., 2002) in this study, the framework can also be used in other coordination research. The framework has the potential to be used as a lens for

analysing dependencies and associated risks and coordination mechanisms for achieving coordination effectiveness in different distributed settings (i.e., distributed, large-scale, or hybrid).

A final contribution of this conceptual model is that it provides a base for extending this model in future work to form a complete theory of coordination in distributed agile software development. Besides unique contributions to IS research, this model extends the knowledge of coordination that could be valuable for other research fields. For example, the inter-and intra-team relationships proposed in this study would be useful for teamwork research in both co-located and distributed contexts. Moreover, the coordination strategy and coordination effectiveness concepts also contribute to the project management fields of research in distributed settings.

9.3 Contribution to Practice

This research contributes to the IS practice in various ways. The conceptual model proposed in this study could help practitioners to understand the fundamental concepts of coordination in DASD projects. One of the primary contributions of this model is the categorisation of dependencies that could be of three types: knowledge, activity and resources dependency. These high-level dependencies and subtypes could help practitioners to identify appropriate coordination strategies to mitigate the associated dependency risks. For example, this study identifies that task knowledge dependency, a subtype of knowledge dependency that includes a shared understanding of the tasks and their executions, such as task interdependence, task priority, task ownership, task progress or blockers. Since task knowledge is cognitive and mostly depends on implicit coordination mechanisms; therefore, practitioners need to pay close attention while selecting an appropriate coordination strategy that supports the development of such shared understanding.

Secondly, the coordination model presented in this study highlights a different form of dependency relationships that may exist in different parts of the software development process. This study identifies that these dependency relationships significantly

influence the dependencies faced and their coordination mechanisms. Having a good understanding of the inter-and intra-team, technical and process dependency relationships, practitioners could predict the type of dependencies that may occur during the software development process and how to manage them effectively to mitigate the dependency risks.

Thirdly, the dependency risks concept outlines different types of vulnerabilities and associated risks due to unmanaged dependencies in DASD projects. The practitioners could use the model to understand these dependency risks and their impacts related to the criticality and importance of the dependency. Additionally, this research presents a prescription of coordination mechanisms for alleviating dependency risks in achieving coordination effectiveness. It serves as a repertoire of mechanisms for improving coordination performance.

Another contribution of this research is that it conceptualises the coordination strategy that acts as a guideline in the coordination strategy formation process. Practitioners could choose from the two modes of coordination activities, including agile and non-agile practices. In terms of agile method adoption, both Scrum and XP practices are most commonly used in DASD projects, but there is no hard-and-fast rule in this adoption, and practitioners can adopt any agile method depending on their needs. While coordinating with distributed parties, the coordination activities rely on technology-mediated communication, such as voice or video conference calls, for conducting meetings and knowledge exchange. In practice, DASD project members could select the communication mode and channel depending on the need; however, this study identifies that video conference calls perform better than voice conference calls as participants can see each other and get their physical expressions, improving their engagement in the meeting.

In a DASD context, proximity, availability, and substitutability are the three structural components that are crucial for team-level coordination. This research suggests that agile practices create opportunities for frequent communication and synchronisation

between co-located team members, which improves proximity. Nonetheless, availability and proximity are a concern for distributed members. This research identifies that interpersonal relationships and additional efforts are imperative to enhance continuous availability and feelings of proximity between the team members. This research also identifies that achieving substitutability at the team level is equally essential for both co-located and distributed team members; therefore, it should be encouraged and practised to improve self-organising behaviour.

This research proposes that the coordination role is an integral part of the effective coordination strategy, particularly while managing the dependencies using boundary-spanning activities. Though the most popular agile methods, such as Scrum and XP, do not mention any specific role for coordination, this research identifies that several roles specified in these methods, for example, Scrum Master, Product Owner, or Proxy PO, can take the role of coordinator. The coordination role could take care of the team's external communication and negotiations between co-located and distributed parties, which could reduce the development teams' coordination burdens.

Another finding with implications for practice relates to managing coordination between highly interdependent distributed teams. The lack of appropriate mechanisms for dependency identification and negotiation is one of the coordination issues that are not adequately supported by agile practices (Bick et al., 2017). Therefore, practitioners could adopt a hybrid approach combining traditional practices for joint release planning and dependency negotiation at the inter-team level and agile methods for coordinating team-level dependency coordination. A hybrid coordination approach not only facilitates control between the distributed parties but also provides flexibility in adopting changing requirements and Adhoc decision-making.

This study indicates that a mismatch between the development and testing cycles is associated with dependency risks. Practitioners are advised to use Test-Driven-Development (TDD) approach to align better the activities related to these two cycles. A smaller iteration cycle, including the development-testing-QA activities, will also be

effective in reducing inefficient work cycles and bug occurrences. This would also provide a better-quality work output and team and customer satisfaction.

Another significant contribution from this research is the understanding of coordination challenges associated with DASD projects. The study identified specific coordination challenges to contextual, organisational, and technical issues. Paying close attention to understanding these challenges could help practitioners identify their causes and adopt appropriate strategies to address them.

9.4 Limitations and Future Research

The limitations of this study are similar to those common to qualitative case studies, and some are unique to this study. A common limitation of qualitative case studies is the lack of generalisability (Yin, 2018), which is valid for this multiple case study research. One of the main reasons is that the selection of the cases was convenient based on geographic proximity (New Zealand) rather than a random sampling of the case and participants. However, the two cases were selected to meet the specific criteria and to contribute to the concept definition and relationship building (Eisenhardt & Graebner, 2007). The cases were selected based on literal replication to achieve similar results that will contribute to preliminary understanding (Eisenhardt, 1989), as well as generalisability, to some extent. Therefore, though the findings of this study cannot be generalised to all kinds of DASD, they provide evidence and contribute to developing a clear and deeper understanding of the complex nature of coordination in DASD projects. Although the concepts and relationships of the coordination model are tentative and need to be verified to provide statistical generalisability, this study followed well-established guidelines to meet the rigor and relevance of case study research (discussed in Chapter 7). Moreover, the findings of this study are drawn from both the case evidence and empirical literature on coordination. Therefore, this research addresses the validity and reliability concerns of the findings.

Other limitations peculiar to this research are the case selection, data collection and analysis and concept development phases. The cases of this study are selected based

on the criteria discussed in Chapter 3 to meet the literal replication requirements; however, it is difficult to find cases that are exactly similar to each other. Since each project is different in nature and varies depending on the organisation, it is critical to achieve pure replication. There might be other organisations and DASD projects that best match the selection criteria, but it is difficult to pursue and convince them to participate in the research.

In terms of data collection, the primary limitation is that the number of participants was small and did not involve participants from different teams. Including a large number of participants from different teams might have contributed to understanding other aspects of coordination, which would have further strengthened the study findings. Moreover, collecting data from the end users, customers or other stakeholders was not possible, which might have provided different perspectives of coordination. Another limitation was capturing the data within a short time. As the interviews were conducted at a specific time of the project, the study might not have captured some dependencies that might have occurred in the past due to participants' inability to recall past events.

While considering the limitations in the data analysis, the interpretivist paradigm is always susceptible to researcher bias. Since qualitative data provides richer and more readily available details that could lead to misinterpretations. Moreover, while observations provide informative evidence, it is difficult to gather a large amount of data through observation as it takes longer and more difficult to organise them. However, this study observed twelve different sessions to collect substantive evidence of the investigating phenomenon. Another limitation of the data analysis involves the verification of thematic coding. The validity of coding could have been enhanced by having a third person not directly involved in the research review the process. However, the coding process was validated by one of the supervisors who repeated the process which was further reviewed by the second supervisor.

9.4.1 Future Research

There are limitations concerning the conceptual model presented in this research. Although the dependency relationships and their impacts are drawn based on the evidence of the case data, they may not be applicable to other project contexts. Furthermore, there may have been other situational factors in the process not revealed in the data. Additionally, the dependency risk concept and its association with the coordination effectiveness concept were not common in both cases. Therefore, further investigation is called for to validate these concepts of the model. The relationship between the coordination strategy and coordination effectiveness is grounded on the empirical literature and case evidence, which address the validity concerns of this relationship. However, the conceptualisation is derived from a small amount of data which needs to be validated with large amounts of data from similar case studies.

In addition, the model of coordination of this study considers proximity as a structural mechanism essential for improving DASD coordination. Though achieving close proximity in a distributed context is problematic (Conchuir et al., 2009; D. Strode, 2012), it could be beneficial if a certain level of spatial and temporal proximity could be achieved. However, this research provides little guidance on achieving that level of proximity, which could be a potential area of investigation in future work on DASD coordination.

This study proposes that the better the coordination mechanisms mitigate the dependency risks, the higher the coordination effectiveness is. However, this research could not provide any quantitative criteria for measuring how well the dependency risks are mitigated by the coordination mechanisms. Another possibility is that there might be multiple combinations of coordination mechanisms to avail the same level of coordination effectiveness, but this research does not provide any conclusive evidence on this aspect. Moreover, the coordination strategy formulation process was not sufficiently evidenced by the data, instead was developed based on the researcher's

interpretation and understanding of the phenomenon, which requires further investigation.

In future, it would be possible to extend this research to cover large-scale distributed software development projects using scaled-agile frameworks. Large-scale DASD is gaining popularity in recent years demanding more empirical research, and the model of coordination model presented in this research could be used as a basis for that. Another useful area of future research would be to apply the proposed analysis framework in other research related to coordination and verify the applicability and usefulness of the framework.

Another prospective future research area would be to define, operationalise and empirically verify the concepts of coordination effectiveness to test its relation to DASD project success. While measuring the coordination effectiveness, the Socio-Technical Congruence (STC) concept proposed by Cataldo et al. (2008) could be a possible starting point, since the effectiveness of coordination closely related to both social and technical aspects. The systematic mapping study by Sierra et al. (2018) presented several methods that could be applied in this process. However, since the STC only considers task dependencies and their effective coordination, the methods need to be revised to include all types of dependencies proposed in this study. Additionally, the current body of knowledge has not yet defined DASD project success and its relationship to coordination performance; it could be a noteworthy area of contribution to the IS and SE research field.

References

Ågerfalk, P. J., Conchúir, E. Ó., Olsson, H. H., & Fitzgerald, B. (2009). Global Software Development: Where are the Benefits. *Communications of the ACM*, *52*(8), 127–131. https://doi.org/10.1145/1536616.1536648

Ågerfalk, P. J., & Fitzgerald, B. (2006). Flexible and distributed software processes: old petunias in new bowls. *Communications of the ACM*, *49*(10), 27–34. https://doi.org/10.1145/1164394.1164416

Ågerfalk, P. J., Fitzgerald, B., Holmström, H., Lings, B., Lundell, B., & Conchúir, E. Ó. (2005). A framework for considering opportunities and threats in distributed software development. *Proc. International Workshop on Distributed Software Development, Paris, France: Austrian Computer Society*, *August*, 47–61. https://doi.org/10.1.1.93.5181

Ågerfalk Pär J., Fitzgerald, B., Holmström Olsson Helena, & Ó Conchúir Eoin. (2008). Benefits of Global Software Development: The Known and Unknown. In D. and R. D. M. Wang Qing and Pfahl (Ed.), *Making Globally Distributed Software Development a Success Story* (pp. 1–9). Springer Berlin Heidelberg.

Al-Ani, B., & Redmiles, D. (2009). Trust in distributed teams: Support through continuous coordination. *IEEE Software*, *26*(6), 35–40. https://doi.org/10.1109/MS.2009.192

Altaf, A., Fatima, U., Butt, W. H., Anwar, M. W., & Hamdani, M. (2019). A systematic literature review on factors impacting agile adaptation in global software development. *ACM International Conference Proceeding Series*, 158–163. https://doi.org/10.1145/3348445.3348463

Alyahya, S. a, Ivins, W. K. b, & Gray, W. A. b. (2013). Raising the awareness of development progress in distributed agile projects. *Journal of Software*, *8*(12), 3066–3081. https://doi.org/10.4304/jsw.8.12.3066-3081

Alyahya, S., Bin-Hezam, R., & Maddeh, M. (2022). Supporting Remote Customer Involvement in Distributed Agile Development: A Coordination Approach. *IEEE Transactions on Engineering Management*, 1–14. https://doi.org/10.1109/TEM.2021.3131964

Alyahya, S., Ivins, W. K., & Gray, W. A. (2011). Co-ordination support for managing progress of distributed agile projects. *Proceedings - 2011 6th IEEE International Conference on Global Software Engineering Workshops, ICGSE Workshops 2011*, 31–34. https://doi.org/10.1109/ICGSE-W.2011.24

Alzoubi, Y. I., & Gill, A. Q. (2014). Agile global software development communication challenges: A systematic review. *Proceedinfs of the Pacific Asia Conference on Nformation Systems (PACIS),* Paper 20. http://aisel.aisnet.org/cgi/viewcontent.cgi?article=1371&context=pacis2014

Alzoubi, Y. I., & Gill, A. Q. (2020). An Empirical Investigation of Geographically Distributed Agile Development: The Agile Enterprise Architecture is a Communication Enabler. *IEEE Access*, *8*, 80269–80289. https://doi.org/10.1109/ACCESS.2020.2990389

Alzoubi, Y. I., & Gill, A. Q. (2022). Can Agile Enterprise Architecture be Implemented Successfully in Distributed Agile Development? Empirical Findings. *Global Journal of Flexible Systems Management*. https://doi.org/10.1007/s40171-022-00298-w

Alzoubi, Y. I., Qumer, A., & Al-ani, A. (2016). Empirical Studies of Geographically Distributed Agile Development Communication Challenges : A Systematic Review. *Information & Management*, *53*, 22–37. https://doi.org/10.1016/j.im.2015.08.003

Ambler, S. (2012). Agility at scale survey: Results from the Summer 2012 DDJ State of the IT Union Survey. *Http://Www.Ambysoft.Com/Surveys/StateOfITUnion201209.Html. Acessed on 11/05/2022*, *22*(09), 2018.

Anderson, D. J. (2010). *Kanban: successful evolutionary change for your technology business*. Blue Hole Press.

Andres, H., & Zmud, R. (2002a). A contingency approach to software project coordination. *Journal of Management Information Systems*, *18*(3), 41–70. https://doi.org/10.1080/07421222.2002.11045695

Andres, H., & Zmud, R. (2002b). A contingency approach to software project coordination. *Journal of Management Information Systems*, *18*(3), 41–70. https://doi.org/10.1080/07421222.2002.11045695

Anh, N.-D., Cruzes, D. S., & Conradi, R. (2012). Dispersion, coordination and performance in global software teams. *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '12*, *Idi*, 129. https://doi.org/10.1145/2372251.2372274

Aranda, J. (2010). *A Theory of Shared Understanding for Software Organizations*. Graduate Department of Computer Science University of Toronto.

Awad, M. A. (2005). A Comparison between Agile and Traditional Software Development Methodologies. *The University of Western Australia*, *1*, 1–300. https://doi.org/10.1145/130840.130843

Bannerman, P. L., Hossain, E., & Jeffery, R. (2011). Scrum practice mitigation of global software development coordination challenges: A distinctive advantage? *Proceedings of the Annual Hawaii International Conference on System Sciences*, 5309–5318. https://doi.org/10.1109/HICSS.2012.512

Barlow, J. B., Keith, M. J., Wilson, D. W., Schuetzler, R. M., Lowry, P. B., Vance, A., & Giboney, J. S. (2011). Overview and guidance on agile development in large organizations. *Communications of the Association for Information Systems*, *29*(July 2011), 25–44. https://doi.org/10.2139/ssrn.1909431

Barlow, J., & Keith, M. (2011). Overview and Guidance on Agile Development in Large Organizations. *Communications of the Association of Information Systems (CAIS)*, *29*, 25–44.

Bass, J. M. (2016). Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology*, *75*, 1–16. https://doi.org/10.1016/j.infsof.2016.03.001

Baxter, P., & Jack, S. (2008). Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers. *The Qualitative Report*, *13*(4), 544–559. https://doi.org/citeulike-article-id:6670384

Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*.

Bendix, L., & Pendleton, C. (2014). Collaboration in the Absence of Communication. *2014 IEEE 23rd International WETICE Conference*, 294–299. https://doi.org/10.1109/WETICE.2014.81

Berntzen, M., Hoda, R., Moe, N. B., & Stray, V. (2022). A Taxonomy of Inter-Team Coordination Mechanisms in Large-Scale Agile. *IEEE Transactions on Software Engineering*, 1–1. https://doi.org/10.1109/tse.2022.3160873

Berntzen, M., & Wong, S. I. (2021). Autonomous but Interdependent: The Roles of Initiated and Received Task Interdependence in Distributed Team Coordination. *International Journal of Electronic Commerce*, *25*(1), 7–28. https://doi.org/10.1080/10864415.2021.1846851

Bick, S., Scheerer, A., & Spohrer, K. (2016). Inter-Team Coordination in Large Agile Software Development Settings: Five Ways of Practicing Agile at Scale. *Proceedings of the Scientific Workshop Proceedings of XP2016*, 4:1–4:5. https://doi.org/10.1145/2962695.2962699

Bick, S., Scheerer, A., Spohrer, K., Kude, T., & Heinzl, A. (2014). Software Development in Multiteam Systems: A Longitudinal Study on the Effects of Structural Incongruences on Coordination Effectiveness. In *International Research Workshop on IT Project Management (IRWITPM)* (Vol. 1). http://aisel.aisnet.org/irwitpm2014http://aisel.aisnet.org/irwitpm2014/1

Bick, S., Spohrer, K., Hoda, R., Scheerer, A., & Heinzl, A. (2017). Coordination Challenges in Large-Scale Software Development: A Case Study of Planning Misalignment in Hybrid Settings. *IEEE Transactions on Software Engineering*, *5589*(c), 1. https://doi.org/10.1109/TSE.2017.2730870

Bjørnson, F. O., Wijnmaalen, J., Stettina, C. J., & Dingsøyr, T. (2018). Inter-team Coordination in Large-Scale Agile Development: A Case Study of Three Enabling Mechanisms. *XP 2018*, 216–231. https://doi.org/10.1007/978-3-319-91602-6_15

Boehm, B. (2002). Get Ready for Agile Methods, with Care. *Computer*, *35*(1), 64–69.

Booch, G. (2015). The Future of Software Engineering (SEIP Keynote). *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference On*, *2*, 3.

Bose, I. (2008). Lessons Learned from Distributed Agile Software Projects: A Case-Based Analysis. *Communications of AIS*, *23*(December 2008), 619–632. http://aisel.aisnet.org/cais/vol23/iss1/34

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77–101. https://doi.org/10.1191/1478088706qp063oa

Brede, N. M., & Šmite, D. (2012). Understanding a Lack of Trust in Global Software Teams: A Multiple-case Study. *Journal of Information Systems Education*, *23*(3), 243–257. https://doi.org/10.1002/spip

Brereton, P., Kitchenham, B., Budgen, D., & Li, Z. (2008). Using a protocol template for case study planning. *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, *2006*, 8.

Brinberg, D., & McGrath, J. E. (1983). *A Validity Network Schema*.

Buchan, J., Nurul, A. B. M., Talukder, A., & Senapathi, M. (2019). Coordination in Distributed Agile Software Development: Insights from a COTS-based Case

Study. *Australasian Conference on Information Systems*, 942–952. https://www.acis2019.org/Papers/ACIS2019_PaperFIN_192.pdf

Calhoun, C. (1995). *Critical social theory: Culture, history, and the challenge of difference*. Wiley-Blackwell.

Campbell, D. J. (1988). Task complexity: A review and analysis. *Academy of Management Review*, *13*(1), 40–52.

Cao, L., & Ramesh, B. (2007). Agile software development: Ad hoc practices or sound principles? *IT Professional*, *9*(2), 41–47. https://doi.org/10.1109/MITP.2007.27

Carmel, E., & Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. *IEEE Software*, *18*(2), 22–29. https://doi.org/10.1109/52.914734

Cataldo, M., & Herbsleb, J. D. (2013). Coordination breakdowns and their impact on development productivity and software failures. *IEEE Transactions on Software Engineering*, *39*(3), 343–360. https://doi.org/10.1109/TSE.2012.32

Cataldo, M., Herbsleb, J. D., & Carley, K. M. (2008). Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development. *Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '08*, *March*, 2. https://doi.org/10.1145/1414004.1414008

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *Computer*, *34*(11), 131–133. https://doi.org/10.1109/2.963450

Conchuir, E. Ó., Ågerfalk, P. J., Olsson, H. H., & Fitzgerald, B. (2009). Global software development: Where are the benefits? *Communications of the ACM*, *52*(8), 127–131. https://doi.org/10.1145/1536616.1536648

Converse, S., Cannon-Bowers, J. A., & Salas, E. (1993). Shared mental models in expert team decision making. *Individual and Group Decision Making: Current*, *1993*, 221.

Creswell, J. D. W., & Creswell, J. D. W. (2017). *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications.

Crowston, K. (1994). *A Taxonomy Of Organizational Dependencies and Coordination Mechanisms*.

Crowston, K. (1997). A Coordination Theory Approach to Organizational Process Design. *Organization Science*, *8*(2), 157–175.

Crowston, K., Rubleske, J., & Howison, J. (2006). *Coordination theory and its application in HCI*.

Dahbur, K., Mohammad, B., & Tarakji, A. B. (2011). *A Survey of Risks, Threats and Vulnerabilities in Cloud Computing*.

Damian, D. ; U. of V., & Moitra, D. (2006). Global Software Development: How Far Have We Come? *Software, IEEE*, *23*(5), 17–19. https://doi.org/10.1109/MS.2006.126

Darnell, R. C. . (2015). *IMPLICIT AND EXPLICIT MEASURES OF COORDINATION EFFECTIVENESS AS PREDICTORS OF AGILE SOFTWARE DEVELOPMENT PROJECT SUCCESS : A REGRESSION APPROACH* (Issue December). Capella University.

De Souza, C. R. B., & Redmiles, D. (2003). Opportunities for extending activity theory for studying collaborative software development. *European Conference in CSCW (ECSCW 2003)*, 14–18.

Dickinson, T. L., & McIntyre, R. M. (1997). A conceptual framework for teamwork measurement. *Team Performance Assessment and Measurement*, 19–43.

Dietrich, P., Kujala, J., & Artto, K. (2013). Inter-team coordination patterns and outcomes in multi-team projects. *Project Management Journal*, *44*(6), 6–19. https://doi.org/10.1002/pmj.21377

Dingsøyr, T., Bjørnson, F. O., Moe, N. B., Rolland, K., & Seim, E. A. (2018). Rethinking Coordination in Large-Scale Software Development. *2018 ACM/IEEE 11th International Workshop on Cooperative and Human Aspects of Software Engineering*, *July*, 10–12. https://doi.org/10.1145/3195836.3195850

Dingsøyr, T., Dybå, T., Gjertsen, M., Jacobsen, A. O., Mathisen, T. E., Nordfjord, J. O., Røe, K., & Strand, K. (2019). Key Lessons From Tailoring Agile Methods for Large-Scale Software Development. *IT Professional*, *21*(1), 34–41. https://doi.org/10.1109/MITP.2018.2876984

Dingsøyr, T., Moe, N. B., Fægri, T. E., & Seim, E. A. (2017). Exploring software development at the very large-scale: a revelatory case study and research agenda for agile method adaptation. *Empirical Software Engineering*, 1–31. https://doi.org/10.1007/s10664-017-9524-2

Dingsøyr, T., Moe, N. B., & Seim, E. A. (2018). Coordinating Knowledge Work in Multiteam Programs: Findings From a Large-Scale Agile Development Program. *Project Management Journal*, *49*(6), 64–77. https://doi.org/10.1177/8756972818798980

Dingsoyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*, *85*(6), 1213–1221. https://doi.org/10.1016/j.jss.2012.02.033

Dorairaj, S., Noble, J., & Malik, P. (2011). Effective Communication in Distributed Agile Software Development Teams. *Agile Processes in Software Engineering and Extreme Programming (XP 2011)*, 102–116.

Dubé, L., & Paré, G. (2003). Rigor in Information Systems Positivist Case Research: Current Practices, Trends, and Recommendations. *MIS Quarterly*, *27*(4), 597–636. https://www.jstor.org/stable/30036550

Dube, L., & Pare, G. (2003). Rigor in Information Systems Positivist Case Research: Current Practices, Trends, and Recommendations. *MIS Quarterly*, *27*(4), 597–636. https://doi.org/10.2307/30036550

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9–10), 833–859. https://doi.org/10.1016/j.infsof.2008.01.006

Easterbrook, S., Singer, J., Storey, M.-A., & Damian, D. (2008). Selecting Empirical Methods for Software Engineering Research. *Guide to Advanced Empirical Software Engineering*, 285–311. https://doi.org/10.1007/978-1-84800-044-5_11

Eisenhardt, K. M. (1989). Building Theories from Case Study Research. In *Academy of Management Review* (Vol. 14, Issue 4, pp. 532–550). https://doi.org/10.2307/258557

Eisenhardt, K. M., & Graebner, M. E. (2007). Theory Building from Cases : Opportunities and Challenges. *The Academy of Management Review*, *50*(1), 25–32.

Ekasari, D. S., Raharjo, T., & Prasetyo, A. (2022). *Challenges and Solution Recommendation in Large-Scale Agile Implementation: A Systematic Literature Review*. 175–180. https://doi.org/10.1109/icimcis53775.2021.9699312

Emam Hossain, Muhammad Ali Babar, Hye-Young Paik, & June Verner. (2009). Risk Identification and Mitigation Processes for Using Scrum in Global Software Development: A Conceptual Framework. *2009 16th Asia-Pacific Software Engineering Conference*, *December*, 457–464. https://doi.org/10.1109/APSEC.2009.56

Endsley, M. R. (1995). Toward a Theory of Situation Awareness in Dynamic Systems. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *37*(1), 32–64. https://doi.org/10.1518/001872095779049543

Engeström, Y. (1987). Learning by expanding: An activity-theoretical approach to developmental research. *Educational Researcher*.

Espinosa, J. A., & Boh, W. F. (2009). Coordination and governance in geographically distributed enterprise architecting: An empirical research design. *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences, HICSS*, 1–10. https://doi.org/10.1109/HICSS.2009.133

Espinosa, J. A., Kraut, R. E., Lerch, J. F., a. Slaughter, S., Herbsleb, J. D., & Mockus, A. (2002). Shared Mental Models and Coordination in Large-Scale, Distributed Software Development. *International Conference on Information Systems*, 513–518. https://www.cs.cmu.edu/afs/cs.cmu.edu/Web/People/jdh/collaboratory/research_papers/ICIS_2001.pdf

Espinosa, J. A., Lerch, J. F., & Kraut, R. E. (2002). Explicit vs. Implicit Coordination Mechanisms and Task Dependencies: One Size Does Not Fit All. *Process and Performance at the Inter- and Intra-Individual Level*, *January*, 39. https://doi.org/10.1.1.59.8342

Espinosa, J. A., & Pickering, C. (2006). The Effect of Time Separation on Coordination Processes and Outcomes: A Case Study. *Proceedings of the 39th Hawaii International Conference on System Sciences*, *00*(C), 191–209. https://doi.org/10.1109/HICSS.2006.463

Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007a). Familiarity, complexity, and team performance in geographically distributed software development. *Organization Science*, *18*(4), 613–630. https://doi.org/10.1287/orsc.1070.0297

Espinosa, J. A., Slaughter, S. A., Kraut, R. E., & Herbsleb, J. D. (2007b). Team Knowledge and Coordination in Geographically Distributed Software Development. *Journal of Management Information Systems*, *24*(1), 135–169. https://doi.org/10.2753/MIS0742-1222240104

Faraj, S., & Sproull, L. (2000). Coordinating Expertise in Software Development Teams. *Management Science*, *46*(12), 1554–1568. https://doi.org/10.1287/mnsc.46.12.1554.12072

Feiner, M. E. (2016). How Scrum Tools May Change Your Agile Software Development Approach. *Software Quality. The Future of Systems- and Software Development*, *0*(0). https://doi.org/10.1007/978-3-319-27033-3_2

Fereday, J., & Muir-Cochrane, E. (2006a). Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development. *International Journal of Qualitative Methods*, *5*(1), 80–92. https://doi.org/10.1177/160940690600500107

Fereday, J., & Muir-Cochrane, E. (2006b). Demonstrating Rigor Using Thematic Analysis: A Hybrid Approach of Inductive and Deductive Coding and Theme Development. *International Journal of Qualitative Methods*, *5*(1), 80–92. https://doi.org/10.1177/160940690600500107

Fernández, D. M., Penzenstadler, B., Kuhrmann, M., & Broy, M. (2010). A meta model for artefact-orientation: fundamentals and lessons learned in requirements engineering. *International Conference on Model Driven Engineering Languages and Systems*, 183–197.

Fink, A. (2019). *Conducting research literature reviews: From the internet to paper* (4th Editio). Sage publications.

Fossey, E., Harvey, C., Mcdermott, F., & Davidson, L. (2002). Understanding and evaluating qualitative research. *Australian and New Zealand Journal of Psychiatry*, *36*, 717–732.

Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software Development*, *9*(August), 28–35. https://doi.org/10.1177/004057368303900411

Fröling, D. (2018). *Developing a User-Centered Information Radiator*.

Fuks, H., Raposo, A., Gerosa, M. A., Pimentel, M., Filippo, D., & Lucena, C. (2008). Inter- and intra-relationships between communication coordination and cooperation in the scope of the 3C collaboration model. *Proceedings of the 2008 12th International Conference on Computer Supported Cooperative Work in Design, CSCWD*, *1*, 148–153. https://doi.org/10.1109/CSCWD.2008.4536971

Galin, D. (2004). *Software quality assurance: from theory to implementation*. Pearson education.

Ghani, I., Lim, A., Hasnain, M., Ghani, I., & Babar, M. I. (2019). Challenges in distributed agile software development environment: A systematic literature review. *KSII Transactions on Internet and Information Systems*, *13*(9), 4555–4571. https://doi.org/10.3837/tiis.2019.09.013

Glaser, B., & Strauss, A. (1967). Grounded theory: The discovery of grounded theory. *Sociology The Journal Of The British Sociological Association*, *12*, 27–49.

Goldkuhl, G. (2012). Pragmatism vs interpretivism in qualitative information systems research. *European Journal of Information Systems*, *21*(2), 135–146. https://doi.org/10.1057/ejis.2011.54

Guba, E. G., Lincoln, Y. S., & others. (1994). Competing paradigms in qualitative research. *Handbook of Qualitative Research*, *2*(163–194), 105.

Gustavsson, T. (2017). Assigned roles for Inter-team coordination in Large-Scale Agile Development : a literature review. *XP2017*, 1–5. https://doi.org/10.1145/3120459.3120475

Gutwin, C., & Greenberg, S. (2004). The Importance of Awareness for Team Cognition in Distributed Collaboration. In *Team cognition: Understanding the factors that drive process and performance* (pp. 177–201). https://doi.org/10.1.1.17.6293

Gutwin, C., Greenberg, S., & Roseman, M. (1996). Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation. In *People and Computers XI* (pp. 281–298). Springer. https://www.cs.usask.ca/faculty/gutwin/1996/96-WorkspacesAwareness.HCI/Workspace-near-final.pdf

Hazzan, O., & Dubinsky, Y. (2009). *Agile software engineering*. Springer Science & Business Media.

Henrik Vedal, Viktoria Stray, Marthe Berntzen, N. B. M. (2021). Managing Dependencies in Large-Scale Agile. *Agile Processes in Software Engineering and Extreme Programming – Workshops*, *2*, 80–86. https://doi.org/10.1007/978-3-030-88583-0

Herbsleb, J. (2016). Building a Socio-technical Theory of Coordination: Why and How (Outstanding Research Award). *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2–10. https://doi.org/10.1145/2950290.2994160

Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. *FoSE 2007: Future of Software Engineering*, *September*, 188–198. https://doi.org/10.1109/FOSE.2007.11

Herbsleb, J. D., & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, *29*(6), 481–494. https://doi.org/10.1109/TSE.2003.1205177

Heylighen, F. (2016). Stigmergy as a universal coordination mechanism I: Definition and components. *Cognitive Systems Research*, *38*, 4–13. https://doi.org/10.1016/j.cogsys.2015.12.002

Hoda, R. (2011). Self-Organizing Agile Teams : A Grounded Theory. *Learning*, 262. http://hdl.handle.net/10063/1617

Hoda, R., Noble, J., & Marshall, S. (2010). Organizing Self-Organizing Teams. *32nd ACM/IEEE International Conference on Software Engineering (ICSE 2010)*, 285–294. http://homepages.mcs.vuw.ac.nz/~elvis/twikipub/Main/RashinaHoda/OSOT.pdf

Hoda, R., Noble, J., & Marshall, S. (2012). Developing a grounded theory to explain the practices of self-organizing Agile teams. *Empirical Software Engineering*, *17*(6), 609–639. https://doi.org/10.1007/s10664-011-9161-0

Hoda, R., Salleh, N., & Grundy, J. (2018). The Rise and Evolution of Agile Software Development. *IEEE Software*, *35*(5), 58–63. https://doi.org/10.1109/MS.2018.290111318

Hofstede, G., Hofstede, G. J., & Minkov, M. (2005). *Cultures and organizations: Software of the mind* (Vol. 2). Mcgraw-hill New York.

Hole, S., & Moe, N. B. (2008). A Case Study of Coordination in Distributed Agile Software Development. *Software Process Improvement - Proceedings of the 15th European Conference European System & Software Process Improvement and Innovation, EuroSPI 2008, Dublin, Ireland, September 3-5, 2008*, *June*, 189–200. https://doi.org/10.1007/978-3-540-85936-9

Hollenbeck, J. R., Beersma, B., & Schouten, M. E. (2012). BEYOND TEAM TYPES AND TAXONOMIES: A DIMENSIONAL SCALING CONCEPTUALIZATION FOR TEAM DESCRIPTION. *Source: The Academy of Management Review*, *37*(1), 82–106. https://doi.org/10.5465/amr.2010.0181

Hossain, E. (2008a). Coordinating mechanisms for agile global software development. *Proceedings - 2008 3rd IEEE International Conference Global Software Engineering, ICGSE 2008*, 257–263. https://doi.org/10.1109/ICGSE.2008.24

Hossain, E. (2008b). Coordinating Mechanisms for Agile Global Software Development. *2008 IEEE International Conference on Global Software Engineering*, 257–263. https://doi.org/10.1109/ICGSE.2008.24

Hossain, E., Babar, M. A., & Verner, J. (2009). How Can Agile Practices Minimize Global Software Development Co-ordination Risks? *Communications in Computer and Information Science*, *0*(0), 81–92. https://doi.org/10.1007/978-3-642-04133-4_7

Hussain, W., Buchan, J., & Clear, T. (2014). Managing Requirements in Globally Distributed COTS Customization. *Proceedings - International Computer Software and Applications Conference*, *18-21-Augu*, 33–38. https://doi.org/10.1109/ICGSEW.2014.13

Jain, R., & Suman, U. (2016). Effectiveness of agile practices in global software development. *International Journal of Grid and Distributed Computing*, *9*(10), 231–248. https://doi.org/10.14257/ijgdc.2016.9.10.21

Jalali, S., & Wohlin, C. (2012). Global software engineering and agile practices: A systematic review. In *Journal of software: Evolution and Process* (Vol. 24, Issue 6, pp. 643–659). https://doi.org/10.1002/smr.561

Jiménez, M., Piattini, M., & Vizcaíno, A. (2009). Challenges and Improvements in Distributed Software Development: A Systematic Review. *Advances in Software Engineering*, *2009*, 1–14. https://doi.org/10.1155/2009/710971

Kahya, M. D., & Seneler, C. (2018). Geographical Distance Challenges in Distributed Agile Software Development: Case Study of a Global Company. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)*.

Kanaparan, G., & Strode, D. E. (2021). A theory of coordination: From propositions to hypotheses in Agile software development. *Proceedings of the Annual Hawaii International Conference on System Sciences*, *2020-Janua*, 6795–6805. https://doi.org/10.24251/hicss.2021.815

Khalil, C. a, Fernandez, V. b, & Houy, T. b. (2013). Can agile collaboration practices enhance knowledge creation between cross-functional teams? *Advances in Intelligent Systems and Computing*, *205 AISC*, 123–133. https://doi.org/10.1007/978-3-642-37317-6_11

Khan, S., & VanWynsberghe, R. (2008). Cultivating the under-mined: Cross-case analysis as knowledge mobilization. *Forum: Qualitative Social Research*, *9*(1). https://doi.org/10.17169/fqs-9.1.334

Kitchenham, B., & Charters, S. (2007a). Guidelines for performing Systematic Literature reviews in Software Engineering. *Engineering*, *45*(4ve), 1051. https://doi.org/10.1145/1134285.1134500

Kitchenham, B., & Charters, S. (2007b). Guidelines for performing Systematic Literature reviews in Software Engineering. *Engineering*, *45*(4ve), 1051. https://doi.org/10.1145/1134285.1134500

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M., & Linkman, S. (2010). Systematic literature reviews in software engineering-A tertiary study. In *Information and Software Technology* (Vol. 52, Issue 8, pp. 792–805). Elsevier B.V. https://doi.org/10.1016/j.infsof.2010.03.006

Klein, H. K., & Myers, M. D. (1999). A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, *23*(1), 67. https://doi.org/10.2307/249410

Klimoski, R., & Mohammed, S. (1994). Team Mental Model: Construct or Metaphor? *Journal of Management*, *20*(2), 403–437. https://doi.org/10.1177/014920639402000206

Korpela, M., Mursu, A., & Soriyan, H. A. (2002). Information systems development as an activity. *Computer Supported Cooperative Work*, *11*(1–2), 111–128. https://doi.org/10.1023/A:1015252806306

Kostin, D., & Strode, D. (2022). *Effective communication in globally distributed Scrum teams*.

Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM*, *38*(3), 69–81. https://doi.org/10.1145/203330.203345

Kudaravalli, S., Faraj, S., & Johnson, S. L. (2017). A Configural Approach To Coordinating Expertise in Software Development Teams. *MIS Quarterly*, *41*(1), 43–64. http://proxy-remote.galib.uga.edu/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=bth&AN=121204221&site=eds-live

Kumlander, D. (2010). Semi- and Fully Self-Organised Teams. *Advanced Techniques in Computing Sciences and Software Engineering*, 8–12. https://doi.org/10.1007/978-90-481-3660-5

Laurent, P., Mäder, P., Cleland-Huang, J., & Steele, A. (2010). A taxonomy and visual notation for modeling globally distributed requirements engineering projects. *Proceedings - 5th International Conference on Global Software Engineering, ICGSE 2010*, 35–44. https://doi.org/10.1109/ICGSE.2010.55

Lee, A. S. (1991). Integrating Positivist and Interpretive Approaches to Organizational Research. *Organization Science*, *2*(4), 342–365.

Lee, A. S., & Baskerville, R. L. (2003). Generalizing Generalizability in Information Systems Research. *Information Systems Research*, *14*(3). https://doi.org/10.1287/isre.14.3.221.16560

Lee, G., & Xia, W. (2010). Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility. In *Source: MIS Quarterly* (Vol. 34, Issue 1). https://www.jstor.org/stable/20721416

Levesque, L. L., Wilson, J. M., & Wholey, D. (2001). Cognitive Divergence and Shared Mental Models in Software Development Project Teams. *Journal of Organizational Behavior*, *22*(2), 135–144.

Lewin, K. (1951). *Field theory in social science: selected theoretical papers (Edited by Dorwin Cartwright.).*

Li, Y. a, & Maedche, A. b. (2012). Formulating effective coordination strategies in agile global software development teams. *International Conference on Information Systems, ICIS 2012*, *5*, 4438–4449. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84886530013&partnerID=40&md5=10f840c3009dfca023900594bc393d6f

Lindsjørn, Y., Sjøberg, D. I. K., Dingsøyr, T., Bergersen, G. R., & Dybå, T. (2016). Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*, *122*, 274–286. https://doi.org/10.1016/j.jss.2016.09.028

Mackenzie, A., & Monk, S. (2004). From Cards to Code: How ExtremeProgramming Re-Embodies Programming as aCollective Practice. *Comput. Supported Coop. Work*, *13*(1), 91–117. https://doi.org/10.1023/B:COSU.0000014873.27735.10

Maimbo, H., & Pervan, G. (2005). Designing a Case Study Protocol for application in IS research. *9th Pacific Asia Conference on Information Systems: I.T. and Value Creation, PACIS 2005*, 1281–1292.

Mak, D. K. M., & Kruchten, P. B. (2007). NextMove: A Framework for Distributed Task Coordination. *Software Engineering Conference, 2007. ASWEC 2007. 18th Australian*, 399–408. https://doi.org/10.1109/ASWEC.2007.33

Malone, T. W., & Crowston, K. (1994). The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, *26*(1), 87–119. https://doi.org/http://doi.acm.org/10.1145/174666.174668

Marques-Quinteiro, P., Curral, L., Passos, A., & Lewis, K. (2013). And Now What Do We Do? The Role of Transactive Memory Systems and Task Coordination in Action Teams. *Group Dynamics Theory Research and Practice*, *17*(SEPTEMBER 2013), 194. https://doi.org/10.1037/a0033304

Marthe Berntzen, Viktoria Stray, & Nils Brede Moe. (2021). Coordination Strategies: Managing Inter-team Coordination Challenges in Large-Scale Agile. *Agile Processes in Software Engineering and Extreme Programming, XP 2021*, *1*, 140–156. https://doi.org/10.1007/978-3-030-78098-2

Maruping, L. M., Zhang, X., & Venkatesh, V. (2009). Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems*, *18*(4). https://doi.org/10.1057/ejis.2009.24

Mathieu, J. E., Marks, M. A., & Zaccaro, S. J. (2002). Multiteam systems. In *Handbook of industrial, work and organizational psychology, Volume 2: Organizational psychology*. (pp. 289–313). Sage Publications, Inc.

McGrath, J. E., & Hollingshead, A. B. (1994). *Groups interacting with technology: Ideas, evidence, issues, and an agenda.* Sage Publications, Inc.

McHugh, O., Conboy, K., & Lang, M. (2011). Using Agile Practices to Build Trust in an Agile Team: A Case Study. *Information Systems Development*. https://doi.org/10.1007/978-1-4419-9790-6_40

Menand, L. (1997). *Pragmatism: A reader*. Vintage Press.

Miles, M. B., & Huberman, A. M. (1994a). *Qualitative data analysis: An expanded sourcebook* (3rd, Ed.). Sage Publications.

Miles, M. B., & Huberman, A. M. (1994b). *Qualitative data analysis: An expanded sourcebook* (3rd, Ed.). Sage Publications.

Mintzberg, H. (1989). *Mintzberg on management: Inside our strange world of organizations*. Simon and Schuster.

Mishra, D., Mishra, A., & Ostrovska, S. (2012). Impact of physical ambiance on communication, collaboration and coordination in agile software development: An empirical evaluation. *Information and Software Technology*, *54*(10), 1067–1078. https://doi.org/10.1016/j.infsof.2012.04.002

Misra, S., & Fernández-sanz, L. (2011). Quality Issues in Global Software Development. *ICSEA 2011: The Sixth International Conference on Software Engineering Advances*, *c*, 325–330.

Modi, S., Abbott, P., & Counsell, S. (2013). Raising Awareness In Distributed Agile Development - A Case Study Perspective. *UK Academy for Information Systems Conference Proceedings 2013*, 1–26. http://aisel.aisnet.org/ukais2013/26

Moe, N. B., & Dingsøyr, T. (2017). Emerging Research Themes and updated Research Agenda for Large-Scale Agile Development. *XP2017*, 1–4. https://doi.org/10.1145/3120459.3120474

Moe, N. B., Dingsøyr, T., & Dybå, 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. https://doi.org/10.1016/j.infsof.2009.11.004

Moe, N. B., Dingsøyr, T., & Rolland, K. (2018). To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development. *International Journal of Information Systems and Project Management*, *6*(3), 45–59. https://doi.org/10.12821/ijispm060303

Moe, N. B., Olsson, H. H., & Dingsøyr, T. (2016). Trends in Large-Scale Agile Development: A Summary of the 4th Workshop at XP2016. *Proceedings of the Scientific Workshop Proceedings of XP2016*, 1:1--1:4. https://doi.org/10.1145/2962695.2962696

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. https://doi.org/10.1177/0149206309356804

Moreno, D. B. A. F. M. (2013). Perspectives on Productivity and Delays in Large-Scale Agile Projects. *Agile Processes in Software Engineering and Extreme Programming*, *0*(0). https://doi.org/10.1007/978-3-642-38314-4_13

Morken, R. A. T. (2014a). *Coordination in Large-Scale Agile Development* (Issue May).

Morken, R. A. T. (2014b). *Coordination in Large-Scale Agile Development* (Issue May). Norwegian University of Science and Technology (NTNU).

Myers, M. D. (1997). Qualitative research in information systems. *Management Information Systems Quarterly*, *21*(June), 1–18. https://doi.org/10.2307/249422

Nasroullahi, E., & Tumer, K. (2012). Combining Coordination Mechanisms to Improve the Performance of Multi-Robot Teams. *Artificial Intelligence Research*, *1*(2).

Nguyen-Duc, A., Cruzes, D. S., & Conradi, R. (2015). The impact of global dispersion on coordination, team performance and software quality-A systematic literature review. *Information and Software Technology*, *57*(1), 277–294. https://doi.org/10.1016/j.infsof.2014.06.002

Niederman, F., Lechler, T., & Petit, Y. (2018). A Research Agenda for Extending Agile Practices In Software Development and Additional Task Domains. *Project Management Journal*, *49*(6), 3–17. https://doi.org/10.1177/8756972818802713

NVivo. (2022). https://www.qsrinternational.com/nvivo-qualitative-data-analysis-software/home

Nyrud, H., & Stray, V. (2017). Inter-team coordination mechanisms in large-scale agile. *Proceedings of the XP2017 Scientific Workshops on - XP '17*, *August*, 1–6. https://doi.org/10.1145/3120459.3120476

Okhuysen, G. A., & Bechky, B. A. (2009). Coordination in Organizations: An Integrative Perspective. *The Academy of Management Annals*, *3*(1), 463–502. https://doi.org/10.1080/19416520903047533

Okoli, C., & Schabram, K. (2010). A Guide to Conducting a Systematic Literature Review of Information Systems Research. *Sprouts: Working Papers on Information Systems*, *10*(2010). https://doi.org/10.2139/ssrn.1954824

Orlikowski, W. J., & Baroudi, J. J. (1991). Studying information technology in organizations: Research approaches and assumptions. *Information Systems Research*, *2*(1), 1–28. https://doi.org/10.1287/isre.2.1.1

Owen, J. (2016). *Global teams: How the best teams achieve high performance*. Pearson UK.

Paasivaara, M., & Lassenius, C. (2014). Communities of practice in a large distributed agile software development organization - Case Ericsson. *Information and Software Technology*, *56*(12), 1556–1577. https://doi.org/10.1016/j.infsof.2014.06.008

Paasivaara, M., Lassenius, C., & Heikkilä, V. T. (2012). Inter-team coordination in large-scale globally distributed scrum: Do Scrum-of-Scrums really work? *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 235–238. https://doi.org/10.1145/2372251.2372294

Petersen, K., & Feldt, R. (2008). Systematic mapping studies in software engineering. *12th International Conference on Evaluation and Assessment in Software Engineering*, 1–10. https://www.isl.ce.yildiz.edu.tr/personal/maktas/file/3599/systemticmapping.pdf

Petersen, K., Vakkalanka, S., & Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, *64*, 1–18. https://doi.org/10.1016/j.infsof.2015.03.007

Petersen, K., & Wohlin, C. (2009). Context in industrial software engineering research. *2009 3rd International Symposium on Empirical Software Engineering and Measurement, ESEM 2009*, 401–404. https://doi.org/10.1109/ESEM.2009.5316010

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, *13*(3), 303–337. https://doi.org/10.1007/s10664-008-9065-9

Pries-Heje, L., & Pries-Heje, J. (2011). Agile & Distributed Project Management : a Case Study Revealing Why Scrum Is Useful. *Ecis*, *2011*, Paper 217. https://doi.org/10.1109/AGILE.2011.34

Prikladnicki, R., & Carmel, E. (2013). Is time-zone proximity an advantage for software development? The case of the Brazilian IT industry. *2013 35th International Conference on Software Engineering (ICSE)*, 973–981. https://doi.org/10.1109/ICSE.2013.6606647

Procter, R., Rouncefield, M., Poschen, M., Lin, Y., & Voss, A. (2011). Agile project management: A case study of a Virtual Research Environment development project. *Computer Supported Cooperative Work*, *20*(3), 197–225. https://doi.org/10.1007/s10606-011-9137-z

Rally Software®. (2022). https://www.broadcom.com/products/software/value-stream-management/rally

Ramesh, B., Xu, P., Cao, L. A. N., Mohan, K., Cao, L. A. N., Mohan, K., Xu, P., Cao, L. A. N., Mohan, K., Ramesh, B., Cao, L. A. N., Mohan, K., Xu, P., Cao, L. A. N., & Mohan, K. (2006a). Can distributed software development be agile? *Communications of the ACM*, *49*(10), 41. https://doi.org/10.1145/1164394.1164418

Ramesh, B., Xu, P., Cao, L. A. N., Mohan, K., Cao, L. A. N., Mohan, K., Xu, P., Cao, L. A. N., Mohan, K., Ramesh, B., Cao, L. A. N., Mohan, K., Xu, P., Cao, L. A. N., & Mohan, K. (2006b). Can distributed software development be agile? *Communications of the ACM*, *49*(10), 41. https://doi.org/10.1145/1164394.1164418

Razzak, M. A., & Ahmed, R. (2014). Knowledge sharing in distributed agile projects: Techniques, strategies and challenges. *Computer Science and Information Systems (FedCSIS), 2014 Federated Conference On*, 1431–1440. https://doi.org/10.15439/2014F280

Redmiles, D., Van Der Hoek, A., Al-Ani, B., Hildenbrand, T., Quirk, S., Sarma, A., Filho, R. S. S., De Souza, C., & Trainer, E. (2007). Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects. *Wirtschaftsinformatik*, *49*, 28–38. https://doi.org/10.1038/leu.2008.246

Rico, R., Sánchez-Manzanares, M., Gil, F., & Gibson, C. (2008). Team implicit coordination processes: A team knowledge-based approach. *Academy of Management Review*, *33*(1), 163–184. https://doi.org/10.5465/AMR.2008.27751276

Robey, D., & Markus, M. L. (1998). Beyond Rigor and Relevance: Producing Consumable Research about Information Systems. *Information Resources Management Journal (IRMJ)*, *11*(1), 7–16. https://doi.org/10.4018/irmj.1998010101

Robson, C. (2002). *Real world research: A resource for social scientists and practitioner-researchers* (2nd ed.). Wiley-Blackwell.

Rolland, K. H., Fitzgerald, B., Dingsoyr, T., & Stol, K.-J. (2016). Problematizing Agile in the Large : Alternative Assumptions for Large - Scale Agile Development. *ICIS* -

International Conference on Information Systems, *Ambler 2014*, 1–21. https://doi.org/10.13140/RG.2.2.27795.07207

Rosemann, M., & Vessey, I. (2008). Toward improving the relevance of information systems research to practice: The role of applicability checks. *MIS Quarterly: Management Information Systems*, *32*(1), 7–22. https://doi.org/10.2307/25148826

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. https://doi.org/10.1007/s10664-008-9102-8

Runeson, P., Höst, M., Rainer, A., & Regnell, B. (2012). Case Study Research in Software Engineering: Guidelines and Examples. In *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley. https://doi.org/10.1002/9781118181034

Šablis, A., & Šmite, D. (2016). Agile teams in large-scale distributed context-isolated or connected? *ACM International Conference Proceeding Series*, *24-May-201*. https://doi.org/10.1145/2962695.2962705

Sablis, A., Smite, D., & Moe, N. (2020). Team-external coordination in large-scale software development projects. *Journal of Software: Evolution and Process*. https://doi.org/10.1002/smr.2297

Sablis, A., Smite, D., & Moe, N. (2021). Team-external coordination in large-scale software development projects. *Journal of Software: Evolution and Process*, *33*(3), e2297.

Salaou, A.-D., Damian, D., Lassenius, C., Voda, D., & Gançarski, P. (2020). Archetypes of delay: An analysis of online developer conversations on delayed work items in IBM Jazz. *Information and Software Technology*, *September*, 106435. https://doi.org/10.1016/j.infsof.2020.106435

Sangwan, R. S., & LaPlante, P. A. (2006). Test-Driven Development in Large Projects. *IT Professional*, *8*(5), 25–29. https://doi.org/10.1109/MITP.2006.122

Saxena, A., Venkatagiri, S., & Bandi, R. K. (2016). Managing Inherent Conflicts in Agile Distributed Development : Evidence from Product Development. *Australasian Conference on Information Systems*, 10. http://ro.uow.edu.au/cgi/viewcontent.cgi?article=1067&context=acis2016

Scheerer, A., Hildenbrand, T., & Kude, T. (2014). Coordination in Large-Scale Agile Software Development: A Multiteam Systems Perspective. *2014 47th Hawaii International Conference on System Sciences*, 4780–4788. https://doi.org/10.1109/HICSS.2014.587

Scheerer, A., & Kude, T. (2014). Exploring coordination in large-scale agile software development: A multiteam systems perspective. *47th Hawaii International Conference on System Science*. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84923478040&partnerID=40&md5=b861aad53ef36d5c915a19b68b40cee4

Schmidt, C. (2016). Agile Software Development. *Agile Software Development Teams*, *0*(0). https://doi.org/10.1007/978-3-319-26057-0_2

Schuemmer, T., & Lukosch, S. (2009). Understanding Tools and Practices for Distributed Pair Programming. *Journal of Universal Computer Science*, *15*(16), 3101–3125.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (Vol. 1). Prentice Hall Upper Saddle River.

Schwaber, K., & Sutherland, J. (2020). *The Scrum Guide The Definitive Guide to Scrum: The Rules of the Game*.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering*, *25*(4), 557–572. https://doi.org/10.1109/32.799955

Sekitoleko, N., Evbota, F., Knauss, E., Sandberg, A., Chaudron, M., & Olsson, H. H. ̈om. (2014). Technical Dependency Challenges in Large-Scale Agile Software Development. *Agile Processes in Software Engineering and Extreme Programming*, *0*(0). https://doi.org/10.1007/978-3-319-06862-6_4

Sharp, H., Dittrich, Y., & De Souza, C. R. B. (2016). The role of ethnographic studies in empirical software engineering. *IEEE Transactions on Software Engineering*, *42*(8), 786–804.

Sharp, H., & Robinson, H. (2004). An ethnographic study of XP practice. *Empirical Software Engineering*, *9*(4), 353–375. https://doi.org/10.1023/B:EMSE.0000039884.79385.54

Sharp, H., & Robinson, H. (2008). Collaboration and co-ordination in mature eXtreme programming teams. *International Journal of Human Computer Studies*, *66*(7), 506–518. https://doi.org/10.1016/j.ijhcs.2007.10.004

Shrivastava, S. v, & Rathod, U. (2015). Categorization of risk factors for distributed agile projects. In *Information and Software Technology* (Vol. 58, pp. 373–387). https://doi.org/10.1016/j.infsof.2014.07.007

Sierra, J. M., Vizcaíno, A., Genero, M., & Piattini, M. (2018). A systematic mapping study about socio-technical congruence. *Information and Software Technology*, *94*, 111–129. https://doi.org/10.1016/j.infsof.2017.10.004

Smite, D., & Dingsøyr, T. (2012). Fostering cross-site coordination through awareness: An investigation of state-of-the-practice through a focus group study. *Proceedings - 38th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2012*, 337–344. https://doi.org/10.1109/SEAA.2012.24

Stake, R. E. (1995). *The art of case study research*. Sage, Thousand Oaks, CA.

Steinfield, C., Jang, C.-Y., & Pfaff, B. (1999). Supporting Virtual Team Collaboration: The Team Scope System. *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work*, 81–90. https://doi.org/10.1145/320297.320306

Steinmacher, I., Chaves, A. P., & Gerosa, M. A. (2013). Awareness support in distributed software development: A systematic review and mapping of the literature. *Computer Supported Cooperative Work*, *22*(2–3), 113–158. https://doi.org/10.1007/s10606-012-9164-4

Strauss, A. (1988). The Articulation of Project Work: An Organizational Process. *Source: The Sociological Quarterly*, *29*(2), 163–178. http://www.jstor.org/stable/4121474

Stray, V., Hoda, R., Paasivaara, M., Lenarduzzi, V., & Mendez, D. (2022). Theories in Agile Software Development: Past, Present, and Future Introduction to the XP

2020 Special Section. *Information and Software Technology*, 107058. https://doi.org/10.1016/j.infsof.2022.107058

Stray, V., & Moe, N. B. (2020). Understanding coordination in global software engineering: A mixed-methods study on the use of meetings and Slack. *The Journal OfSystems & Software*, *170*, 109231. https://doi.org/10.1016/j.jss.2020.110717

Strode, D. (2012). A Theory of Coordination in Agile Software Development Projects. *Researcharchive.Vuw.Ac.Nz*. http://researcharchive.vuw.ac.nz/bitstream/handle/10063/2505/thesis.pdf?sequence=1

Strode, D. E. (2016). A dependency taxonomy for agile software development projects. *Information Systems Frontiers*, *18*(1). https://doi.org/10.1007/s10796-015-9574-1

Strode, D. E., Hope, B. G., Huff, S. L., & Link, S. (2011). Coordination Effectiveness In An Agile Software Development Context. *Proceedings of the Pacific Asia Conference on Information Systems (PACIS), Brisbane, Australia, 7-11 July, 2011*, Paper 183. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.231.9750&rep=rep1&type=pdf

Strode, D. E., Huff, S. L., Hope, B., & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*, *85*(6), 1222–1238. https://doi.org/10.1016/j.jss.2012.02.017

Strode, D., & Huff, S. L. (2015). A coordination perspective on agile software development. In *Modern Techniques for Successful IT Project Management*. https://doi.org/10.4018/978-1-4666-7473-8.ch004

Sureshchandra, K., & Shrinivasavadhani, J. (2008). Adopting agile in distributed development. *Proceedings - 2008 3rd IEEE International Conference Global Software Engineering, ICGSE 2008*, 217–221. https://doi.org/10.1109/ICGSE.2008.25

Talby, D., Keren, A., Hazzan, O., & Dubinsky, Y. (2006). Agile software testing in a large-scale project. *IEEE Software*, *23*(4), 30–37. https://doi.org/10.1109/ms.2006.93

Talukder, A. B. M. N. A., Senapathi, M., & Buchan, J. (2017). Coordination in Distributed Agile Software Development : A Systematic Review. *28th Australasian Conference on Information Systems (ACIS 2017)*, 1–12.

Tell, P., & Babar, M. A. (2012). Activity theory applied to global software engineering: Theoretical foundations and implications for tool builders. *Proceedings - 2012 IEEE 7th International Conference on Global Software Engineering, ICGSE 2012*, *2011*(April 2011), 21–30. https://doi.org/10.1109/ICGSE.2012.24

Temitayo, V., Badru, A., & Ajayi, N. (2017). Adopting Scrum as an Agile Approach in Distributed Software Development: A Review of Literature. *Ieee*, *1*(2), 1–5.

Thompson, J. D. (1967). Organizations In Action: Social Science Bases of Administrative Theory. *McGraw-Hill*. https://doi.org/10.1007/s13398-014-0173-7.2

Vallon, R., da Silva Estácio, B. J., Prikladnicki, R., & Grechenig, T. (2018). Systematic literature review on agile practices in global software development. *Information*

and Software Technology, *96*, 161–180. https://doi.org/10.1016/j.infsof.2017.12.004

Van De Ven, A. H., Delbecq, A. L., & Koenig Jr., R. (1976). Determinants of Coordination Modes within Organizations. *American Sociological Review*, *41*(2), 322–338. https://doi.org/10.2307/2094477

van de Ven, A. H., Delbecq, A. L., & Koenig Jr., R. (1976). Determinants of Coordination Modes within Organizations Author (s): Andrew H . Van De Ven , Andre L . Delbecq and Richard Koenig , Jr . *American Sociological Review*, *41*(2), 322–338. https://doi.org/10.2307/2094477

Verner, J. M., Brereton, O. P., Kitchenham, B. A., Turner, M., & Niazi, M. (2014). Risks and risk mitigation in global software development: A tertiary study. *Information and Software Technology*, *56*(1), 54–78. https://doi.org/10.1016/j.infsof.2013.06.005

VersionOne. (2022). https://www.collab.net/products/versionone

Viktoria Stray, Nils Brede Moe, Diane Strode, & Emilie Maehlum. (2022, May). *Coordination Value in Agile Software Development*.

Wagenaar, G., Helms, R., Damian, D., & Brinkkemper, S. (2015). Artefacts in Agile Software Development. *PROFES 2015*, 133–148. https://doi.org/10.1007/978-3-319-26844-6_10

Wagner, C. (2006). Breaking the Knowledge Acquisition Bottleneck Through Conversational Knowledge Management. *Information Resources Management Journal*, *19*(1), 70–83. http://www.idea-group.com

Wegner, D. M. (1987). Transactive Memory: A Contemporary Analysis of the Group Mind. In *Theories of Group Behavior* (pp. 185–208). https://doi.org/10.1007/978-1-4612-4634-3_9

Weick, K., & Roberts, K. (1993). Collective mind in organizations: Heedful interrelating on flight decks. *Administrative Science Quarterly*, *38*(3), 357–381. https://doi.org/10.2307/2393372

Wilson, R. D., & Creswell, J. W. (1996). Research Design: Qualitative and Quantitative Approaches. In *Journal of Marketing Research* (Vol. 33, Issue 2). Sage publications. https://doi.org/10.2307/3152153

Wiredu, G. O. (2006). *A Framework for the Analysis of Coordination in Global Software Development*.

Wohlin, C., & Petersen, K. (2010). The effect of moving from a plan-driven to an incremental software development approach with agile practices. *Empirical Software Engineering*, *15*(6). https://doi.org/10.1007/s10664-010-9136-6

XPlanner. (2022). https://www.openhub.net/p/xplanner

Xu, P. (2009). Coordination In Large Agile Projects. *The Review of Business Information Systems*, *13*(4), 29–43. http://search.proquest.com/docview/194692352?accountid=14719%5Cnhttp://openurl.uquebec.ca:9003/uqam?url_ver=Z39.88-2004&rft_val_fmt=info:ofi/fmt:kev:mtx:journal&genre=article&sid=ProQ:ProQ:abiglobal&atitle=Coordination+In+Large+Agile+Projects&title=The+Re

Xu, P., & Cao, L. (2006). Coordination in Agile Software Projects. *AMCIS 2006 Proceedings*, 3680–3685.

Yadav, V., Adya, M., Nath, D., & Sridhar, V. (2007). Investigating an “Agile-Rigid” Approach in Globally Distributed Requirements Analysis. *11th Pacific-Asia Conference on Information Systems*, *January*. http://aisel.aisnet.org/pacis2007%0Ahttp://aisel.aisnet.org/pacis2007/12

Yin, R. K. (2002). Case Study Reserach: Design and Methods. In *SAGE Publications* (2nd ed.). Sage Publications. https://doi.org/10.1016/j.jada.2010.09.005

Yin, R. K. (2009). Case study research : design and methods. In *Applied social research methods series ; Vol. 5.* (pp. xiv, 219 p.). https://doi.org/10.1097/FCH.0b013e31822dda9e

Yin, R. K. (2018). Case study research and applications : design and methods. In *SAGE Publications* (6th ed.). SAGE.

Young, C., & Terashima, H. (2008). How Did We Adapt Agile Processes to Our Distributed Development? *Agile Conference (AGILE), 2008*, 304–309. http://i-proving.com/wp-content/uploads/2008/04/How-Did-We-Adapt-Agile-Processes-to-Our-Distributed-Development2.pdf

Yu, X., & Petter, S. (2014). Understanding agile software development practices using shared mental models theory. *Information and Software Technology*, *56*(8), 911–921. https://doi.org/10.1016/j.infsof.2014.02.010

Zaitsev, A., Gal, U., & Tan, B. (2020). Coordination artifacts in Agile Software Development. *Information and Organization*, *30*(2), 100288. https://doi.org/10.1016/j.infoandorg.2020.100288

Zuill, W., Programming, I. M., Programming, W. M., Owner, P., Programming, E., Programming, M., & Meadows, K. (2016). Mob programming: A whole team approach. *Agile 2014 Conference, Orlando, Florida*, *3*.

Appendix A: Ethics Approval

AUTEC Secretariat

Auckland University of Technology
D-88, WU406 Level 4 WU Building City Campus
T: +64 9 921 9999 ext. 8316
E: ethics@aut.ac.nz
www.aut.ac.nz/researchethics

9 May 2018

Mali Senapathi
Faculty of Design and Creative Technologies

Dear Mali

Re Ethics Application: **18/156 Understanding coordination in distributed agile software development**

Thank you for providing evidence as requested, which satisfies the points raised by the Auckland University of Technology Ethics Committee (AUTEC).

Your ethics application has been approved for three years until 8 May 2021.

Standard Conditions of Approval

1. A progress report is due annually on the anniversary of the approval date, using form EA2, which is available online through http://www.aut.ac.nz/researchethics.
2. A final report is due at the expiration of the approval period, or, upon completion of project, using form EA3, which is available online through http://www.aut.ac.nz/researchethics.
3. Any amendments to the project must be approved by AUTEC prior to being implemented. Amendments can be requested using the EA2 form: http://www.aut.ac.nz/researchethics.
4. Any serious or unexpected adverse events must be reported to AUTEC Secretariat as a matter of priority.
5. Any unforeseen events that might affect continued ethical acceptability of the project should also be reported to the AUTEC Secretariat as a matter of priority.

Please quote the application number and title on all future correspondence related to this project.

AUTEC grants ethical approval only. If you require management approval for access for your research from another institution or organisation then you are responsible for obtaining it. If the research is undertaken outside New Zealand, you need to meet all locality legal and ethical obligations and requirements. You are reminded that it is your responsibility to ensure that the spelling and grammar of documents being provided to participants or external organisations is of a high standard.

For any enquiries, please contact ethics@aut.ac.nz

Yours sincerely,

Kate O'Connor
Executive Manager
Auckland University of Technology Ethics Committee

Cc: a.talukder@aut.ac.nz; jbuchan@aut.ac.nz

Appendix B

B.1 Participant Information Sheet

Date Information Sheet Produced: 10 April 2018

Project Title: Understanding Coordination in Distributed Agile Software Development

Research Introduction

An Invitation

My name is A B M Nurul Afser Talukder and I am a doctoral student in School of Engineering, Computer and Mathematical Sciences at Auckland University of Technology, under the supervision of Dr Mali Senapathi and Jim Buchan. I am conducting research investigating coordination process within the distributed agile software development teams.

Please note that your participation in this research is voluntary in nature, and you may decline or withdraw your participation without any adverse consequences. None of the participants are identified nor will the information gathered be used to hamper, hinder or harm your career.

The following questions and answers are intended to address the most common questions that the participant may ask about this particular research project. If you need further information, feel free to contact the researcher, Jennifer Yang. My contact details can be found at the end of this document. It is recommended that you use e-mail to reach me.

Project Description and invitation

This research aims to understand the current state of coordination practices for Distributed Agile Software Development (DASD) teams, to gain some insights into the activities, strategies, expectation and. This will result in a clearer picture of practices and roles regarding coordination and recommendations to address any gaps and challenges, and assisting practitioners with decision making and providing a structure for further research in this area.

I would like to invite you to participate in my research into the area of coordination in Distributed Agile Software development. Please note that your participation in this research is voluntary in nature, and you may decline or withdraw your participation without any adverse consequences. None of the participants is identified nor will the information gathered be used to hamper, hinder or harm your career.

How was I identified and why am I being invited to participate in this research?

You have been identified from the research Supervisor's personal network of industry practitioners as someone with expertise in the area of team-based distributed agile software development.

How do I agree to participate in this research?

To follow up on this invitation to participate in this research, please confirm your acceptance by email. You will also reconfirm your consent to participate in the interview formally by signing the Participant's Consent Form which we will give you just prior to your interview.

Your participation in this research is voluntary (it is your choice) and whether or not you choose to participate will neither advantage nor disadvantage you. You are able to withdraw from the study at any time. If you choose to withdraw from the study, then you will be offered the choice between having any data that is identifiable as belonging to you removed or allowing it to continue to be used. However, once the findings have been produced, removal of your data may not be possible.

What will happen in this research?

If you accept this invitation to participate, you will be interviewed by the researcher. This will be a loosely structured interview where you will be asked some open-ended questions related to your project, your role in that project and your experience of coordination practices in within your team and other distributed teams, as well as challenges you have encountered. The researcher will participate some of the events of day-to-day coordination activities where you might be present and during this period, the researcher will observe and take notes according to the points of research interest.

The interviews will be held at your usual work place(s) or any neutral place if requested. The researcher will take some notes for later analysis and also record the interview as a memory aid. The analysis will involve coding the data to identify trends and themes that provide insights to practitioners' perceptions of Distributed Agile team composition and leadership. Note that it is anticipated that the recording of the interview will not be transcribed in full, but quotes may be extracted as evidence of patterns identified. The data will have all references to the organisation and individuals removed for analysis.

At the end of this research a report summarising the main results will be made available to you if requested on the Consent Form. Furthermore, it is expected that papers may be published in academic journals reporting the main conceptual findings of this research project.

What are the discomforts and risks?

During the interview sessions, there is a possibility you may feel uncomfortable about sharing your point of view about the project operations.

Additionally, during the observation sessions, you may feel uncomfortable and unwilling to share any information related to business and work process secret or share your personal feelings in front of other participants.

You may feel uncomfortable that your line manager will know who is participating in the study and who has elected not to take up the invitation, and that this could affect their perception of you, and future prospects.

You may feel uncomfortable about having your interview recorded.

You may feel uncomfortable that your colleagues or line managers may overhear what you say during the interview, and that this could negatively affect their perception of you.

How will these discomforts and risks be alleviated?

In order to alleviate the first area of possible discomfort, you will be reminded of our assurance of confidentiality of all interview data at the start of the interview process. You may choose not to answer specific questions, and you can also withdraw from participating in the interview at any stage. You can also request that your interview data be withdrawn from the study before the completion of data collection.

The second possible area of discomfort will be addressed by stressing the voluntary nature of participation to both you and your company. We understand the time pressures faced by you as an employee, and recognise that it is not always feasible or practical to participate in such studies. While your line manager will know you have been approached, participation or non-participation will not be specifically recorded or communicated apart from the need to organise a specific time and date for your interview. During the observation sessions, the research will respect the company and its member's privacy and will be ready to leave the meeting place or the location site at any time if asked by the participants.

Recording of the interview is not a prerequisite of conducting the interview. Before the interview begins you will be asked for permission to record the interview. Even if consent to record is provided, you will be reminded that you can request that the recording be stopped or wiped at any stage of the interview.

A soundproof room will be requested for the interviews at the company premises, or, at your request, the interview will be conducted at a neutral place away from work. This obviates the risk of being overhead.

What are the benefits?

As well as adding to the body of knowledge and influencing practice in this general area, the insights gained from this study will be made available to yourself and your colleagues and it is hoped that the knowledge gained will be useful for improving the practice in your organization.

How will my privacy be protected?

All of the materials related to the participants' information (consent form, tape, and interview notes) will be stored at AUT in a locked cupboard for at least 6 years. After that the material will be destroyed.

It is not anticipated that a transcriber will be involved transcribing the recorded interview. The researcher may transcribe small parts of the recorded material to use as exemplars and evidence of trends and claims resulting from the analysis.

The data from the interviews will be anonymised and analysed for principles and insights that are independent of the interviewee's identity. Furthermore, demographic data will be coded, and the data stored in a separate place so that the identity of each participant will be separated from their responses.

If participants decide to withdraw from this research project for any reason before the completion of data collection, all of the materials relating to their interview will be destroyed as soon as practicable after their request.

In addition, your line manager will not hear or see the content of this research data. The only people who will have access to your data will be the researcher and the researcher's supervisors.

What are the costs of participating in this research?

Time is the only cost to you. The interview will take around one hour of your time.

What opportunity do I have to consider this invitation?

Due to time restrictions in undertaking the fieldwork for the research, we would ideally like to have notice of your agreement within a week of you receiving this invitation.

Will I receive feedback on the results of this research?

If you would like a report summarising the results of this research, please tick the appropriate box on the Consent Form, provided at the interview.

What do I do if I have concerns about this research?

Any concerns regarding the nature of this project should be notified in the first instance to the Research Supervisor, Mali Senapathi; mali.senapathi@aut.ac.nz; 09 921 9999 ext. 5213.

Concerns regarding the conduct of the research should be notified to the Executive Secretary of AUTEC, Kate O'Connor, ethics@aut.ac.nz , 09 921 9999 ext. 6038.

Whom do I contact for further information about this research?

Please keep this Information Sheet and a copy of the Consent Form for your future reference. You are also able to contact the research team as follows:

Researcher Contact Details:

A B M Nurul Afser Talukder

Software Engineering Research Lab (SERL)

School of Engineering, Computer and Mathematical Sciences

Auckland University of Technology

Private Bag 92006

Auckland 1142

New Zealand

Phone: + 64 9 921 9999 x 6376

Email: a.talukder@aut.ac.nz

Project Supervisor Contact Details:

Mali Senapathi

Senior Lecturer / Programme Leader, Master of Service-Oriented Computing

School of Engineering, Computer and Mathematical Sciences

Auckland University of Technology

Private Bag 92006

Auckland 1142

New Zealand

Phone: + 64 9 921 9999 x 5213

Email: mali.senapathi@aut.ac.nz

B.2 Consent Form – Individual Participant

Project title: ***Understanding coordination in Distributed Agile Software Development teams***

Project Supervisor: ***Mali Senapathi***

Researcher: ***A B M Nurul Afser Talukder***

- ○ I have read and understood the information provided about this research project in the Information Sheet dated 10 April 2018.
- ○ I have had an opportunity to ask questions and to have them answered.
- ○ I understand that notes will be taken during the interviews and that they will also be audio-taped and transcribed.
- ○ I understand that I may withdraw myself or any information that I have provided for this project at any time prior to completion of data collection, without being disadvantaged in any way.
- ○ If I withdraw, I understand that all relevant information including tapes and transcripts, or parts thereof, will be destroyed.
- ○ I agree to take part in this research.
- ○ I wish to receive a summary of the research findings (please tick one):

  Yes ○ No ○

Participant Signature:

.....

Participant Name:

.....

Participant Contact Details (if appropriate):

……...

……...

……...

……...

Date:

B.3 Consent Form – Organisation

Project title: ***Understanding coordination in Distributed Agile Software Development teams***

Project Supervisor: ***Mali Senapathi***

Researcher: ***A B M Nurul Afser Talukder***

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 his research in the following organisation.

Name of Organisation _____

I have had an opportunity to ask questions and to have them answered.

○ I agree that all people involved in the project nominated for this study can choose to participate without coercion.

Name: .....……

Signature .....……

Role ……

Date ……

B.4 Interview Protocol (Team member)

Project title: **Understanding coordination in Distributed Agile Software Development teams**

Project Supervisor: **Mali Senapathi**

Researcher: **A B M Nurul Afser Talukder**

This protocol is designed to take interviews with any particular team member from local or distributed site

---

1. Collect the questionnaire on software development practices
2. Background information
    - What is your Name?
    - What is your role in this project?
    - How many years of IT experience do you have?
    - What is your educational background?
    - What is your previous experience with agile development?
    - How much prior working experience did you have with your local peers on this work?
    - How much prior working experience did you have with distributed site peers on this work?
3. Briefly describe the project
    - What portion of this work was/is done across sites (relative to co-located)?
    - What are the main resources (products/partial products/artifacts) used?
    - How work is allocated among team members both locally and distributed sites?
4. Dependency
    - What are some typical dependencies in this project?
    - Whom did you need to communicate to do your work (a) locally and (b) at distributed sites?
        - Are you aware of their expertise, their availability to fulfil the dependency?
        - Does this awareness affect the overall coordination process?

- What types of information/knowledge do you need to exchange with them (a) locally and (b) at distributed sites?
- What are the roles
- What meetings including agile meetings are used to coordinate those dependencies?
- What technologies are used to coordinate those dependencies?

5. Challenges
    - What do you think makes this project a well-coordinated project?
        - Can you provide an example from this project?
    - What are the challenges did you face while communicating or exchanging information/knowledge (a) locally and (b) at distributed sites?
        - Types of problems, sources of problem, reasons for occurrence
    - How are these problems addressed, or how could they be addressed effectively (a) locally and (b) at distributed sites?
    - What practices/activities, agile or not, do you think are particularly useful for coordinating both locally and distributed sites?
        - Aim for explicit examples/evidence
6. Recommendation
    - What one thing would you change to make coordination better?
        - Any practice, technology, or activity?
    - Anything else to add?

B.5 Interview Protocol (Project Manager / Team Leader)

Project title: **Understanding coordination in Distributed Agile Software Development teams**

Project Supervisor: **Mali Senapathi**

Researcher: **A B M Nurul Afser Talukder**

This protocol is designed to take interviews with project manager/team leader

---

1. Collect the questionnaire on project profile and software development practices
2. Background information
    - What is your Name?
    - What is your role in this project?
    - How many years of IT experience do you have?
    - What is your educational background?
    - What are your previous experience with agile development?
3. Briefly describe the project.
    - Purpose
    - History
    - Completion status
    - How it is structured (team members and their roles)
    - How many members are collocated and how many members are distributed?
    - Length of time working together
    - How much expert in using agile methods?
    - What are the methods that has already been used? For example, Scrum, Kanban or Hybrid.
    - How/why, did you tailor agile methods?
4. Project profile and process
    - Review questionnaire of project profile details and list agile practices here
    - Overall project process (initiation to conclusion)
        - Consider the iteration length and work activities
        - Consider the daily work activities
        - Any other activities?

- Who is involved (stakeholders)?
- What portion of this work was/is done across sites (relative to co-located)?
- What are the main resources (products/partial products/artifacts) used?
- How work is allocated among team members both locally and distributed sites?

5. Dependency
    - What are some typical dependencies in this project?
    - Whom did you need to communicate to do your work (a) locally and (b) at distributed sites?
        - Are you aware of their expertise, their availability to fulfil the dependency?
        - Does this awareness affect the overall coordination process?
    - What types of information/knowledge do you need to exchange with them (a) locally and (b) at distributed sites?
    - What are the roles
    - What meetings including agile meetings are used to coordinate those dependencies?
    - What technologies are used to coordinate those dependencies?
6. Challenges
    - What do you think makes this project a well-coordinated project?
        - Can you provide an example from this project?
    - What are the challenges did you face while communicating or exchanging information/knowledge (a) locally and (b) at distributed sites?
        - Types of problems, sources of problem, reasons for occurrence
    - How are these problems addressed, or how could they be addressed effectively (a) locally and (b) at distributed sites?
    - What practices/activities, agile or not, do you think are particularly useful for coordinating both locally and distributed sites?
        - Aim for explicit examples/evidence
7. Recommendation
    - What one thing would you change to make coordination better?
        - Any practice, technology, or activity?
    - Anything else to add?

Appendix C: Detailed View of the Model of Coordination for DASD projects