Towards a Broader Understanding of Coordination in Software Engineering: A Case Study of a Software Development Team

by

Lucas David Greaves Panjer
B.Sc., University of Western Ontario, 2003

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Towards a Broader Understanding of Coordination in Software Engineering: A Case Study of a Software Development Team

by

Lucas David Greaves Panjer
B.Sc., University of Western Ontario, 2003

SUPERVISORY COMMITTEE

Dr. Daniela Damian, (Department of Computer Science)
Supervisor

Dr. Margaret-Anne Storey, (Department of Computer Science)
Supervisor

Dr. Janice Singer, (Department of Computer Science)
Departmental Member

Supervisory Committee

Dr. Daniela Damian, (Department of Computer Science)
Supervisor

Dr. Margaret-Anne Storey, (Department of Computer Science)
Supervisor

Dr. Janice Singer, (Department of Computer Science)
Departmental Member

ABSTRACT

Coordination of people, processes, and artifacts is a significant challenge to successful software engineering that is growing as the scale, distribution, and complexity of software projects grow. This thesis presents an exploratory case study of coordination of interdependent work in a practicing software development team. Qualitative analysis of stakeholder interviews was used to develop nine theoretical propositions that describe coordination behaviours. One proposition was refined by quantitatively exploring the structure of explicit dependencies between work items in relation to their resolution times. Structure measures drawn from social network analysis were used to quantify the structure of explicit dependencies between work items, revealing some lower resolution times were associated with degree centrality measures, but that network structures only explain a small proportion of the variance in resolution times. The results are compared with existing theories of coordination in software engineering and directions for further research are outlined.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

Completing this thesis required support and contributions from many people. Foremost, I would like to thank my supervisors, Margaret-Anne Storey and Daniela Damian, for their ongoing collaboration, contributions, and insight into my research and development. I am grateful for Janice Singer's input and guidance as a committee member and for her advice during the development of interview tools. I am privileged to have had the community, and constructive criticism, of my peers in the SEGAL and CHISEL research groups. I am grateful for the willingness to participate, and the contributions of time and energy, of interview participants at IBM. Additionally, I thank Marcellus Mindel, Jean-Michel Lemieux, and Harold Ossher of IBM, who in a variety of ways, enabled data collection and a setting for this research. Finally, I thank my family and friends for their support and encouragement of my ongoing inquiry and learning.

CHAPTER 1:
INTRODUCTION

1.1 Introduction

This thesis presents a study of coordination of interdependent work in software engineering. We study coordination because large software projects have high complexity, in part, due to interdependencies between components, people, and processes. Coordination is the phenomenon that manages and resolves these interdependencies. Our research goal is to further our understanding of coordination from a field investigation of coordination, generating theoretical propositions from a case study of a practicing software development team. These propositions can be used to extend or augment existing theory of coordination in software engineering, and serve as a basis for future investigation.

To achieve our research goal we used a sequential exploratory mixed methods case study approach, studying a practicing software development team. Stakeholder interviews and a qualitative thematic analysis were used to generate nine theoretical propositions based on the views and experiences of software developers. These propositions describe strategies and effects of coordination of interdependent work in software engineering. One of these propositions, relating the structure of explicit work item dependencies to resolution time, was further refined using a quantitative statistical analysis of work items and explicit work item dependencies. Network structure measures drawn from social network analysis were used to quantify the structure of explicit dependencies between work items and discern their relationship to resolution times. Analysis shows that the degree centrality of work items in a dependency network are associated with some reduced

resolution time, but that network structures overall can only account for a small proportion of the variance in resolution time.

1.2 Motivation

Software engineering is a complex and difficult activity, becoming especially difficult as the size and complexity of projects require many contributors. The difficulties lie in management of both the cognitive load of developing and designing software, as well as the management of dependencies during implementation and maintenance. Dependencies arise from the need to integrate a broad range of software development activities and artifacts. These activities span requirements specification, architecting solutions, planning implementation, to documenting, testing, configuring, and writing of code. Maintaining awareness of, and managing these dependencies are some of the primary activities of participants in software development, and are some of the primary causes of system failure, unfulfilled requirements, and schedule slip.

Managing the complexity of dependencies requires maintaining awareness of dependencies, their changes, and coordination of involved participants to effectively implement requirements and meet project goals. In the context of software engineering, the act of coordination "is the management of dependencies between activities" (Malone and Crowston 1994) and it "provides the only possibility that separate task groups will be able to consolidate their efforts into a unified system design" (Conway 1968). It is this activity, of coordination, upon which we focus this research study.

When software projects become large, there is a need to distribute development and engineering activities across many participants and stakeholders in the

development process. The range of stakeholders and roles may span from customers and analysts who elicit and develop requirements, to documentation, configuration, and testing specialists, to programmers, project leaders, and business managers. All of these stakeholders work together to produce respective pieces of a software system deliverable, and through their work, each is producing or modifying artifacts that relate to, describe, or are an executable portion of the software system. In order to operate correctly, and to maintain consistency, many of these artifacts have dependencies upon other artifacts. These can be dependencies in executable code that will cause system failure when not met, or can be synchronization needs, such as between a requirement and its implementation, or between documentation and the executable product.

Since the size, complexity, and geographic distribution of software development projects continue to increase, the software engineering research and industrial communities have continually developed new techniques and tools to manage and reduce associated complexities. These tools and techniques, such as continuous build, modelling techniques, or collaboration tools create new types of processes and artifacts that become part of the engineering process. The increasing number of participants, stakeholders, processes, and artifacts will always involve the creation and maintenance of dependencies and will exacerbate coordination difficulties. Management of these interdependencies requires increasingly effective coordination mechanisms, processes, and tools.

1.3 Research Problem

The research problem is that we have a limited understanding of coordination in today's software engineering environment, and that the practice of software

engineering is becoming increasingly complex, involving more people, increased distribution, and development of larger systems. We need an understanding of coordination of interdependent work in software engineering to enable reasoning about, and empirical confirmation of coordination as a phenomenon. From this understanding, theories of coordination in software engineering can be developed that provide a shared model and vocabulary for research and industry. Since dependencies exist between many artifacts, participants, and processes in the software development process, and coordination is at the core of resolving and resolving these dependencies, it is important to understand how coordination functions so that tools, processes, and practices that enable and improve coordination can be developed.

1.4 Research Goal

The goal of this research is to further our understanding of coordination from a case study of coordination in a practicing software development team, developing theoretical propositions that describe the methods and effects of coordination by participants in the software engineering process. These insights and propositions can be used to add new perspectives to existing theories or to form the basis of a new theory of coordination in software engineering. To meet this research goal we seek to answer the question, “How do software developers coordinate interdependent work?” Specifically, what practices and tools are used by software developers to manage interdependent work? and what do these practices achieve? We seek to derive theoretical propositions describing coordination in software engineering from the experience and wisdom of practicing software developers and to refine and empirically test one of these propositions by analyzing work

management artifacts that are created throughout the software development process. This approach anchors the theoretical propositions in real-life practices and techniques and will provide a description of software engineering coordination that reflects current practice and effects.

1.5 Research Approach

This research to study coordination was conducted using a case study approach with mixed methods in data collection and analysis. In this study we interacted with a software development team, conducting qualitative interviews, and accessed a work item repository from the same project to conduct a further quantitative analysis. We use both qualitative and quantitative analysis methods to develop and then refine theoretical propositions about coordination in software engineering.

The sequence and flow of data and findings across the methods is illustrated in Figure 1. The sequential mixed methods approach is an exploratory configuration of methods. The first stage is the collection of field data through interviews with stakeholders and qualitative analysis of the transcripts. The goal and output of this stage is a set of theoretical propositions describing relationships between constructs, that describe coordination. In the second stage of the research, one of the propositions generated through the qualitative analysis is quantitatively explored and refined using data extracted from the team's work item repository. Statistical testing allows a quantitative measurement of the agreement between the qualitative and quantitative research findings, and provides insight into their differences. By exploring the phenomenon of coordination of interdependent work through multiple research methods and approaches we are able to gain greater insight into coordination in software engineering.

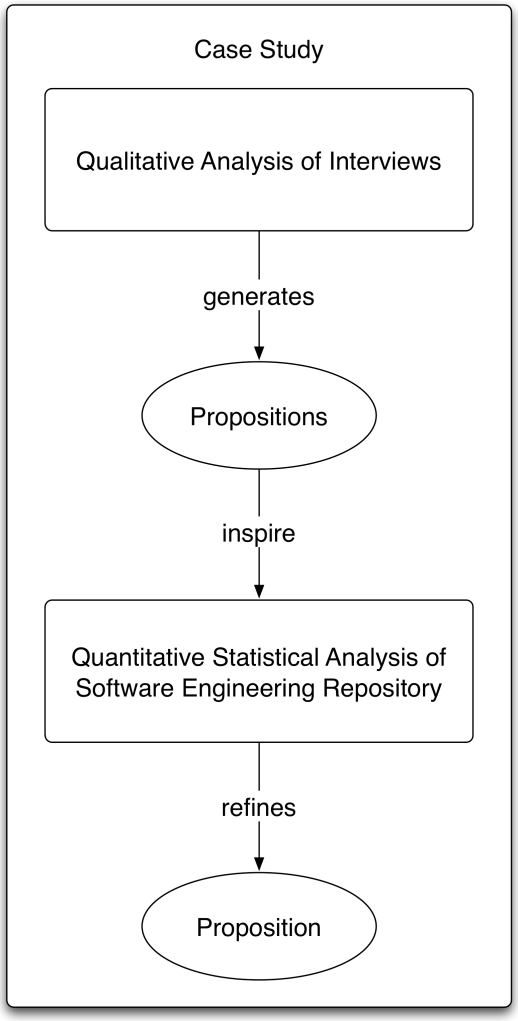

Figure 1: Research approach

In the context of this case study, work items are defined as artifacts that are stored in a shared repository that describe and define defects, enhancements, or tasks within the software project. Each work item has a variety of fields and supports a comment thread, connections to source code changes, and relationships (such as "Depends On") to other work items. Depends on relationships are used to represent dependencies between work items as explicitly expressed by contributors. These dependencies are not necessarily source code or technical dependencies, but could represent scheduling or other coordination needs.

1.6 Contributions

This thesis makes several contributions to knowledge of coordination in software engineering. These contributions are as follows: First, a set of theoretical propositions defining expected effects of coordination behaviours in software engineering derived from analysis of interviews with practicing software developers. Second, an increased understanding of the relationship between the structure of explicit work item dependencies and resolution time through further refinement of one proposition using a quantitative statistical analysis. Finally, many avenues for future work are identified and are presented as potential research approaches for further developing and refining the theoretical propositions developed in the qualitative analysis of interviews. The remainder of the opportunities for future work are based on insights developed across both the qualitative and quantitative analyses conducted within this case study.

1.7 Organization of this Thesis

This chapter has provided an introduction, problem, and research goal to motivate this thesis. The remainder of this thesis is organized as follows: Chapter 2 provides a context for this research, including the research setting, an introduction to concepts, and a review of current research work in the field of coordination in software development. Chapter 3 describes the research setting, design, and data collection methods used to conduct this mixed methods study of interviews work management artifacts. Chapter 4 describes the qualitative analysis technique and thematic findings from stakeholder interviews along with the development of theoretical propositions. Chapter 5 describes the quantitative analysis and further refinement of one of the propositions using work item data from a software

engineering work management repository. Chapter 6 integrates the findings of the qualitative and quantitative work to provide a refined theoretical proposition describing one aspect of coordination in software engineering. Finally, Chapter 7 outlines future work, summarizes the contributions, and concludes the thesis.

CHAPTER 2:
BACKGROUND AND RELATED WORK

2.1 Introduction

Theory development is important in software engineering to guide reasoning and development of solutions to identified problems. However, most software engineering research focuses on research tools and experimental techniques and is not rooted in an explicitly acknowledged underlying theoretical motivation, stance, or a stated goal to generate a theoretical understanding of software engineering phenomena. Coordination theory within software engineering is beginning to be developed. The theories developed tend to be adapted from existing wisdom from software engineering, models from other domains, or from testing commonly held beliefs. This practice leaves open the possibility of missing important aspects or portions of theory are not readily conjectured.

The approach of our research is to provide a foundation to augment and expand existing theory by broadening our understanding of coordination from studying a practicing software development team then generating theoretical propositions describing coordination in software engineering. These propositions are guided by software engineering practitioners' views and experiences, and are generated, supported, and refined empirically from within the software engineering practice.

The remainder of this chapter reviews coordination in software engineering, theory development in software engineering, presents related research about coordination theory in software engineering, coordination tools and techniques, and social network analysis as it relates to coordination in software engineering.

2.2 What is Coordination?

Coordination as a phenomenon within people working in groups has been studied for the past several decades in the context of organizational management, and recently has seen more specialized study in the areas of computer supported collaborative work and software engineering. In the most general sense, coordination means the act of working together harmoniously. The term can be used to describe the "organization of the different elements of a complex body or activity so as to enable them to work together" (Compact Oxford English Dictionary) in the context of a system. Alternatively, it can be described as a "cooperative effort resulting in an effective relationship" (Compact Oxford English Dictionary) when describing a team.

Focusing on organizations and teams, coordination has been defined as "the integration or linking together of different parts of an organization to accomplish a collective set of tasks" (Van de Ven, Delbecq, and Koenig 1976, 322). This definition is in reference to the human, tool, and process oriented aspects of an organization. Each member must execute the correct function at the correct time in order to synchronize and accomplish a goal. Specializing coordination further to software engineering, Kraut and Streeter (1995, 69) state that "coordination in software development means that different people working on a common project agree to a common definition of what they are building, share information, and mesh their activities".

Most simply, "coordination is managing the dependencies between activities" (Malone and Crowston 1994, 87), and it is interesting to note that if there is no interdependence then there is nothing to coordinate. Malone and Crowston note that "often, however, good coordination is nearly invisible, and we sometimes notice

coordination most clearly when it is lacking" (1990, 357). As Conway states, coordination is so important, it "provides the only possibility that separate task groups will be able to consolidate their efforts into a unified system design" (Conway 1968). Coordination, or acting as a team and ensuring that all dependent entities are working together is critical in ensuring successful software development. This becomes increasingly important as the project size and geographic distribution scales increasing the effect of known coordination barriers such as distance, time shift, and cultural differences. As components of software are separated and the barriers to coordination are raised, it becomes ever more difficult to manage and resolve the interdependencies between components and work.

Clearly, coordination is a key aspect of successful software development, and it will become more important as the size and complexity of software engineering projects continues to scale and become globally distributed. By understanding and modeling coordination in software engineering we enable a theory driven model of development for processes and tools in industry and research. The concept and use of theory in software engineering along with related work in coordination theory is reviewed in the next sections.

2.3 Theory Development in Software Engineering

When developing experimental or novel techniques for developing software systems it can be useful to have an inspirational or driving force to guide your design. New directions in the development of research and experimental tools need to be driven by an underlying theory so that there is a guiding theme and motivation to drive the design. Theoretical models and explanations will generate this guiding theme by enabling analytic reasoning about the problem at hand by using a model of

the domain and known relationships between constructs. The use of such a theory allows a designer of a novel technique to predict the effects of their proposed technique within the theoretical domain before conducting experiments or industrial trials. Theory may also be used to create a benchmark against which to compare previous tools and results, and as a guide for coalescing and defining a research community and direction (Sim, Easterbrook, and Holt 2003).

Sjøberg, Dybå, and Anda (2008) describe theory as a model of a system where the concepts of the domain are characterized as constructs and the behaviours are characterized as propositions. Propositions relate or express an interaction between constructs. For example, a proposition might be "UML modelling during software development increases the reliability of the resulting software system", and the constructs in this case are "UML modelling" and "reliability".

In examining the possibility of theory development, Sjøberg et al. (2008, 316) explain that theory in software engineering can be achieved in three major ways: 1) Through application of existing theory from another domain. Adoption of previously existing theory consists of working with a previously existing theory from outside the domain and showing how the problem at hand maps to the existing theory. 2) Through refinement of existing theory. Adapting a theory that already exists within the domain is a process of refinement. Previous theory might be partially correct and new data or analysis may require adaption and correction of portions of the theory to explain any contradictory evidence. 3) Through development of new theory from scratch. New theory can be developed by working within a domain to understand its concepts and behaviours and then characterizing them. This characterization forms a theory that can be applied to new scenarios within the

domain. In the following sections we discuss previously developed theory of coordination and how it fits into these categories.

2.4 Coordination Research

In this section, research related to coordination in software engineering is described. This includes studies of the problems and characteristics of coordination, related research tools for enabling coordination, coordination theory in software engineering, and recent usage of social network analysis in coordination research.

2.4.1 Understanding Coordination from Practice

Recently, research in coordination has seen many research results that range from quantifying and identifying aspects of coordination, to developing experimental tools and techniques to assist in coordination, to research that develops theory of coordination in software engineering. Most of this research approaches a specific problem and attempts to characterize it, or presents a tool that aims to ease a specific coordination friction point and the theoretical stance of the research is generally not explicitly acknowledged or identified. Several examples of this type of research are presented here to demonstrate the interest in and apparent need for research studying coordination in software engineering.

Fussell et al. (1998) quantify the effect of coordination strategies and find that techniques used for communication predicted coordination and overload. De Souza et al. (2004) show how coordination and practices such as well-defined APIs can lead to coordination failures, such as at the time of system integration. Geographic distance is a commonly cited cause for coordination issues. Herbsleb et al. (2000) show that work across multiple sites takes longer than collocated work and Herbsleb and Mockus (2003a) show that geographically distributed development

teams are at a disadvantage due to limited communication and coordination ability. Communication patterns and social structures have been proposed as a measure for detecting coordination issues. Fonseca, de Souza, and Redmiles (2006) suggest using the social networks of developers to indicate potential coordination problems and Nguyen et al. (2008) show that failure in system integration can be predicted using communication patterns of the team building the system.

There are a number of published tools that attempt to deal with coordination problems. Most of these tools use analysis and innovative representation of software engineering artifacts to aid coordination. For example, Halverson et al. (2006) present task (or work item) visualization tools to alert developers to coordination needs by creating glyphs which compactly represent multiple attributes of work items. Many other tools visualize code dependencies and relationships to raise awareness of changes, including Tukan (Schümmer and Hake 2001), Palantir (Sarma, Noroozi, and van der Hoek 2003), and Ariadne (Trainer et al. 2005), which also bridges the gap between social and technical dependencies by analyzing code authorship. Storey, Čubranić, and Germán (2005) survey a variety of visualization tools to support human activities and interaction in software engineering settings. Further tools, such as Requirements Explorer (Kwan, Damian, and Storey 2006) and Emergent Expertise Locator (Minto and Murphy 2007) attempt to deduce dynamic and emerging teams based on dependencies between and authorship of either code, requirement, or work item artifacts in the engineering workspace. The Jazz research project (Cheng et al. 2003, 2007; Frost 2007) presents early research tools integrating collaborative development tools into the IDE. Jazz differs from many previous tools because it focuses on integrating tools such as group chat, instant messaging, and shared design artifacts to smooth and

enable collaboration and coordination. The Jazz research project has evolved into the IBM Rational Jazz product suite (Jazz Community Site).

Again, in terms of characterizing coordination, Cataldo et al. (2007) summarized four of their case studies of coordination and note that coordination problems still persist, even in the presence of tools and practices designed to mitigate them. This observation, along with the tools and techniques being produced, highlight the need for a theoretical characterization of coordination. If coordination problems persist despite attempts to mitigate them, or tools are not being used despite their prescription and applicability to certain problems, then the mechanisms that a team is using to coordinate are not understood and need to be discovered and modelled as a theory. Understanding of coordination and the problems introduced by modern software engineering suggest the use of a theory driven approach to development of tools and processes. Specifically, an approach that is rooted in the practices of the stakeholders in the software engineering process and can be iteratively refined and used by the software engineering coordination community to assist in shared reasoning, tool design, and collaborative research.

2.4.2 Coordination Theory in Software Engineering

The most common and most cited "theory" of coordination in software engineering is that of Conway's Law. It is first presented by Conway (1968) as observations on organizational structure, and has been taken up by the software engineering community as law. Conway states "organizations which design systems are constrained to produce designs which are copies of the communication structures of these organizations", (1968, 31). Or, roughly, in the context of developing software, that any piece of software reflects the organizational structure that produced it. This view of coordination is, in fact, not law, but an observation

upon which much of the theory of coordination research literature is built. Conway's observations seem intuitive and logical, and have been supported at times through empirical research. More recently, Herbsleb and Grinter (1999) revisited Conway's Law and suggest attending to the rule of Conway's Law by enforcing modularity, adding site visits to mitigate distance, and using tools to share information and enable coordination.

Several recent works have addressed theory of coordination using all three of these theory development techniques of applying models from other domains, modifying theory, and generating theory from within a domain. In refining theory, Herbsleb and Mockus formulate a theory (2003b) of coordination using Conway's Law and the Parnas effect, a measure of the impact of changes due to modularity of a software systems, based on modularity suggestions by Parnas (1972), as a basis for generating hypotheses. These hypotheses describe outcomes of coordination, are formalized as mathematical relationships, and are empirically tested within a case study to determine their applicability. In a later study, Herbsleb, Mockus, and Roberts (2006) present a theory of coordination as a distributed constraint satisfaction problem. This is an example of adopting theory or common problem pattern from another domain and mapping it into the domain of interest. Previously, Mockus, Fielding, and Herbsleb (2002) demonstrate the technique of developing theoretical propositions (as hypotheses) and testing them in two open source projects as a first step towards theory development.

All of these contributions are useful in providing several steps towards a theory driven process for research and development of tools and processes. This thesis seeks to provide further avenues for broadening empirical theories of coordination in software engineering by generating theoretical propositions that describe the

coordination of interdependent work in a practicing software development team. These previous theoretical contributions are discussed in the context of our results in Chapter 6.

2.4.3 Social Network Analysis in Coordination Research

Chapter 5 of this thesis uses network analysis techniques to evaluate the structure of dependencies between work items. Network analysis and specifically social network analysis has a rich history in the social sciences, and has been taken up recently by software engineering researchers. Social network analysis measures such as centrality, a measure of how centralized a node is within a network, or brokerage, a measure of the number of paths which include a node (Wasserman and Faust 1994), quantify the importance of nodes in the network and give a measure of the structure of networks. By conducting analyses with respect to social network analysis measures we get can gain insight into which types of network structures are effective for coordination outcomes.

Social network analysis has been used to model and evaluate the structure of social interaction patterns in software projects, such as Crowston and Howison's (2005) work examining social communication structures in open source projects, and Bird et al.'s (2006) study of the social networks in email archives of open source projects as a mechanism to study the coordination patterns of the developers. Further work has studied the evolution of those structures such as Damian et al.'s (2007a) work showing that communication based social networks are dynamic around work items. Damian, Marczak, and Kwan (2007b) further use social networks in requirements engineering to identify patterns of collaboration across geographically distributed development sites.

Recently, network analysis has been used in machine learning scenarios, building predictors to alert participants in the software development process to upcoming trouble. Nguyen et al.'s (2008) tools predicted software build results using social network analysis of the communication between those who contributed to the build. Zimmermann and Nagappan (2008) adapted the use of social network analysis, using network measures not with people or communication, but with dependencies between software components, predicting defects based on the structure of the dependency graph. It is this adapted usage of network analysis that we use in Chapter 5 of this thesis, where we analyze the structure of dependencies relationships between work item artifacts in relation to resolution time.

2.4.4 Opportunity for this Research

There are many studies of coordination assistance tools and quantifying aspects of coordination within software engineering as well as early theory works such as Conway's (1968) which are, fundamentally, theory development through observation. Other advances in theoretical models are examples of refining theory from within a domain (Herbsleb and Mockus 2003b) and applying theory from another domain (Herbsleb, Mockus, and Roberts 2006). This thesis fits into this theory development setting by generating propositions that can augment existing theory or form the basis of new theory, and anchoring the development of these theoretical propositions in the views and observations of software engineering practitioners in the field.

This thesis seeks to build a set of theoretical propositions describing coordination from within the domain of software engineering by firmly anchoring the theoretical propositions in the views and experiences of the stakeholders who practice the coordination. The intention is that, by generating more theoretical propositions and

beginning to refine them through further empirical studies, existing theory can be augmented and broadened. This thesis forms the start of that process.

Since coordination problems are still present even in the face of tooling and processes to mitigate them, we take an exploratory approach to ask directly from the software engineering stakeholders how they manage to coordinate interdependent work. Our exploratory approach first analyzes interviews to develop a set of theoretical propositions that describe the relationships between constructs in the software engineering domain around coordination of interdependent work. Then, one of these propositions is further refined using a statistical approach, demonstrating how to increase refine and increase the applicability of a developed theoretical proposition.

2.5 Summary

Software engineering coordination research has approached coordination from several angles; analyzing the behaviour of software engineering projects, proposing tools to aid coordination and collaboration, and several theoretical contributions. There is an opportunity to create and foster further theory of coordination that is based on the practices of stakeholders in the software engineering process. It is this approach of beginning to derive theory from practice that is the focus of this research. In the next chapters we develop our research design and methods to capitalize on the views and experiences of software developers to derive theory from practice.

CHAPTER 3:
RESEARCH DESIGN, SETTING, AND DATA COLLECTION

3.1 Introduction

This chapter presents our motivation for using an exploratory case-study research design for theory building using interviews and work item analysis. We describe the configuration of mixed methods in data collection and analysis. Then, the research setting and context is described along with an explanation of the data collection methods used to acquire data for analysis in both stages of this research. Data analysis methods, along with the results of the data collection and analysis of interviews and work item data are presented in Chapter 4 and Chapter 5.

3.2 Research Goal & Approach

The goal of this research is to gain a greater understanding of the coordination of interdependent activities in software engineering and to generate theoretical propositions in order to work towards building or expanding theories of coordination in software engineering. In this context, coordination activities are considered to be activities that lead to the resolution of interdependent tasks. This work is relevant to research and industrial settings because the development of theoretical models of coordination allows reasoning and prediction about how new or proposed processes and tools might function. By using, testing, and extending our theoretical understanding of coordination in software engineering, we enable a higher level of discussion and inquiry.

Our approach is to contribute to the development of theory by generating theoretical propositions and by beginning to evaluate these propositions empirically

using mixed methods in data collection and analysis. First, we interviewed practicing software engineers to determine how they view and practice coordination of interdependent work. A qualitative thematic analysis of these interviews was performed and the generated themes were used in conjunction with contextual knowledge from observations of the project to form theoretical propositions. Second, we performed a quantitative statistical analysis of work item data extracted from the same team's repository to further explore one of the propositions. This analysis allowed for refinement of the proposition derived from participant's observations. This process formed a refined propositions that is supported and defined by collected data and analyses in the case under study. By iteratively developing a proposition using multiple data sources and analyses we enhanced the depth and strength of the findings.

3.3 Research Setting

3.3.1 Jazz Project

The site of this case study was a professional software development team from IBM. They are part of the team developing the larger *Jazz* project, which is a new suite of products that has evolved from the Jazz research project originating from IBM Research (Cheng et al. 2003, 2007; Frost 2007). The Jazz project has been under development for the past several years and is approaching it first production release. The intention of the Jazz project is to integrate multiple tools and activities of software development into a shareable and integrated tool for software development teams. This set of tools can be thought of as an integrated development environment for teams, designed to enable and facilitate the common communication, collaboration, and coordination processes teams use to produce complex software.

In the words of the Jazz team: “Jazz is an IBM Rational project to build a scalable, extensible team collaboration platform for integrating work across the phases of the development lifecycle.” (Jazz Community Site). The Jazz suite will encompass several products in several component configurations. The core components that will be packaged are a work management system, a source configuration management system, an automated build system, and an agile planning and scheduling system. By using tight integration and interoperation of tools, the Jazz team claims that more efficient and effective software engineering is possible.

3.3.2 Jazz Development Team

The initial stage of this study was conducted using interviews with members of one team from the Jazz project as well as work item data from the engineering repository for the entire Jazz development team. The larger Jazz development team consists of approximately 150 contributors and 31 functional teams, with some teams as sub-teams of larger teams and some contributors assigned to multiple teams. These team members are located at 15 locations worldwide, primarily in North America and Europe.

The development process used to develop Jazz is called the Eclipse Way (Eclipsepedia, Gamma 2005, Frost 2007). It is a process that the Eclipse foundation and development team have created during the development and subsequent maturation of the Eclipse platform. The Eclipse Way is an iteration-based agile development process that emphasizes delivery of a working product at the end of each 6-10 week milestone, continuous integration, automated testing, and dedicated time for testing and retrospectives at the end of each milestone. The process focuses on open and transparent planning commitments for teams at the beginning of

milestones, and delivering a workable version of those features on-time, emphasizing removal of functionality over delay of milestone releases.

The interview portion of this study was conducted at an IBM regional office. The interview participants were all members of the same team. Six participants were located at the office and one at another IBM office. The team's component consists of a server component, and several client components and the functionality of the code ranges from server-side action implementations to command-line and graphical user interfaces.

3.4 Research Design

This research uses an exploratory mixed methods approach to developing, then refining theoretical propositions of coordination in software engineering. The design of this research integrates qualitative analysis of interviews with software developers, with quantitative analysis of a work item database. At the outset of this study we knew we had access to software developers as research participants, and that we could access data from their software engineering repositories. The research design integrates and utilizes these two data sources by using data collection and analysis methods in series, allowing the quantitative analysis to refine the qualitative research results.

The first stage of the research was to design and conduct semi-structured participant interviews with up to 10 software developers. Then, using transcripts from the interviews, a qualitative thematic analysis using techniques drawn from grounded theory was completed. Using the thematic analysis and contextual information from the project, theoretical propositions were developed that describe how coordination of interdependent work occurred within the project. In the

second stage of the research, one of these propositions was chosen to be further explored using work item information from the entire project to refine and augment the theoretical proposition. Quantitative statistical techniques were used to explore and test the structure of explicit work item dependencies extracted from the project's work item repository.

Our research method integrates multiple data sources and analyses in order to compare results and develop a empirically supported theoretical proposition that is rooted in the practices of a professional software development team. Using a mixed methods case study research design we build upon each stage of the research and develop a set of theoretical propositions and refine one of them with further evidence. This comparison and refinement using distinct approaches, enables the development of a more applicable proposition than would be possible using a single approach, maximizing the utility of the available data sources within the case under study.

3.4.1 Case Study

Case studies are often used in software engineering research, as it can be very difficult to obtain empirical data or on-going access to working software developers, making large scale or comparative research methods rarely plausible. Since data can be limited, case study approaches allow researchers to extract the maximum amount of value from each piece of available data. In this research project, work item data and interactions with a professional software development team were available with one team in a larger project, making a case study approach particularly fitting for this research. Case studies can provide a rich and meaningful insight into an instance in a research domain, and can shed considerable light on the domain in general.

Case studies are appropriate when the research asks why or how questions and when the "focus is on a contemporary phenomenon in some real-life context" (Yin 2003, 13). Further, case studies allow the researcher to carry out exploratory "research into the processes leading to results" (Gillham 2000, 11). Easterbrook et al. (2008, 296) suggest, in the context of software engineering, that "exploratory case studies can be used as initial investigations of some phenomena to derive new hypotheses and build theories." Since our goal is to develop theoretical propositions describing coordination, it follows that using a case study approach is appropriate, allowing the process of coordination to be studied and understood as a current phenomenon. These qualities of a case study approach fit well with our research goal of understanding coordination of interdependent activities in software engineering scenarios.

Case studies can be used in the development of theory by generating a local theory that is rooted within the context of one specific case. As Sjøberg et al. (2008) describe, a local theory is a theory about the domain of interest that is valid within the case it was developed. Through multiple case studies, either through repetition or approaching the same problems using differing methods, one can develop a strong understanding of a field of interest. This local theory can then be tested, extended, and refined by applying it to other cases within the domain. Through successive application and refinement of a theory within a domain, the validity and overall applicability within the domain is increased. Figure 2 illustrates the application of two case studies in a domain of interest and area of applicability of a theory that is developed by the combination of those two cases. In this research approach we use one case and generate theoretical propositions that are valid

within the case, and can be used in further cases to extend their applicability and to facilitate further theory development.

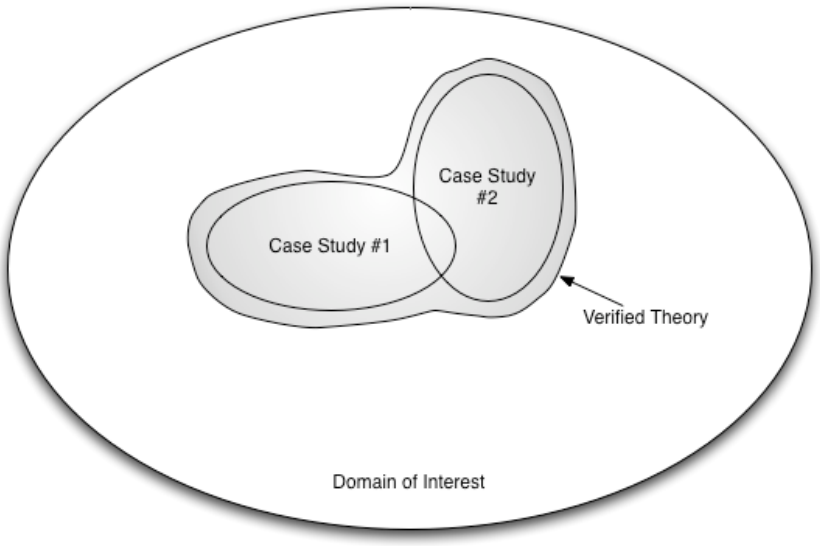

Figure 2: Building theory using case studies

3.4.2 Mixed Methods Data Collection and Analysis

This study used a sequential exploratory mixed methods design (Creswell 2003, 213; Leech and Onwuegbuzie 2007) in data collection and analysis, employing qualitative then quantitative methods, using interview and work item repository data sources respectively. This approach was used to converge and confirm findings from one method to another and as Johnson and Onwuegbuzie (2004, 16) suggest, using a pragmatic approach that "attempt[s] to fit together the insights provided by qualitative and quantitative research". An overview of the sequence and flow of data and findings between the methods is illustrated in Figure 3. The configuration of methods allowed an exploratory approach to generation of theoretical propositions using qualitative methods and subsequently refining them using quantitative methods.

We gained access to two primary data sources: face-to-face access to practicing software engineers from a team in the Jazz project and a work item repository from the Jazz project which consists of a superset contributors interviewed team. While each of these data sources is well suited to many types of data collection and analysis methods, for this research we focused on two that best address our research goal: 1) qualitative thematic analysis of transcripts from interviews with software developers, and 2) quantitative statistical analysis of work item attributes and explicit dependency structures.

The first stage of the research consisted of conducting interviews with software developers followed by transcription and qualitative thematic analysis of these transcripts. The goal and output of this method is a set of theoretical propositions that describe coordination of interdependent work. These propositions are then ready for further exploration and refinement. In the second stage of the research, one of the propositions generated in the qualitative analysis of interviews stage is quantitatively explored by using statistical techniques on work item data extracted from the team's Jazz repository. Statistical testing allows the proposition to be explored and to assess the agreement with the qualitative analysis. The integration of results is used to further explain and add detail to the proposition. By exploring the phenomenon of coordination of interdependent work through multiple research methods we are able to generate greater insight into our research questions.

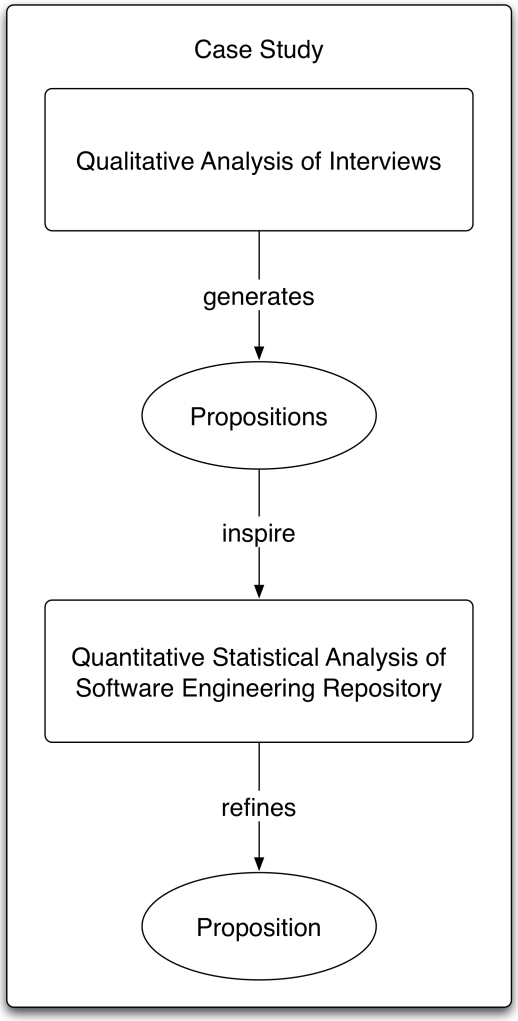

Figure 3: Arrangement of research methods

3.5 Data Collection

Data collection was conducted on-site at an IBM regional office over three months, between June 2007 and August 2007. The first two months were spent observing and building rapport with the target team. This observation consisted of document inspection of the teams plans, wikis, the Jazz work item repository, and casual conversations with team members. Using this contextual knowledge, the interview scripts were designed and the repository data extraction process and tools were planned and constructed. This period of unstructured casual observation was not

directly used as a data source, but provided insight into the team, site, and appropriate methods for structured data collection.

3.5.1 Interviews

The structured collection of data into how software developers coordinate interdependent work began with a series of interviews with software developers in the Jazz project development team. Prior to the formal data collection of semi-structured interviews, the team was observed and the project structure and development practices were monitored. This observation period was used to help design an interview that would be relevant and appropriate for participants of the project at hand. Interviews were used as the first type of data collection as a method of grounding the developing theoretical propositions in the views and experiences of the software development team in this case study. The outcome of the qualitative interview analysis was used to develop theoretical propositions to be used in the quantitative analysis of the work items from the same project.

Participants were interviewed using a semi-structured questioning approach allowing for follow-up and clarification of responses. The interview questions were scripted so that the same questions would be asked, in the same order, to each participant. Each question also had several predefined follow-up prompts for the interviewer. Follow-up prompts were designed to ensure consistent coverage of topics, either probing specific areas for more detail, or providing direction for the interviewer for further questioning. The interview script consisted of 13 questions with follow-up prompts for each question. The major questions were all worded and asked in the same manner for each participant, but the follow-up questions varied by response. The semi-structured nature of the interviews meant that the interviewer was free to let participants guide their responses in whatever direction

they felt was relevant, but also ensured topic coverage was consistent across participants. The interviewer did not re-focus participants or limit their contributions to topics perceived to be relevant, but did ask follow-up questions to elicit more information about responses that seemed important.

The questions in the interview script were clustered in three major topics: context, workflow and coordination, and issue conceptualization. Contextual questions, such as "Tell me about your roles, responsibilities, and typical activities within the Jazz team(s)" or "What are your typical activities and interactions during a workday?" were used to acquire some background information about each participant. This information along with notes on observations taken during the interview helped to form a background story and uncover similar and potentially differentiating aspects of each participant.

Workflow and coordination questions such as "During your work, how do you determine what you should be working on?" and "Do you find that there are dependencies between your work and work by others in your team or from other teams? How do you find these dependencies?" probe into how each team member finds what work they should complete, how they know what others are working on, and how they manage the interdependencies between those pieces of work. Participants were asked specifically, how they conduct these activities, what tools they use, what data they create in engineering repositories, and how they recall that data when necessary.

Issue conceptualization questions such as "If you are working on an issue, how do you know that you understand the whole issue?" and "Describe your communication with others when assessing work and finding related work" were asked. These questions were asked to assess which software development artifacts are used

during development and how participants use software repositories and other engineering tools to understand issues with the software under development. The goal of these questions was to reveal the tool usage and behaviour patterns participants used to gain an understanding of a development task or problem. The full interview question script is shown in Appendix A.

Each interview was administered by the same researcher in August 2007. All interviews were conducted privately and individually and were scheduled for 60 minutes and took between 40 and 70 minutes. The interviews were conversational in style, and were audio recorded for later transcription and analysis.

3.5.1.1 Participant Recruitment

Seven interview participants were recruited using an internal broadcast email sent by a third party from within IBM. The recruiter sent the email, shown in full in Appendix B, directly to a list of potential participants. The participants responded directly to the interviewer, independent and anonymous of the recruiter. Interested participants communicated directly with the interviewer to obtain further information about the interview to decide if they wanted to participate. Each participant arranged and attended their interview independently and anonymously from co-workers, their employer, and the public. Participants were not compensated for their involvement in the study.

At the beginning of each interview, each participant read and signed a consent form, shown in full in Appendix C. Any questions arising from the review of this document were discussed and resolved prior to participation. This form outlined the purpose of the study, potential benefits of the research, the research groups conducting the study, and the sponsoring organization. The form also described protections afforded to protect the participant's anonymity, privacy, and

confidentiality. Finally, it outlined known potential harms that may arise from their participation in the study, such as unintentional breach of privacy resulting in professional harm. Ethical approval of the research protocol for this study was obtained from the University of Victoria Human Research Ethics Board.

3.5.1.2 Interview Participants

All seven interview participants in our study belonged to the same team of the Jazz project. Six interviews were conducted in-person and on-site and one interview was conducted via telephone with a participant at another IBM location. All participants were technical members of the development team, with responsibilities for several sub-components ranging from server modules to user interfaces. Work experience of the participants with this organization ranged from less than one to approximately 15 years, and the project itself has approximately 3 years of history. Experience of the participants includes areas such as real-time object-oriented modelling, user-interface toolkit development, and software configuration management. Most members of the team are originally from other product teams within the organization having experience primarily in the Rational product divisions and the Eclipse foundation. To protect the anonymity and confidentiality of the participants they are referred to as Participant 1 through Participant 7 and names and identifying information are redacted or anonymized as necessary.

3.5.2 Software Engineering Repository Data Collection

The second phase of this research examined artifact data collected from the Jazz repository used by the Jazz development team. The Jazz repository serves as a central repository for many of the functions that are commonly distributed through multiple tools in the software development process. In particular, the Jazz

repository contains records of, and user interfaces for, work items, source control, automated builds and tests, planning, and process control. All artifact types are stored in a common repository structure accessible through a web services API, an Eclipse-based client, and a web-based client. The Eclipse-based client and the web-based client both use the web services API to query and manipulate data in the repository. The repository uses a relational database to maintain persistence for its transactional operations.

In order to acquire work item data to quantitatively analyze we determined that the optimal way to acquire a copy of the Jazz development team's Jazz repository was to obtain a backup copy of the transactional database. This would be reverse engineered using the Jazz client as a guide to functionality and operation. Unfortunately, this approach was not available, due to privacy and legal reasons. There was no available access to the database used to store the records for the Jazz repository, nor was there a system or specified format for data interchange. To overcome this lack of access to data, we built and used a custom data extraction tool to acquire work item data for analysis.

Our data extraction tool is an Eclipse plug-in that operates using the Jazz client API. The plug-in uses the Java API available to Jazz clients to conduct queries for artifacts of interest. The queries are coded using the same techniques and APIs that the Eclipse-based Jazz client uses to build its UI and functionality. Upon completion of the queries the Java objects received are serialized to XML and written to files on disk. This plug-in was run against the Jazz server for the Jazz development team. Since the Jazz development server is restricted to internal IBM operations and communication networks, the plug-in was run from within the IBM enterprise and the resulting XML files were securely transferred to a research server.

The XML was then deserialized and imported into a research oriented relational database schema. In this case, we use MySQL as a relational database server. This schema and configuration allows the artifacts to be stored in a manner that is simple to query and analyze for answering research questions and uses a database schema that is independent of architectural and optimization concerns of the Jazz product. In this manner, all available work items were extracted from the Jazz team's Jazz repository. This data set consists of 24,838 work items created between June 10th, 2005 and January 21st, 2008.

3.6 Summary

This research utilizes a case study approach, while employing mixed methods in data collection and analysis. This configuration supports building of theoretical propositions by allowing an exploration and refinement of coordination practices. First, the views and observations of participants in the software engineering process were collected via interviews, and the transcripts were thematically analyzed using coding techniques drawn from grounded theory. The results of the thematic analysis were used to develop theoretical propositions describing aspects of coordination in software engineering. Then, using one proposition, a deeper quantitative analysis was conducted using work item artifacts extracted from the team's Jazz repository. The specific analysis techniques and findings for qualitative and quantitative stages of this research are described in Chapter 4 and Chapter 5 respectively.

CHAPTER 4:
INTERVIEW ANALYSIS AND FINDINGS

4.1 Introduction

In this chapter, the interview analysis method, including transcription, coding, and thematic analysis are described along with findings and initial theoretical propositions. Analysis techniques drawn from grounded theory were used, in order to generate a theoretical description of how coordination of interdependent work is managed in a software engineering environment. After reviewing the analysis techniques, we then describe the key findings from this stage of the research: four major themes of proximity, work item authoring patterns, properties of coordination mechanisms, and uncooperative behaviours. Following a review of the thematic analysis, a set of theoretical propositions are developed and presented that emerged from this analysis of the views and experiences the interview participants. These theoretical propositions are the first version of propositions describing coordination of interdependent work in software engineering and they form the basis for further refinement.

4.2 Interview Analysis

Participant interviews were conducted with 7 participants as described in Chapter 3. All were audio recorded and notes were taken during the interview. The interview recordings were then transcribed into text documents for use in coding and analysis. Interview data was coded and analyzed using a qualitative thematic analysis approach using coding methods drawn from grounded theory. Coding techniques used in this research were drawn from grounded theory (Glaser and

Strauss 1967; Strauss and Corbin 1998; Charmaz 2006) because it provides a methodology and approach that builds theory based on the data of the situation under study. The intention of using grounded theory coding techniques in this analysis was to stay rooted in the views and experiences of the participants as coordination practitioners. As grounded theory seeks to discover and develop theories from research grounded in data rather than deducing testable hypotheses from existing theories (Glaser and Strauss 1967; Charmaz 2006), we use grounded theory coding techniques in our methods to see what our participants lives are like, and to study how they explain their statements and action, as they have the most intimate knowledge of their participation in a phenomenon (Charmaz 2006).

Our goals are to build theoretical propositions that are well supported in the views and experiences of the participants of this study. With this in mind, the study was approached with no preconceived notion of how developers collaborate to resolve dependencies, attempting to bring a blank slate to the analysis. It follows that methods and techniques drawn from a grounded theory approach are appropriate in this study in order to generate theoretical propositions that are based in the views and experiences of the stakeholders within the case study. These theoretical propositions can be evaluated and modified iteratively using further approaches such as the quantitative analysis of one of the propositions presented in Chapter 5.

Figure 4 illustrates the analysis process from stakeholder interviews through to proposition development. The first step in the interview analysis process was the transcription of audio files to plain text transcripts. To facilitate deeper understanding of the interviews and to provide insight and familiarity with the participant's observations and experiences the interviews were transcribed by the

author. Audio from all interviews was transcribed by the author using the NCH Express Scribe tool for variable speed audio playback. The transcription recorded roughly word-for-word the spoken words of the interviews. Non-conversational sounds and other noises were not transcribed, however the words spoken were not paraphrased or reordered. This resulted in approximately 43,000 transcribed words over approximately 170 pages of transcripts. After completion of the transcription process the transcripts were ready for qualitative coding.

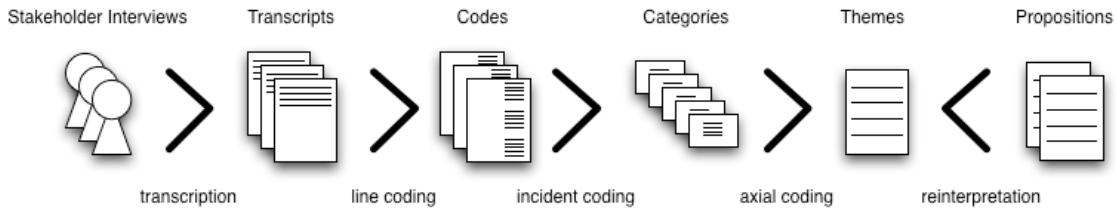

Figure 4: Qualitative Analysis Process

A thematic coding analysis was conducted on the interview transcripts using a series of grounded theory coding techniques illustrated by and adapted from Charmaz (2006, ch. 3), Strauss and Corbin (1998, chs. 8, 9, 10), and Glaser and Strauss (1967). These coding techniques were line-by-line, incident-by-incident, and axial coding. Line-by-line coding applies codes to each line of the transcript regardless of whether the line forms a complete sentence or phrase. This technique is applied initially to create a set of codes that represents the low level concepts expressed by the participants without emphasis on the meaning or inter-relationship, or interpretation of the concepts. After line-by-line coding, the transcripts were coded again using incident-by-incident coding. This is a higher level coding technique that combines previous codes as appropriate and codes incidences of specific phenomena. For example, it might appear that a participant is concerned about privacy of some of their work and this incidence would be coded as *privacy*. Finally, axial coding was used to coalesce and refactor previous codes into

categories and, finally, into themes. Axial coding involves reviewing the existing codes and conceptually mapping them into higher level concepts. These are then applied as codes within the transcripts. Coding was conducted using the Atlas.ti computer-aided qualitative data analysis tool. The coding process developed 444 distinct line, incident, and axial level codes after conducting and refactoring codes using the three coding techniques. An example section of transcript along with codes is shown in Figure 5.

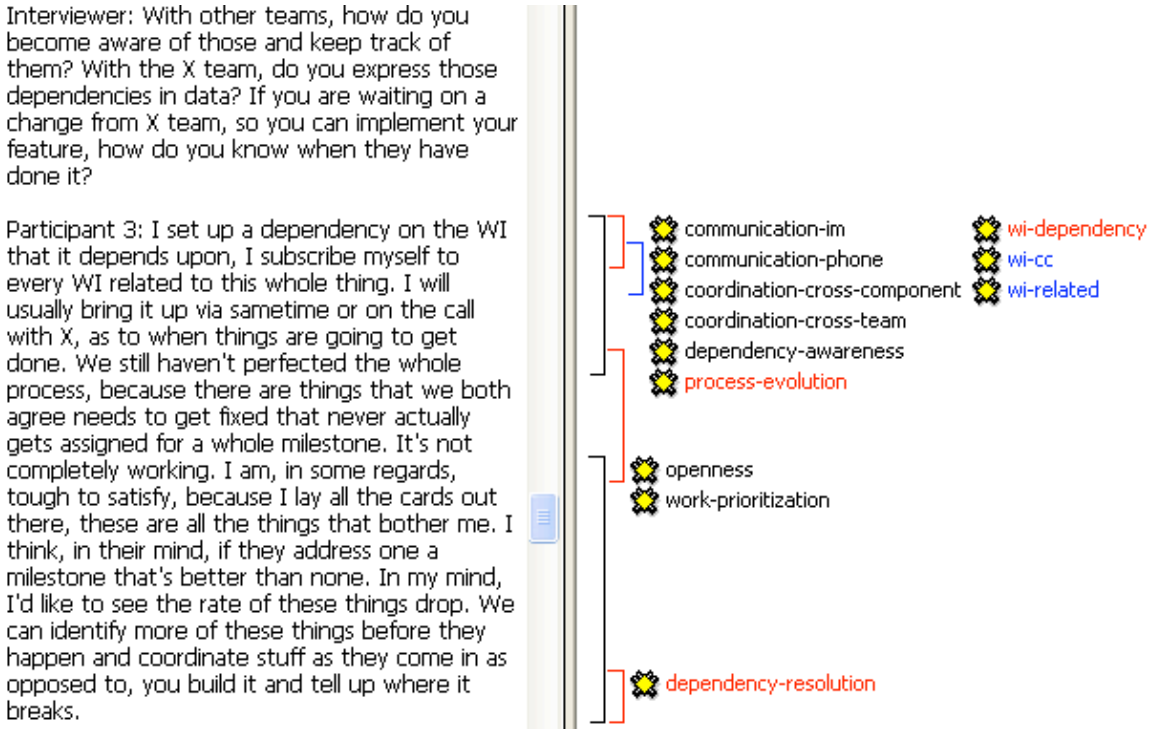

Figure 5: Example transcript coding

4.3 Themes

The coding process formed the basis of a theory generation portion of the analysis. Using the axial coding technique, four major themes of proximity, modification request authoring patterns, properties of coordination mechanisms, and strategic behaviours were derived from interview analysis. These themes, along

with context derived from the interview analysis and informal observations, are described in the following subsections.

4.3.1 Proximity in Space, Software, and Organization

Proximity is a common topic that is often discussed in Computer Supported Collaborative Work and collaboration literature, and as Kraut (1988) notes, in research organizations there is a marked increase in collaboration when collaboration partners are physically nearby. Similarly, our analysis reveals that proximity is an issue directly related to ease of collaboration and coordination. Challenges arise not only in relation to *geographic* proximity, but also *architectural* and *organizational* proximity.

Architectural proximity is closeness within the system architecture of a participant, or their work, relative to a coordination partner's location in the architecture. For example, if two people are working on the same component, they would have high architectural proximity, whereas, if they were working on different subsystems of the same product, they would have lower architectural proximity. Organizational proximity is the closeness of participants in the team structure of the organization. Participants located at different sites, but assigned to the same team, would have lower geographic proximity, yet they would still have higher organizational proximity. Coordination techniques around these concerns has also been noted by Herbsleb and Grinter (1999) in their review of Conway's Law. These variants seem logical, as it is not only how physically distant your colleagues are, but also how organizationally distant their team is, and how architecturally separated and differentiated their work product and expertise are, that affects coordination potential.

Participants reported that it is more difficult to coordinate work over geographic distances with work partners working on tightly coupled work, or with work partners working on dependent but loosely coupled work. The source of the difficulty seems to arise from maintaining open communication and awareness channels that deliver information at the correct times. In some cases, participants noted that they would not even attempt to conduct highly coordinated work with work partners who have low proximity. They would, instead, attempt to assign such work to other colleagues, or to modularize their work to match the coordination abilities of their potential work partner.

> Participant 2: There have been times in the past where I have worked more closely with X, and we have, if needed, picked up the phone and had a higher bandwidth conversation. X has also come and visited here a couple of time, so I who X is […] and to a point, what kind of person X is. I think that's important for a team, so it's not just some name on the other end of an instant message or work item or something.
>
> As far as day to day work goes, I'm not normally interacting that closely with X, and I think because X is […] a remote person […] it is harder to maintain a daily awareness of what X is up to. It's, just as anything, easier to get up and walk down the hallway and have a conversation.

Similarly, it was reported to be more difficult to coordinate across APIs, or with other teams, reflecting the influence of architectural and organizational proximity, either because of differences in technical needs and abilities, in the case of architectural proximity, or because of differences in cultural or political stances within the organization. For example, Participant 6 noted issues with low organizational proximity while discussing issues around working with a remote team member:

> Participant 7: It's not so much that X is [geographically] remote, but the culture is different. Here we have the same Eclipse culture of doing thing, whereas [there] they have the background of ClearCase and Rational where they do things in longer cycles. It's a different way to approach things. If I work with people [from the] Process [team] who come from Eclipse it's more like the same way of working.

4.3.2 Work Item Authoring Patterns for Dependency Management

Individual developers noted the use of links between work items to define relationships and to assist in the management of dependencies between work items. Relationships are authored during the creation and management of work items by adding typed links between work items. Possible relationships types are: *duplicate*, *related*, *parent/child*, and *blocking/depends on*, typical of many work item management systems.

Participants expressed, to varying degrees, that they used links to create relationships between individual work items and that they traverse these relationships as a source of information when attempting to diagnose bugs, or while designing solutions to meet requirements. Further, they allow others to view the work items and become aware of dependencies. Participants also noted their creation and maintenance of links helped recall important pieces of information when assessing or returning to a work item. The participants involved with a collection of related work items could also be used to identify contributors who may be interested in the changes to a work item. Participant 2 discusses several types of relationships used in work item management:

> Participant 2: You can mark work items as explicitly blocking others or being blocked by another one, and we do use that for coordination across the different layers, across teams, or across components, or across server and client APIs, low-level versus the UI feature. The description

> thing for example, I couldn't add the UI for that until X added get and set description and description property on type workspaces. So we'll often have two work items for that, one for the low-level API work and one for the UI work and mark as dependent on the other.
>
> […]
>
> There might be a related issue that isn't the same issue so you wouldn't mark it as a dupe, but you want these two tied together for whatever reason, just awareness for the owner of the other problem that there is a related issue over here that I'm working on.

There was a notable relationship between the amount participants said they cared about the linking relationships and their location in the software stack or component hierarchy. Developers located at the top of the hierarchy—those developing components that depend on many other components, such as user interfaces—cared significantly more than others, because they had many more dependencies of lower layers in the software hierarchy. These developers invested much more energy into the management of links, and work items in general, than developers located at the bottom of the hierarchy, such as those working on data management routines.

Several recurring patterns of artifact generation and structures for dependency management were noted. These patterns allow developers to remember dependencies, raise awareness of dependencies, and facilitate coordination to resolve dependencies. In particular, the expression of dependencies or other relationships as data allowed tooling to effectively propagate awareness notifications. A number of common dependency management patterns in work items are described below.

Decomposition was used by some developers in order to break assigned work into manageable pieces, allowing incremental completion of technical implementation. The degree of decomposition ranges from simply attaching progress updates to a work item, to fully decomposing one into atomic units of work, creating trees of work items linked through *parent/child* relationships. This technique is used to allow fine-grained planning, delegation, and implementation of a larger work item or feature.

Collation was used to mark duplicates, and to cluster related work items. Duplicate work items are marked using a *duplicates* relationship and are effectively closed. Related work items are linked together using *related* relationships, though some participants noted trouble determining in which work item a conversation was occurring. By combining information from several work items into one, the conversation could be centralized. This enables more effective monitoring of changes to an issue.

Listing was used to manage a list of work items that needed to be completed in order to finish some larger task. This task or goal is represented using a master work item, sometimes referred to as an umbrella, checklist, or super work item. This pattern is used when working towards a larger goal, such as a release. In the following quote Participant 4 describes how many work items might get linked together using *duplicates* and *related* relationships, how they get listed in an umbrella or management work items, and other links to different artifact types, such as planning document:

> Interviewer: For something [that] is in progress, maybe it's a bug that's come in and it turned into fifteen work items. How do you keep track of everything that's going on there?

> Participant 4: If it's something that big, like really fifteen work items, it's usually on the plan, and we'll keep track in one master work item, we'll try to dupe them all and have one, or link them all together and have one that tracks the actual work that's happening and we'll have it in the plan, and we'll have the work item assigned. We try and do some timing information of work items, but it's going to be taking this much time. If there's adoptions required like other teams, we'll give teams an idea of when we're going to deliver it to the team, and then they have to when work might be required.

A **blocking** pattern for dependency management across components was seen in several forms. The desire to express a blocking relationship between two work items would commonly be implemented by creating a *blocking/depends on* link to a work item that represents an unresolved bug in the database, creating it if necessary. Participant 7 discusses the blocking pattern using two work items:

> Participant 7: If [a work item] blocks a lot of people, usually it's important. If it seems to me that it could reveal a bigger problem of some sort. Like down the road it could lead to database corruption.

Participant 4 also commented on blocking patterns:

> Participant 4: I think one of the best practices that we've been trying to think about is that it would really help if the originating team had a work item and actually marked it as blocked to ours because that would be a good communication mechanism that it's something that's blocking them.
>
> So, in theory, we can trace back and see that team X has to implement security for their X although to implement that they need most of it from us, it would help if they had a task themselves that was linked into our request so that at some point when we have queries across links in Jazz, I'd be able to have a quick way of saying just show me all the dependencies we have.

The blocking work item is then assigned to the target team. This technique is desirable because it further decomposes the problem and allows artifact awareness and management systems to operate more effectively than if the work item was simply reassigned from team to team.

4.3.3 Properties of Coordination Mechanisms

Throughout the analysis of the interview transcripts concepts of trust, accountability, synchronicity, and privacy repeatedly arose in discussions about the choices participants made in how they would communicate and act in order to coordinate dependent work. Trust, accountability, synchronicity, and privacy are all descriptive properties of communication and coordination mechanisms.

Trust refers to how much the user trusts that by using an artifact or communication mechanism the message will be conveyed to the recipient, or when coordinating with another person or team, how assured the participant is that the other party will respond appropriately, or perhaps even get the message at all. If one does not trust that the message will be received by the recipient or that an action will not be interpreted correctly, or even that the recipient will not respond in the desired manner, then the trust in the usefulness of the coordination mechanism is reduced. An example of not trusting the information received from coordination tools, such as work items, and needing to use their contextual knowledge of the project to filter information, is shown below:

> Interviewer: Do you ever select thinks based on the priority or the severity that the submitter sets?
>
> Participant 7: No, I don't care what they think, how severe they think it is.
>
> Interviewer: They can set one of those fields right?

> Participant 7: Oh, they can set it, but I just ignore it. X files everything as critical or blocker. So, if I were to select on that I'd be working on all of X's bugs.

Accountability refers to the degree to which specific actions will be accountable. For example, when communicating about a dependency problem via instant messenger the conversation may be effective but there is generally no recorded transcript of the conversation, and no audit trail for future participants to follow. By not being able to record or store a conversation or design decision, coordination tools such as instant messenger can become less attractive, and at other times more attractive if lack of accountability or increased privacy is desired.

Privacy was a consideration when choosing a communication medium. Specifically participants would choose a mechanism that would protect either the sender or recipients from adverse exposure in potentially embarrassing or contentious situations. Sometimes information was withheld from normal communication medium, such as work items, wikis, and mailing lists to maintain the privacy of an engineering initiative in order to prevent unwanted conversation or group noise.

Synchronicity refers to the degree to which the coordination mechanism, usually a communication mechanism, provides synchronous communication capabilities. A dependency could be resolved through discussion and negotiation in a face-to-face manner, on a phone call, via instant messenger conversation, or via a conversation in a work item, among other options. The previous options provide a range of synchronicity affordances and will be chosen according to coordination needs. When a rapid and time critical response is needed there is often a progression from asynchronous to more synchronous coordination mechanisms. The following quote

illustrates coordination and coordination tool choices based on highlighting synchronicity as well as accountability and privacy concerns:

> Interviewer: When you are talking to [other team members] are you using instant messaging usually?
>
> Participant 3: Yeah, instant messaging usually, but then the work item. I like using the work item because it keeps an audit trail of the discussion that is public. But if it's an emergency or it might be embarrassing I talk over instant messenger. I don't call people unless I have to, I usually only call people when they ask me to call them instead of doing it over instant messenger.

These properties of coordination mechanisms were all important in differing coordination scenarios reported by the participants. The common element is that the qualities of coordination mechanisms are considered when making choices about which mechanism to use, and how to escalate a coordination scenario when the resolution is not being achieved in a timely fashion. There is a general awareness and consideration of the differing aspects of specific coordination, often communication, mechanisms and these properties are carefully weighed when selecting a coordination strategy.

4.3.4 Strategic Behaviours

This analysis noted several instances of strategic behaviour being used to achieve specific goals within the software project. We define strategic behaviour as behaviour that is used to accomplish a goal, but seems outside of normal or designed tool usage. These behaviours include timed communication, such as withholding information for a period of time, or carefully timing the release of information to ensure the continuation of a conversation. These behaviours can also be as simple as exerting social pressure by using a public forum, such as a work item

discussion, to add momentum and encourage another team to implement certain designs.

Strategic behaviours can be more overt, such as attaching a reference implementation when asking a dependent team to implement functionality. This can be helpful and facilitate collaborative development, but can sometimes be seen as overstepping code ownership and domain expertise boundaries. The target team can feel that they are being forced to make changes in an ad hoc manner without sufficient time to construct an appropriately engineered solution. An extreme case is when a team or developer intentionally breaks the build to force a reaction from other teams to achieve a desired outcome. This situation can occur when a dependent piece of work is late but the submitter has constructed a fix that will work in their local workspace using the expected contributions from the blocking team. When project wide integration is attempted, it fails, and it can appear that the blocking party is at fault. Participant 7 discusses a technique used to induce another team to implement desired functionality:

> Interviewer: If you submit a request to them to support something you need and you know that it's blocking you do you get a better response?
>
> Participant 7: I try to avoid having things block me. I sort of try to make it a blocker for them. If I deliver my stuff to the integration stream, the build is then failing for them, so they are forced to fix that. Whereas [my team's] stream is clean, because I have my patches in the [my team's] stream. I try to make it almost a blocker for them.
>
> Interviewer: If you deliver your prototype to [your team's] stream does that make it all the way to the integration stream?

> Participant 7: It makes it to integration, oh yeah. But the piece in [the other component] doesn't get delivered, so tests in integration fail.

These behaviours may be outside the normal or designed usage of coordination and engineering tools, and can even seem uncooperative, but their ultimate goal is to resolve a work item or to implement a feature. By accomplishing team goals or implementing requirements, the participants are coordinating, resolving dependencies, and driving the project forward. The adaptation and use of coordination tools in these ways is interesting because it may highlight a team process issue or a tooling deficiency.

4.4 Development of Theoretical Propositions

A preliminary set of theoretical propositions describing coordination was developed. These propositions describe interactions between constructs in the domain of software engineering relating to coordination. This was accomplished using a technique illustrated by Sjøberg, Dybå, and Anda (2008). First, constructs and common themes were extracted from the coding analysis. These were used as nouns or subjects in the propositions. Constructs were derived by reviewing the categories that were identified during the thematic analysis and noting the ones which seemed to be important concepts within software engineering. A summary of constructs used in our propositions is shown in Table 1.

Construct	Description
Contributor	Someone who contributes to the project in the form of code, work items, work item comments, designs, etc.
Team	A logical group of contributors with a specific responsibility.
Work item	An artifact representing a piece of work that needs to be completed. Can be a bug, task, enhancement, or

	similar.
Resolution time	The time from work item creation to resolution.
Code	Source code that forms the executable software system.
Architecture	Design of the software system. Often in terms or the mapping or layering of components.
Plan	A schedule of work and commitments for a time period.
Work item dependency	A dependency relationship recorded between two work items.
Code/technical dependency	A technical dependency between pieces of code that will likely cause system failure if not met.
Proximity	Closeness, in geography, system architecture, or team structure.
Coordination activity	Activities such as commenting on work items, chatting over instant messenger, sending emails, holding meetings, or manipulating artifacts in the workspace that assist or achieve coordination goals.
Coordination tool	Tools that are used for coordination activities such as telephone, instant messenger, work items, SCM, wikis, design documents, source code.
Coordination needs	Necessary coordination activities.
Expertise	Knowledge of a software system, component, or issue.

Table 1: Summary of theoretical constructs

Theoretical propositions were then formed by reinterpreting the themes in combination with the contextual knowledge from conducting interviews and informal observation of the Jazz team. The reinterpretation noted interactions between constructs, identifying the relationships or influences between constructs to generate propositions. For example, one might note that developers in different roles need different levels of awareness of their team. Here, developers, roles, awareness, and team are constructs, and the proposition is formed by noting the interaction relationship between them. In this case the interaction noted would be that the level of awareness of others' work differs by role. In total, 9 propositions were developed.

As the analysis progressed, each proposition was noted along with possible explanations of the behaviour. Several propositions which appeared to be good candidates for further exploration and explanation using Jazz repository data were noted, along with proposed methods for quantitatively testing or exploring the proposition. These propositions are descriptions of how constructs in the domain of software engineering are expected to interact based on how this team was observed to behave. These propositions form a preliminary theory of how coordination occurs in the context of the team under study. Each proposition is presented below with an description and reasoning explaining why it might be occurring. Note that Proposition #5 is used in the second stage of this work, described in Chapter 5.

4.4.1 Propositions

Proposition #1: Technical dependencies, such as code dependencies, imply work item dependencies, but work items dependencies do not necessarily imply technical dependencies.

Explanation: Because use cases and requirements are expressed as work item artifacts and source code change sets are attached to implementing work items, the work items become interlinked using relations such as *blocking/depends on* and *related* in a pattern that corresponds with code dependencies. It follows that some of the structure of the work items reflects the architecture of the implemented system. There will also be other work items and work item dependencies that reflect other coordination needs, such as scheduling, that are not represented in code.

Proposition #2: Coordination activities are higher for work items with technical dependencies on other work items.

Explanation: When there is an increased need for coordination due to code level dependencies, there is more communication through work items and other engineering artifacts to resolve those dependencies. Since participants stated they used work items as a primary repository for recording and discussing design decisions, it follows that the dependencies between design activities will match the technical dependencies in code.

Proposition #3: Contributors working on higher layers in the software stack will have higher coordination needs of their work items than participants working on lower level components.

Explanation: Components at higher layers in the software stack or higher in the software architecture tend to have more dependencies on the services and interfaces of lower level components. Since there are more dependencies, either direct or indirect, there is greater need to coordinate to resolve these needs.

Proposition #4: There is more recorded coordination activity for work items with geographically distributed or cross-team membership.

Explanation: Collocated members report having much of their awareness and project knowledge acquired through informal means such as lunchtime discussions, informal hallway conversations, and by attending meetings. This type of activity is not generally noted as coordination activity, and is not recorded in the software engineering repositories. In contrast, participants noted that when coordinating work with remote members the Jazz repository served as a primary mechanism for recording rationale, decisions, and commitments.

Proposition #5: Specific structures of explicit dependencies between work items reduce the resolution times of those work items.

Explanation: Participants noted that using relations between work items increases navigability and traceability or work items, and specific patterns of dependency management were noted in the thematic analysis was used to better manage interdependent work items. These structures promote information flow between contributors to those work items and this increase in shared knowledge may improve coordination and lead to lower resolution times.

Proposition #6: Scheduled and planned work items, such as new features, are resolved more effectively than unplanned work items, such as defect reports.

Explanation: Open planning and published schedules enable awareness and foster team coordination of technical dependencies earlier in each development milestone. Participants noted that they tend to try and assess risk and complexity of work items and plan risky or complicated work items at the beginning of milestones. This allows the maximum amount of time to resolve the interdependence between work, especially where significant coordination is necessary, in an attempt to ensure these work items are completed on schedule. Bug reports or enhancement requests are often assigned and triaged as they are created, but do not have explicit planning and resource allocation, and have longer resolution times.

Proposition #7: Higher geographic, organizational, and structural distance (low proximity) increases time to resolution due to increased coordination needs.

Explanation: Geographic proximity has been shown to affect general communication and collaboration abilities. Team organizational and structural proximity have similar effects due to culture shifts, technical knowledge, and coordination ability of participants.

Proposition #8: When in times of crisis, coordination tool choices tend towards higher bandwidth, and synchronicity to mitigate distance (low proximity).

Explanation: Participants seek higher synchronicity, bandwidth, privacy, or richness, due to increased communication needs in time of crisis. Participants described common escalation patterns of communication when crises occur which started with common, but generally asynchronous tools such as work item comments, or wiki pages, and how they moved to instant messenger, group chat, or in extreme cases, to individual phone and group conference calls.

Proposition #9: Expertise recommendations can be made using contributors to previous work items that changed code in the same component.

Explanation: Participants noted that code responsibility, expertise, and component or subcomponent level dependencies change infrequently, thus there is little concern for expertise location systems, even in relatively large systems. Using the contributors (commenters, owners, code contributors, and subscribers) from previous work items that have participated in the resolution of work items touching the same components or code areas will result in good recommendations.

4.4.2 Proposition Summary

The preceding propositions describe behaviours and phenomena that occurred in the project under study while coordinating interdependent work in a software engineering project. The propositions are summarized in Table 2, below. These propositions describe ways in which constructs in the domain interact, and what the outcomes of those interactions may be; they are theoretical descriptions of the behaviours of a coordination system. As theoretical propositions, these can be used for predictions, reasoning, or further study to refine and confirm their validity and applicability. As a first step towards further refinement, Proposition #5 is explored quantitatively the second stage of this research. Further analysis of all propositions is beyond the scope of this thesis. Proposition #5 was chosen because it seemed important, given the frequency that participants mentioned their use of patterned work item and relationship management strategies, described in Section 4.3.2.

Proposition	Description
1	Technical dependencies, such as code dependencies, imply work item dependencies, but work items dependencies do not necessarily imply technical dependencies.
2	Coordination activities are higher for work items with technical dependencies on other work items.
3	Contributors working on higher layers in the software stack will have higher coordination needs of their work items than participants working on lower level components.
4	There is more recorded coordination activity for work items with geographically distributed or cross-team membership.
5	Specific structures of explicit dependencies between work items reduce the resolution times of those work items.
6	Scheduled and planned work items, such as new features, are resolved more effectively than unplanned work items, such as defect reports.
7	Higher geographic, organizational, and structural distance (low proximity) increases time to resolution due to increased coordination needs.
8	When in times of crisis, coordination tool choices tend towards higher bandwidth, and synchronicity to mitigate distance (low proximity).
9	Expertise recommendations can be made using contributors to previous

	work items that changed code in the same component.

Table 2: Summary of theoretical propositions

4.5 Summary

This chapter describes the analysis technique and findings from the qualitative analysis of 7 interviews with stakeholders that were conducted in August 2007. After a period of 3 months of informal observation and interview development, the interviews were conducted by the author with members of a professional software development team. Coding techniques drawn from grounded theory, and computer aided qualitative data analysis tools were used to code transcripts of interviews to generate themes.

Four major themes of proximity, authoring patterns, properties of coordination mechanisms, and strategic behaviours were developed. From these themes, and by reasoning about the interaction between actors, tools, and artifacts in the software engineering environment, theoretical propositions about the nature of coordination in software development were formed. These propositions describe coordination in software engineering and can be refined and augmented with further investigation. As an initial step in this refinement process, Proposition #5 is explored in Chapter 5 using a quantitative statistical analysis approach.

CHAPTER 5:
REPOSITORY ANALYSIS AND FINDINGS

5.1 Introduction

In order to refine the theoretical propositions developed in the first stage of this research, and to provide more insight into the behaviours and techniques used by software developers to manage dependencies, we chose one proposition developed from the thematic analysis and theoretical proposition development in Chapter 4 for further analysis. This stage of the research allows greater insight into the proposition developed from participant interviews by integrating a second data source and analysis method. We chose to further analyze Proposition #5 from the qualitative analysis, using a work item based statistical analysis of the relationship between explicit work item dependency structures and work item resolution times. For reference the proposition is repeated below:

> ***Proposition #5: Specific structures of explicit dependencies between work items reduce the resolution times of those work items***

Proposition #5 is interesting because further investigation can provide insights into the ways that dependency relations are created and the resolution time or effort required to complete that work. By creating dependency relationships in a work management database, software developers are creating explicit dependency representations of real life work dependencies. Many real world dependencies may never become represented in the work management database. However, understanding the relationships between the articulation and recording of explicit dependencies between work items, theory of coordination, as well as tools and

practices, can be developed that enable effective work management. The data necessary to conduct a dependency network analysis was also accessible and appropriate using our data extraction tools described in Chapter 3.

To approach the investigation of Proposition #5 we built work item dependency networks as graphs and computed a number of work item attribute, network structure, and communication measures. Then, using work items as the unit of analysis we analyzed the work item's relation to resolution times using statistical tests. The results of this quantitative analysis are discussed and related to the thematic analysis that was used to develop the proposition. Finally, we show that there is a small, but statistically significant, relationship between network centrality measures and work item resolution times, but that none of the measures derived from work items can explain a large proportion of the variance in resolution times.

5.2 Repository Data Analysis Approach

After choosing Proposition #5 for further exploration, analysis of the Jazz repository data consisted of several stages: data extraction, calculating analysis measures, and statistical analysis relating explicit work item dependencies to resolution times. Initially, information from the Jazz repository of the Jazz development team was extracted. Work items, their attributes, and the relationships between them were extracted using a plug-in to the Jazz environment. Since we do not have access to the Jazz server or underlying relational database that is used for persistence, we created a plug-in which uses the Jazz client-side Java API to gather instantiated objects of artifacts that interested us. This is the same API that would be used by the client portion of the Jazz tools to build a user interface or other Jazz based tools. Our plug-in conducted a search for all work items and then all

relationships between those work items. It then serialized the work item and relationship objects to XML. This XML was used as the source information that was deserialized and imported into a research oriented relational database.

From Proposition #5, we conjectured that the attributes of work items, the structure of the network of dependencies, and the amount of communication across dependencies may have an influence on the time to resolution of those work items and developed a set of analysis variables to address each of these areas. These variables were calculated and derived from the work items extracted from the Jazz team's repository.

Variables that quantifies the structure of the network of dependency relationships between work items are network measures drawn from social network analysis techniques described by Wasserman (1994). Our application of these measures was inspired by recent work by Zimmermann and Nagappan (2008) who used social network analysis measures to analyze the relationship between dependencies between software components and software failure rates. Social network analysis measures were chosen because we suspected that dependency networks may reflect the communication structures stakeholders would use to resolve dependencies. In addition to network measures of the dependency graphs of work items, we also derive and calculate several measures of the amount of recorded communication that occurs over work items dependencies.

Analysis of the extracted and derived variables in relation to the resolution time for each work items was approached using simple statistical analysis techniques. An exploratory path of analysis was followed, starting with the simple techniques and increasing complexity of investigative techniques as the investigation became increasingly refined. A correlation analysis technique was used first, testing

between each independent variable and the dependent resolution time variable. This was followed by a difference of medians tests of resolution times between work items in with dependencies and work items without dependencies, and then a difference of medians test for each independent variable across quantiles of resolution times.

The remainder of this chapter describes the process of developing work item dependency networks, the independent and dependent variables, and each statistical test and result. All of the variables are computed, and the statistical tests are conducted using a series of Perl and R scripts.

5.3 Finding Groups of Interdependent Work Items

To enable measurement of the structure of work items and dependencies between work items from the Jazz repository, we extracted and identified all of the work items and relationships between work items. In a Jazz repository, all work items can be linked or related to other work items using several types of relationships. These relationships include “blocks/depends on”, “related”, “parent/child”, and “duplicate of/duplicated by”. The work item relationship structure of the Jazz database can be conceptualized as a graph by using the set of work items as vertices and set of relationships as edges. In this case, the set of “blocks/depends on” directed edges are used to construct the graph.

A breadth-first search was conducted to identify all the components of the graph. Components are sets of vertices that are connected with edges. Figure 6 illustrates an example work item dependency graph component. The work items are connected to other work items in the component using “blocks/depends on” edges, or a dependency relationship. The network measures described in the following

sections use the graph or network structure we are identifying here using the "blocks/depends on" dependency relationship, and are subsequently applied to each component. In particular, we are interested in determining the connectedness of the graph, how many components composed of at least 2 work items exist, and what the population of graph components look like. Components of size 1 are singular work items with no explicit dependency relationships to other work items.

After conducting a breadth-first search using the "blocks/depends on" relation 490 graph components (or groups of interdependent work items) of size 2 or more, were identified. There are 1,169 work items with dependencies, (work items in components of size greater than 1), and there are 23,469 work items without dependencies (work items with component size of 1). Figure 7 shows a histogram of the component sizes for work items that have explicit dependencies.

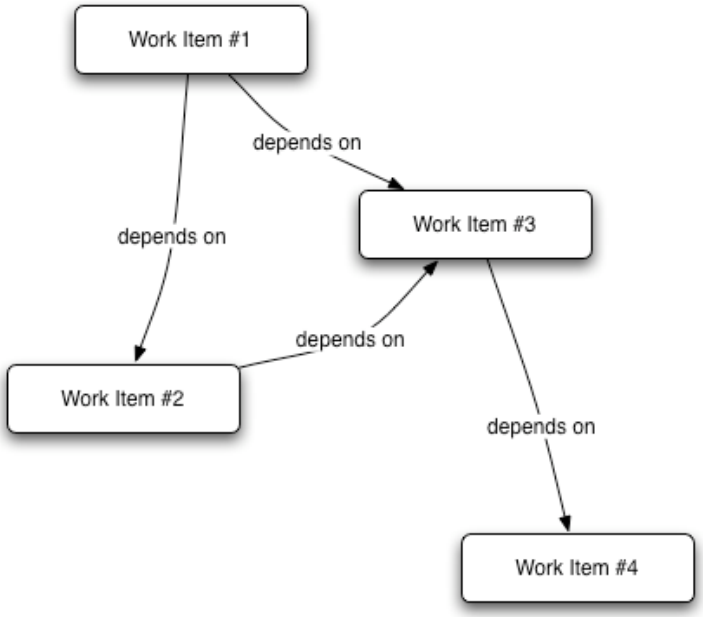

Figure 6: Example dependency component

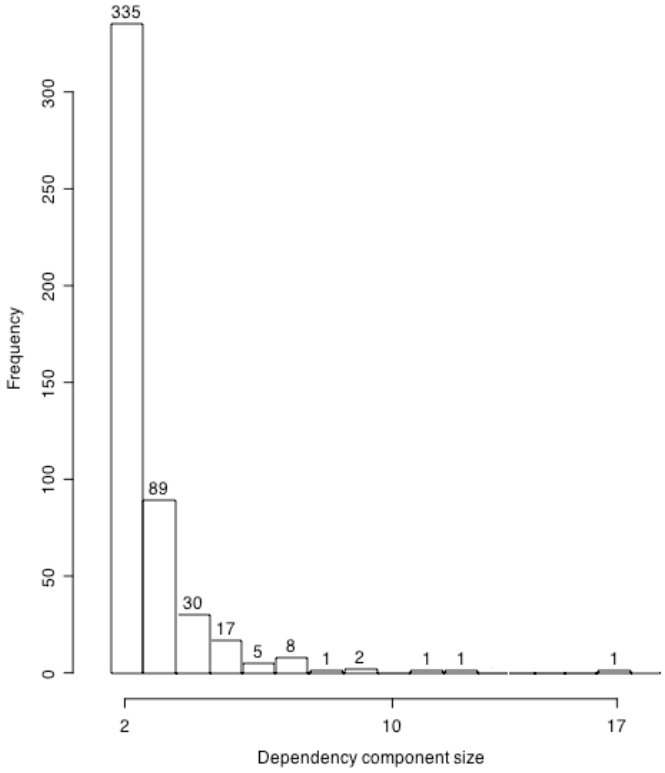

Figure 7: Work item dependency component size histogram

5.4 Independent and Dependent Analysis Variables

To conduct statistical analysis, independent variables that are thought to contribute to the variance or behaviour of the dependent variable were extracted or derived in three categories: 1) work item attributes, 2) dependency network structure measures, and 3) communication across dependency measures. The dependent variable of resolution time is also derived from repository data. Throughout this analysis we use the work item as the unit of analysis. In cases where network measures would apply to a group of interdependent work items (i.e., a graph component), the scores are applied to all work items in the respective component. Each of the independent and dependent variables are described in the following subsections.

5.4.1 Work Item Attribute Variables

Work item attribute variables are direct elementary attributes of work items as extracted from the Jazz repository. The work item variables are outlined in Table 3, below.

Variable	Description
Number of contributors	The number of contributors for this work item, including the creator, owner, commenters, code contributors, and attachment contributors .
Number of teams	The number of distinct teams the contributors are assigned to.
Number of sites	The number of geographic sites the contributors are from.
Number of comments	The number of comments on the work item.
Category	The category, or functional area of the work item.
Reduced category	The category of the work item at a depth of 1 in the categorization tree.
Number of change sets	The number of change sets attached to this work item.
Number of attachments	The number of other attachments, such as screen shots, attached to this work item.

Table 3: Work item variables

5.4.2 Network Structure Variables

Network structure variables are measures of the structure of the dependency network. In this research we draw many of the network structure measures from social network analysis measures. Since dependency structure may mimic the communication channels which are used to resolve them, it seems that social network analysis measures such as centrality and brokerage would be suitable for characterizing the structure of the dependency network.

It should be noted that most network structure measures, including those used here, do not yield descriptive or interesting measures until the size of the connected network reaches at least 4 work items. In this analysis we use these network structure variables while restricting their application to work items in dependency

components of size 4 or greater. The size of the component is defined as the number of work items in the component.

The most basic network measures are those which measure the size and density of the dependency component. These are described in Table 4.

Variable	Description
Component size	The number of work items in this component. = n
Component edge count	The number of relationships in this component. = m
Potential edges	The number of possible relationships in this component. = (n(n-1))/2
Edge density	The density of relationships in this component. = m/Potential Edges

Table 4: Network size and density

In social network analysis, it is common to consider network measures at the scale of the entire network and at scale of the ego network. The ego network is the sub-graph which includes a target vertex, its immediate connected neighbours, and all edges between any members of that set of vertices. The ego network is used to limit the size of a graph under analysis and to provide measurements within the neighbourhood of a vertex of interest. In our dependency graph data set, the graph is composed of many disconnected components so applying network measures to ego networks is not appropriate or necessary to limit the size of the graphs. Each network measure is naturally limited by the size of the graph components. However, the need to compare network measures across components is necessary. To achieve this comparability, each network measure recorded as a raw and normalized value. The measures are normalized by dividing the score by the number of work items in the component.

Centrality is a network measure that is commonly used to measure the importance of a node in a social network. In the context of a work item dependency

network centrality may denote the relative importance of a piece of work as a work item that blocks many other work items would be very important and would have high degree centrality. Several common measures of centrality that are used in this analysis of dependency networks are described below.

Degree centrality is the simplest form of centrality. It is defined as the sum of the in degree and out degree of a vertex Freeman (1979). The degree centrality measures the importance of a work item based the number of ties it has to other work items. In our analysis, we use both the in-degree and out-degree measures. In-degree denotes the number of work items which are blocking this work item, or that this work item is waiting on before it can be resolved. The out-degree denotes the number of work items which this work item blocks, or the number of work items that cannot proceed until this work item is resolved.

Closeness centrality is a centrality measure that takes the distance of other work items in the dependency graph into account. Several different variations of computing closeness are used in our analysis:

- Closeness centrality is the sum of lengths of the shortest paths from a work item to all other work items. Increased closeness centrality implies that a work item has higher importance or dependencies. This indicates how many dependencies would need to be traversed to get to this work item from another work item. We use the definition provided by Freeman (1979).
- Distance weighted reach centrality is number of work items that can be reached from this work item, each weighted by the reciprocal of the distance (Wasserman and Faust 1994). This shows how many other work items are reachable from this work item, weighing further work items less.

- Eigenvector centrality is a relative score based on the scores of other nodes in the graph. Work items that have greater dependency on other work items in the network contribute more to the score of the work item (Bonacich 1987; Katz 1953).
- Information centrality is a measure of the bandwidth of individual vertices in the dependency graph. Highly information-central work items tend to have a large number of short paths to many other work items within the dependency structure. Information centrality is calculated as the harmonic mean of paths that end at the work item. This measure values longer paths more greatly (Wasserman and Faust 1994; Stephenson & Zellen 1989).

Betweenness centrality is another common measure of the centrality of a vertex in a graph (Freeman 1979). Betweenness quantifies the degree to which a work item is between other pairs of work items. High betweenness of a work item can be seen as an important for resolving dependence due to the number of short paths which pass through it.

Two-step reach is the percentage of the network which can be reached from a work items in two steps or less (Wasserman and Faust 1994). Two-step reach quantifies how easy it would be to find dependent work items through short paths. We calculate two-step reach at a component level and apply the measurement to all work items in that component.

Brokerage is the percentage of all possible work item pairs in a network that are not reachable (Gould and Fernandez 1989). Brokerage represents the ability to broker or pass information between work items. We calculate brokerage at a component level and apply the measurement to all work items in that component.

All centrality, reach, and brokerage measures are computed using R scripts as implemented by the *sna* package (Butts 2007) and are summarized in Table 5, below.

Variable	Description
In degree	The number of work items that block this work item, or that this work item is waiting on before it can be resolved.
Out degree	The number of work items that are blocked by this work item, or the number of work items that cannot proceed until this work item is resolved.
Closeness centrality	The sum of lengths of the shortest path to a work item from all other work items. This indicates how many dependencies would need to be traversed to get to this work item from another work item.
Distance weighted reach	The number of work items that can reach a work item with the sum of at each distance weighted by the reciprocal of its distance. This shows how many other work items are reachable from this work item, weighing further work items less.
Eigenvector centrality	A relative score based on the scores of other nodes in the graph. Work items that have greater dependency to other nodes in the graph contribute more to the score of the work item.
Information centrality	The harmonic mean of paths that end at the work item. This measure values longer paths more and shows how many short dependency paths this work item has.
Betweenness centrality	The number of shortest paths between this work item and other work items. This measure shows how many work items this work item is between using dependency relations.
Two-step reach	The number of work items that are two steps away from each other. This shows how long the average path to find another dependent work item would need be.
Brokerage	The number of work item pairs that are not directly connected to each other. This shows how many dependency paths must travel through broker work items.

Table 5: Network structure variables

Each of the work item network measures described above is recorded as a raw and normalized value. The scores are normalized by the size of the component (number of work items) the work item is in. As well, the mean of the raw and the

normalized scores of each work item across the whole component are calculated and recorded for each work item.

5.4.3 Communication Variables

Communication across or around dependencies is a technique is used to coordinate the resolution of those dependencies. In order to measure the amount of communication across dependencies in work item dependency components, several variables are derived from the work item data that quantify the amount of co-commenting that occurs in dependent work items. We use commenting on work items as a source of communication information because it was reported as the primary mechanism of communication, discussion, and debates about design, implementation, and defects. These communication measures use comments and quantify the information transfer across dependencies. Each of the communication variables is described in Table 6.

Variable	Description
Number of co-commenters	The number of contributors who comment on both work items in a single dependency relationship.
Number of 2+ commenters	The number of contributors who comment on at least 2 work items in a dependency component.
Average work items commented	The average number of work items that contributors comment on in each component. A measure of the completeness of communication by commenters.

Table 6: Communication variables

Each of these communication variables is recorded as a raw and normalized value. The measures are normalized by dividing the score by the number of distinct contributors involved in the respective work item dependency component. This gives a measure of the degree of communication participation for each dependency component population.

5.4.4 Dependent Variable

To investigate the relationship between the explicit dependency structure of a work item and the resolution time of the same work item, we need to calculate and define the resolution time of each work item. The difference between the creation time of the work item, and the time when the work item is transitioned to the "Resolved" state and a status of "Fixed" is used to calculate the resolution time. We use the first time the work item has been transitioned to the "Resolved" state and "Fixed" status in any cases where the work item is reopened or has later changes. The dependent variable is summarized in Table 7.

Variable	Description
Resolution time	The difference between the resolution date and the creation date.

Table 7: Dependent variable

5.5 Data Cleaning for Analysis

After extracting work items from the production Jazz repository, importing them from serialized XML into a research database, and identifying components, ineligible and erroneous data points were removed. Each cleaning step is described below and the results of each cleaning step are summarized in Table 8.

The first two steps removed work items that are ineligible for this analysis because they are not resolved and fixed. We removed the unresolved work items, as they do not have a resolution date and it is not possible to calculate resolution times. Then we removed all of the work items which have do not have a resolution status of "Fixed". A work item that is in the resolved state also has an associated resolution. This resolution can be "Fixed", or a number of other outcomes such as "Not Fixed", "Won't Fix", "Invalid", and so on. For simplicity, we only consider work items that are "Fixed" to be eligible for analysis.

Then the remaining eligible work item resolution times are checked to be a reasonable value. Some work items were found to have negative resolution times and were removed from the data set. Investigation revealed that some of the negative times were caused by time zone differences between the Jazz Eclipse client and the Jazz server. It is suspected that in some cases, such as setting the work item creation time, the client's clock is used and in other cases, such as the setting a resolution time, the server's clock is used. Given that there are negative timing errors, it is also possible that there are positive timing errors that we were unable to detect. Further, we could not characterize all negative resolution timing errors as clock synchronization errors, so there exist negative timing errors from another source other than time zone differences that we cannot identify or characterize. These timing errors are further discussed as a threat to the validity of this quantitative analysis in Chapter 6.

Finally, all work items that have unknown contributors, such as owners, creators, commenters, or change set contributors are removed. During data extraction from the Jazz server a list of contributors and their geographic location was provided, but was not complete. Since we count the number of sites or locations where a work item's contributors are located, we cannot include work items that have unknown contributors. This limits our data set to those work items where we can identify all of the contributors.

Work items were not filtered or categorized based on the work item type, such as "task", "enhancement", or "defect", as participants noted that that field is not reliable as each new work item defaults to defect upon creation. There is the possibility that work item type would have some relation to resolution times. Further, it may be that work items that are enhancements naturally have more dependencies and take

longer to execute. Unfortunately, we do not have reliable data to explore or control this variable.

Data Cleaning Stage	Number of work items	Percent of eligible work items	Number work items with dependencies	Percent of eligible work items with dependencies	Ratio of work items with and without dependencies
Original extraction	24,638	-	1,169	-	5%
Keep only resolved work items	14,033	-	654	-	5%
Keep only fixed work items	9,029	100%	565	100%	6%
Keep only non-negative resolution times	5,717	63%	473	84%	8%
Keep only work items with known contributors	4,753	53%	406	72%	9%

Table 8: Summary of data set sizes during data cleaning

5.6 Relationship between Explicit Dependency Structure and Resolution Time

Statistical analysis approaches were used to explore the relationship between work item explicit dependency structure and resolution times. Non-parametric statistical methods were used because the distribution of the resolution time dependent variable does not follow a normal or other obvious distribution. Thus, it is necessary to use statistical methods that do not rely on assumptions about the distribution of the data (Siegel 1956; Maxwell 2002). All statistical tests were conducted using scripted analysis tools developed using the R statistical analysis package.

First, a Spearman rank correlation was performed between each independent variable and the dependent variable of resolution time to assess the amount of variance in the resolution time that could be explained by each variable. Second, a Mann-Whitney *U* test was conducted between the resolution times of work items with dependencies and work items without dependencies to assess whether the medians were significantly different. Third, a Kruskal-Wallis test of each independent variable across six quantiles of resolution times was conducted to determine if there was a significant difference of medians in the independent variables across quantile groups of resolution time.

5.6.1 Correlations between Independent Variables and Resolution Time

As a first investigation of the influence of work item attribute variables and dependency network variables on resolution times, we conducted a Spearman correlation between each independent variable and the resolution time. We conducted two sets of correlations. The first uses variables that apply to all dependency components of size 2 or more, and the second set uses network analysis variables that are only applicable for work item components of size 4 or more. The results are shown in Table 9 and Table 10, with the strongest correlations highlighted in bold.

Since none of the variables show even a weak correlation (*rho*>0.5) with resolution time, we can reason that none of the variables can independently explain a significant portion of the variance in resolution time. Though we cannot rule out interaction between variables to explain the resolution time. The *rho* measures are positive if an increase in the variable is correlated with increased resolution times, and is negative if it is correlated with decreased resolution times. For example, the Number of Contributors variable (*rho*=0.2219, *p*=6.35E-06) shows that the number

of contributors to this work item tends to be higher when resolution times are higher.

Work Items with Dependencies Component size >= 2 (*n*=406)	***rho* (>0.2 bold)**	***p* (<0.01 bold)**
Work Item Variables		
Number of Contributors	**0.2219**	**6.35E-06**
Number of Teams	0.1204	0.01519
Number of Sites	0.1087	0.02847
Number of Comments	**0.2703**	**3.13E-08**
Category	0.04571	0.3582
Category Reduced	-0.03950	0.4273
Number of Change Sets	0.1651	**0.0008374**
Number of Attachments	0.05600	0.2606
Number of Blocking (Out Degree)	-0.1052	0.03406
Number of Depends On (In Degree)	**0.2111**	**1.80E-05**
Number of Related	**0.2750**	**1.78E-08**
Number of Duplicated By	0.1872	**0.0001486**
Number of Children	0.1459	0.003222
Number of Parents	0.1195	0.01602
Number of Textual References To This Work Item	**0.2018**	**4.22E-05**
Number of Textual References From This Work Item	**0.2151**	**1.23E-05**
Communication Variables		
Component Average Number of Work Items Commented On	-0.1033	0.0375
Normalized Component Avg. Num. Work Items Commented On	-0.1743	**0.0004184**
Component Number of Commenters on 2+ Work Items	0.06681	0.1791
Normalized Component Number of Commenters on 2+ Work Items	-0.1389	**0.005064**
Component Number of Comments on Both Dependent Work Items	0.06065	0.2227
Normalized Component Number Comments on Both Dependent Work Items	-0.1497	**0.002496**

Table 9: Spearman correlations between variables and resolution time for work items in components of size 2 or greater

Work Items with Dependencies Component size >= 4 (*n*=114)	***rho* (>0.2 bold)**	***p* (<0.01 bold)**
Dependency Network Variables		
In Degree (Number of Depends On)	0.1839	0.05016
Out Degree (Number of Blocking)	0.08811	0.3512
Component Number of Work Items	0.1991	0.03369
Component Number of Links	**0.2130**	0.02284
Component Number of Possible Links	0.1991	0.03369
Component Link Density	-0.1747	0.06297
Centrality Closeness	**0.2184**	0.01955
Component Centrality Closeness	0.1248	0.1859
Centrality Distance Weighted Reach	0.09033	0.3392
Component Centrality Distance Weighted Reach	0.1608	0.08743
Centrality Eigenvector	-0.07780	0.4106
Component Centrality Eigenvector	-0.05358	0.5713
Centrality Information	0.07387	0.4347
Component Centrality Information	-0.08928	0.3448
Two Step Reach	0.08060	0.3939
Component Two Step Reach	0.15688	0.09553
Normalized Two Step Reach	-0.01674	0.8597
Component Normalized Two Step Reach	-0.1168	0.2160
Brokerage	0.1407	0.1353
Component Brokerage	0.1213	0.1986
Normalized Brokerage	0.1384	0.1420
Component Normalized Brokerage	0.09817	0.2987
Betweenness	0.1395	0.1387
Component Betweenness	-0.06376	0.5004
Normalized Betweenness	0.1359	0.1492
Component Normalized Betweenness	-0.063756	0.5004

Table 10: Spearman correlations between variables and resolution time for work items in components of size 4 or greater

5.6.2 Resolution Times of Work Items with and without Dependencies

To assess whether work items with dependencies have a statistically significantly different median resolution times than work items without dependencies we perform a Mann-Whitney *U* test between populations. Non-parametric statistical methods and a comparison of medians are used as the distribution of resolution times is not normal (Siegel 1956; Maxwell 2002). The test showed that the null hypothesis, that the medians of the two populations are the same, or that the presence of dependencies has no influence on resolution times, can be rejected (p=2.2e-16). Figure 8 shows a box plot of the work items with and without dependencies. Work items without dependencies have a median resolution time of 6.72 days and mean of 31.81 days. Whereas, work items in dependency components have a median resolution time of 22.10 days and a mean of 49.39 days. The difference in medians is a practically significant increase of 15.37 days for those work items with dependencies.

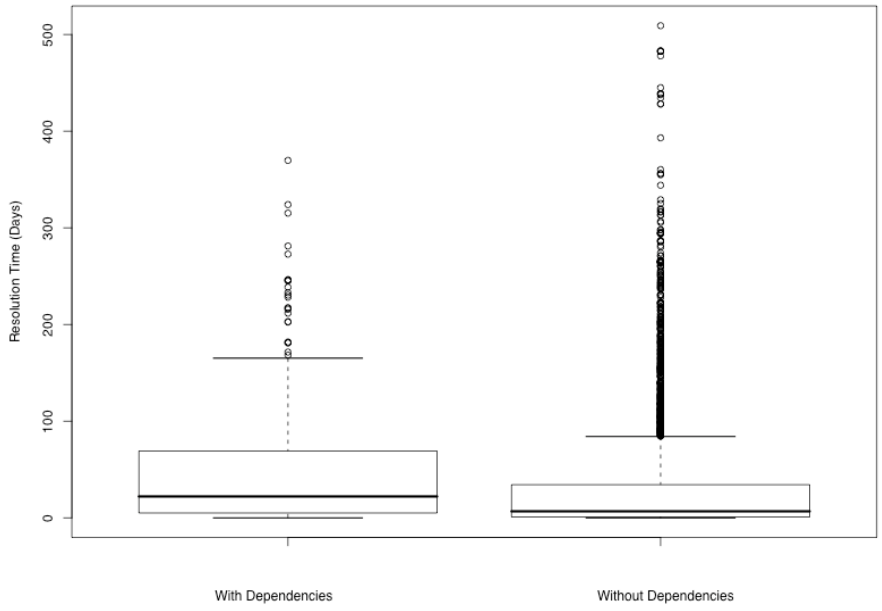

Figure 8: Box plot of work item resolution times with and without dependencies

To gain a sense of the distribution of work items with and without dependencies, see Figure 9 for a histogram of resolution times of work items with dependencies and Figure 10 for a histogram of resolution times of work items with no dependencies.

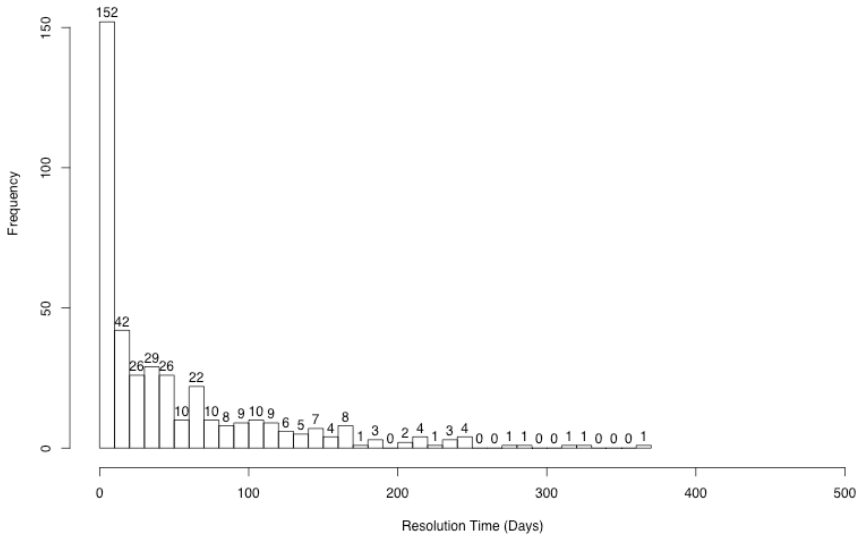

Figure 9: Resolution time histogram of work items with dependencies

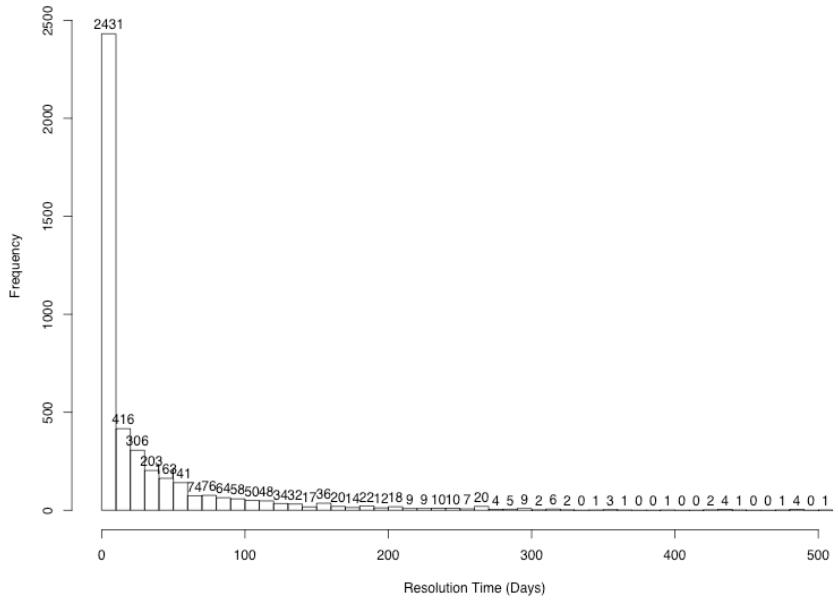

Figure 10: Resolution time histogram of work items without dependencies

5.6.3 Independent Variable Medians across Resolution Time Quantiles

Since we can see a significant difference in the median resolution times between work items with and without dependencies, we wanted to determine if the network structure of dependencies or communication measures have any influence on the resolution time of work items that have dependencies. First, we partitioned the work items with dependencies into 6 quantiles (groups of equal population) with 67 or 68 work items in each. This allows us to reason about classes of work items that have roughly logarithmically increasing windows of resolution times. We chose to have 6 quantiles because the resolution time boundaries at this level roughly map to the way that software developers estimate and plan work, such as estimating a task to take several days, one week, or a month. Figure 11 illustrates the distribution of resolution times into groups for work items with dependencies. The *x*-axis in each of

the following figures is labelled with the maximum number of days for each quantile.

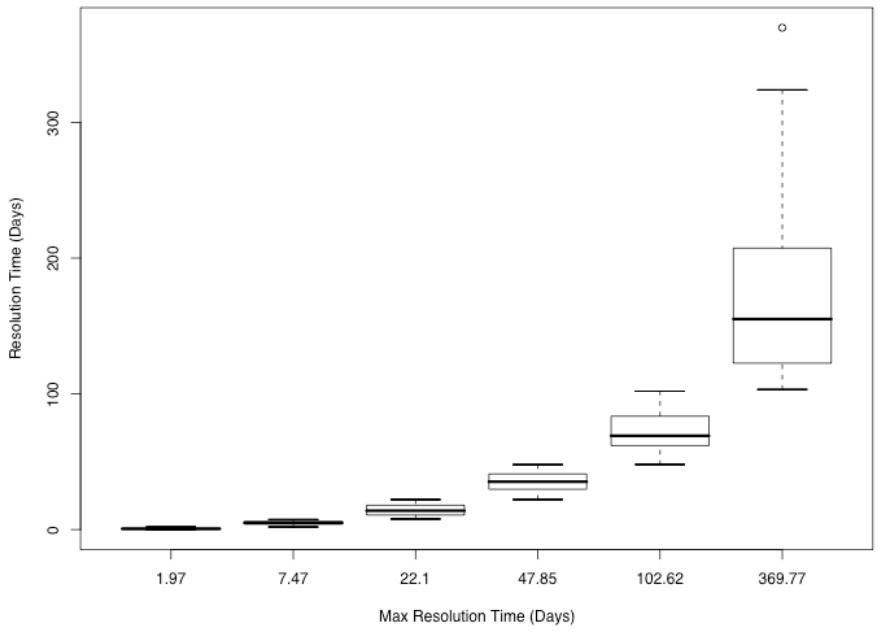

Figure 11: Box plots of resolution time quantiles for work items with dependencies

To assess whether any one independent variable has significantly different medians across the 6 groups, we tested each independent variable using a Kruskal-Wallis test between groups, evaluating at α=0.01 since we have a relatively large sample (n=406). We provide an adjusted p-value using Benjamini and Hochberg's (1995) technique for controlling the false discovery rate in multiple independent tests. Several variables are statistically significant, meaning that at least one of the 6 groups has a significantly different median. As with the correlation tests, we conduct two sets of Kruskal-Wallis tests. The first set uses variables that apply to all dependency components of size 2 or more, and the second set uses network analysis variables that are only applicable for work item components of size 4 or greater. The

results are summarized in Table 11 and Table 12 with significant variables (p<0.01) highlighted in bold.

Work Items with Dependencies Component size >= 2 (*n*=406)	***p-adjusted* (<0.01 bold)**
Work Item Variables	
Number of Contributors	**0.004005**
Number of Teams	0.1415
Number of Sites	0.5552
Number of Comments	**0.0001936**
Category	0.9577
Category Reduced	0.8474
Number of Change Sets	0.1159
Number of Attachments	0.8474
Number of Blocked Work Items (Out Degree)	0.01136
Number of Depends On Work Items (In Degree)	**0.0001936**
Number of Related Work Items	**0.007755**
Number of Duplicate Work Items	0.1159
Number of Children	0.1459
Number of Parents	0.3949
Number of Textual References To This Work Item	0.2182
Number of Textual References From This Work Item	0.01136
Communication Variables	
Component Average Number of Work Items Commented On	0.2182
Normalized Component Avg. Num. Work Items Commented On	0.03877
Component Number of Commenters on 2+ Work Items	0.03877
Normalized Component Number of Commenters on 2+ Work Items	0.1159
Component Number of Comments on Both Dependent Work Items	0.03877
Normalized Component Number Comments on Both Dependent Work Items	0.1159

Table 11: Kruskal-Wallis test between 6 groups for work items in components of size 2 or greater

Work Items with Dependencies Component size >= 4 (*n*=114)	***p-adjusted* (<0.05 bold)**
Dependency Network Variables	
In Degree (Depends On)	0.4370
Out Degree (Blocked)	0.8225
Component Number of Work Items	**0.03911**
Component Number of Links	**0.03911**
Component Number of Possible Links	**0.03911**
Component Link Density	**0.03911**
Centrality Closeness	**0.03911**
Component Centrality Closeness	0.05099
Centrality Distance Weighted Reach	0.6156
Component Centrality Distance Weighted Reach	0.6382
Centrality Eigenvector	0.7535
Component Centrality Eigenvector	**0.03911**
Centrality Information	0.2654
Component Centrality Information	0.5299
Two Step Reach	0.3646
Component Two Step Reach	0.4482
Normalized Two Step Reach	0.7427
Component Normalized Two Step Reach	**0.03911**
Brokerage	0.3646
Component Brokerage	0.4631
Normalized Brokerage	0.8225
Component Normalized Brokerage	0.1225
Betweenness	0.8225
Component Betweenness	0.2014
Normalized Betweenness	0.8225
Component Normalized Betweenness	0.9436

Table 12: Kruskal-Wallis test between 6 groups for work items in components of size 4 or greater

Each of the significant variables and other interesting variables are illustrated in the following sections with a box plot of the variable measure across all groups and an explanation of the behaviour.

5.6.3.1 Contributors and Comments

The number of contributors was shown to be significantly different (*p-adjusted*=0.004005) in at least one resolution time group. A box plot for each resolution time group is shown in Figure 12. It shows that the median for all resolution time groups is 2 except for the last group, for resolution times up to approximately 369 days where the median number of contributors increases to 3.

The comment count was also shown to be significantly different (*p-adjusted*=0.0001936) in at least on resolution time group. A similar pattern is shown in the comment count set of box plots across groups shown in Figure 13. The number of comments has a median of 4 in all groups except where work items take up to 2 days to resolve with a median of 2, and work items that take up to 102 days to resolve with a median of 4, and those that take up to 369 days have a median of 6. The range of the number of comments is increasing as the resolution times are increasing.

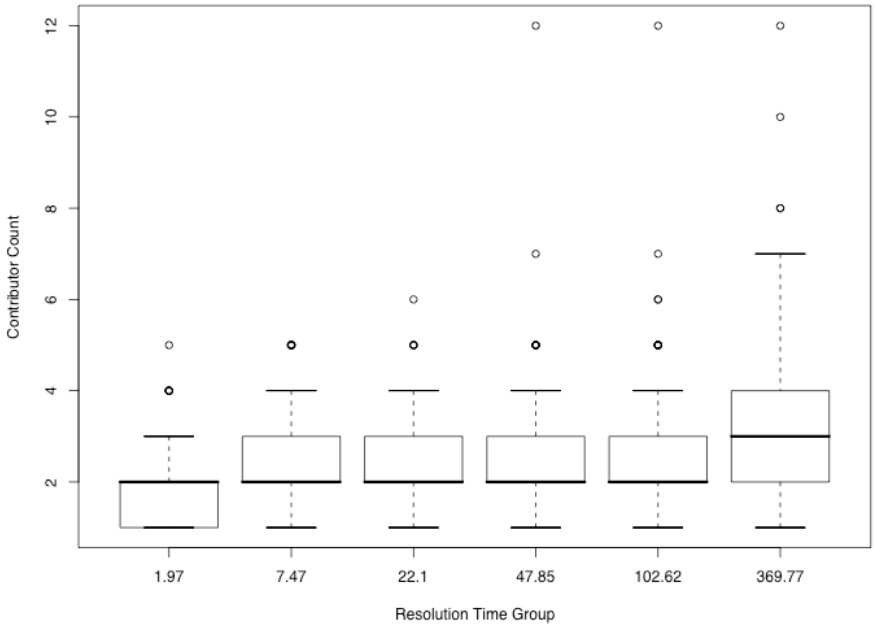

Figure 12: Contributor count box plots across groups

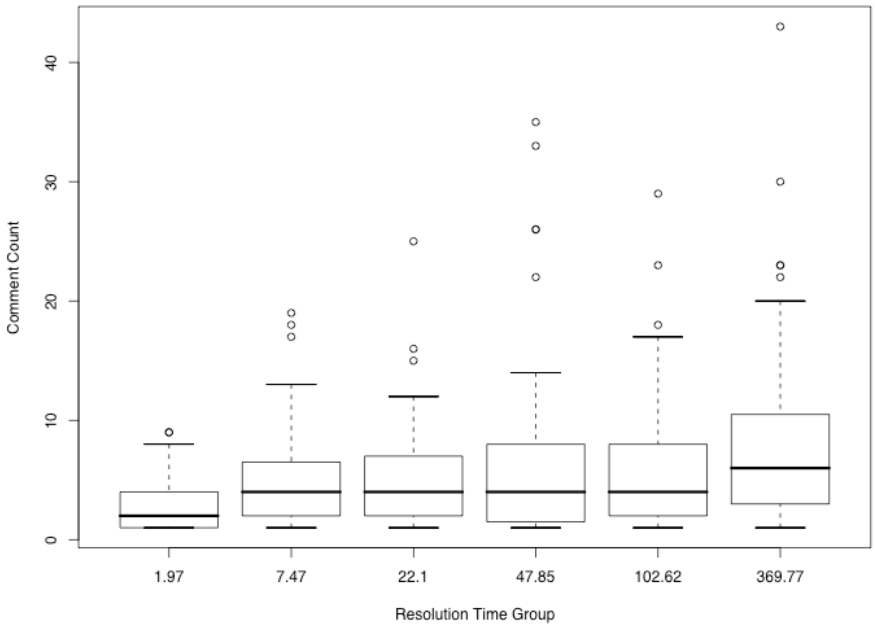

Figure 13: Comment count box plots across groups

5.6.3.2 Blocked and Dependent Work Items

The number of blocked work items was only shown to be significantly different in at least one group before adjustment (*p-adjusted*=0.01136). Box plots showing the number of blocked work items by resolution time group are shown in Figure 14. The median number of blocked work items is 1 for work items that have resolution times of less than 22 days, and the median is 0 for those with resolution times between 22 and 102 days. This matches the pattern of participants working on and resolving work items more quickly if they are blocking other work items. Though, there is another class of work items take up to 369 days to resolve that block a median of 1 work items.

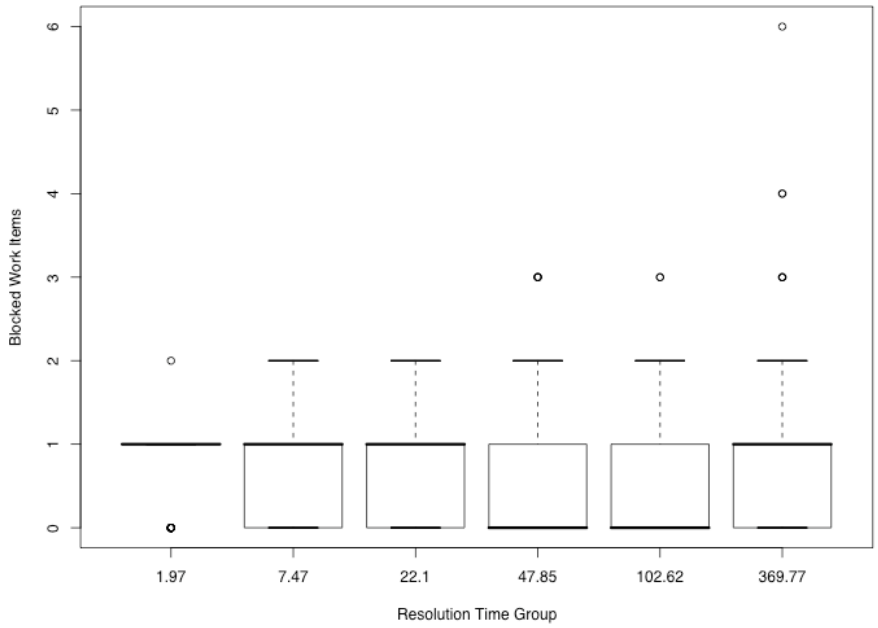

Figure 14: Blocked work item box plots across groups

The number of depends on work items was shown to be significantly different in at least one group (*p-adjusted*=0.0001936). Figure 15 shows that the median number of work items this work item depends on increases as the resolution time

grows. Work Items that take 7 or fewer days have a median of 0 work items upon which they depend. While for resolution time groups where the resolution time is more than 7 days the median number of work items that each work item depends on increases to 1.

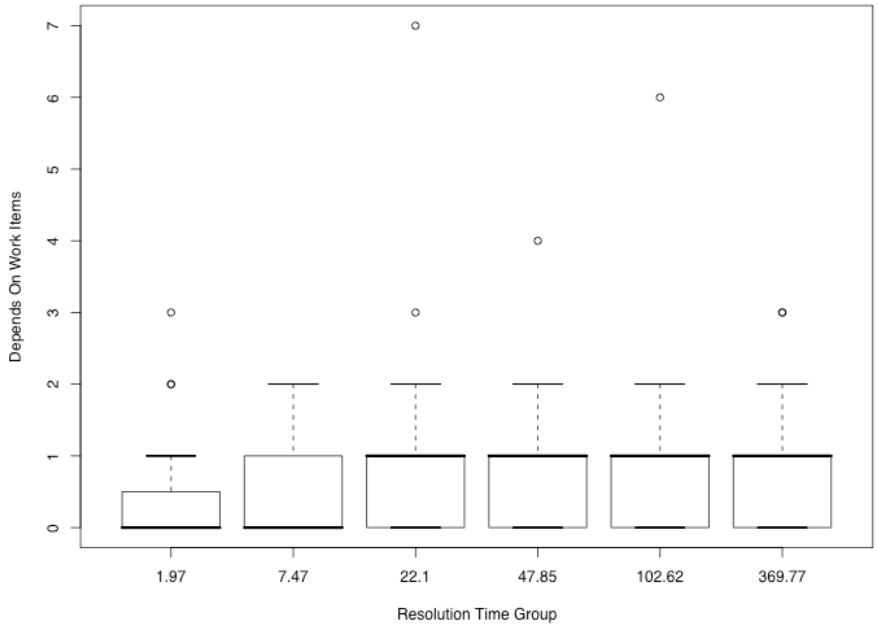

Figure 15: Depends On work items box plots across groups

5.6.3.3 Related and Textual References

The number of related work items (*p-adjusted*=0.007755) was shown to be significantly different in at least one group. The number of textual references from this work item to other work items was shown to be significant before adjustment (*p-adjusted*=0.01136). Figure 16 and Figure 17 show that the number of related work items and the number of textual references across groups medians increase as resolution times reach 102 and 369 days respectively. This shows that as work items age, the networks of relationships between work items grows larger, and in particular long living work items accumulate more referencing relationships.

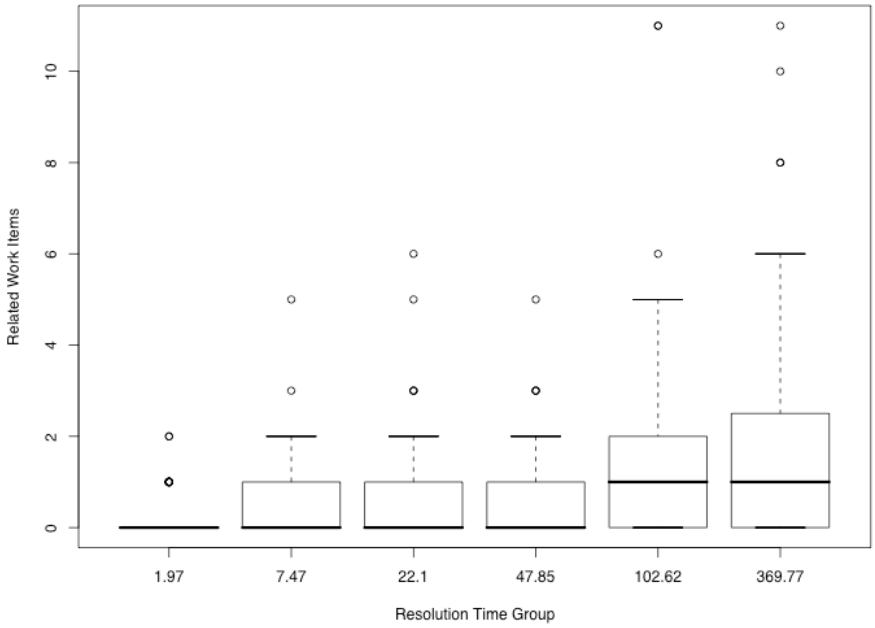

Figure 16: Related work items box plots across groups

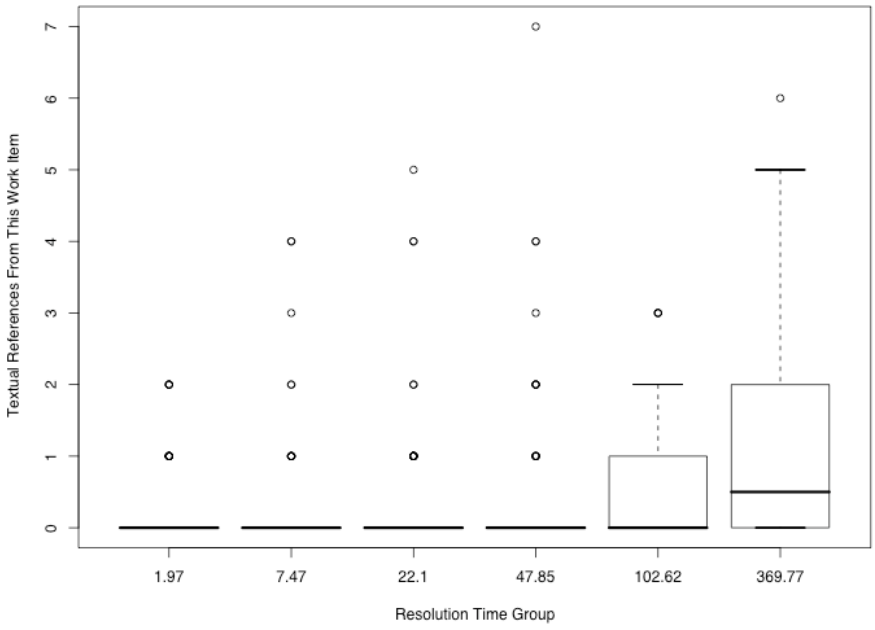

Figure 17: Textual references from this work item box plots across groups

5.6.3.4 Component Size and Dependency Link Density

When evaluating the dependency structure variables using the Kruskal-Wallis test, for work items in components of size 4 or greater, 4 variables relating to the size and density of dependency links in a component were significant (*p-adjusted*<0.05). The number of work items in a component (*p-adjusted*=0.03911), the number of dependency links in a component (*p-adjusted*=0.03911), the number of possible dependency links to form a complete network (*p-adjusted*=0.03911), and the dependency link density (*p-adjusted*=0.03911). These measures are illustrated below in Figure 18, Figure 19, Figure 20, and Figure 21.

Work items that take a long time to resolve have higher median dependency component sizes, and in turn have, overall lower median dependency link density in their respective components. The number of dependency links rises along with resolution time, though, work items with longer resolution times are in larger components that are less connected than work items with shorter resolution times.

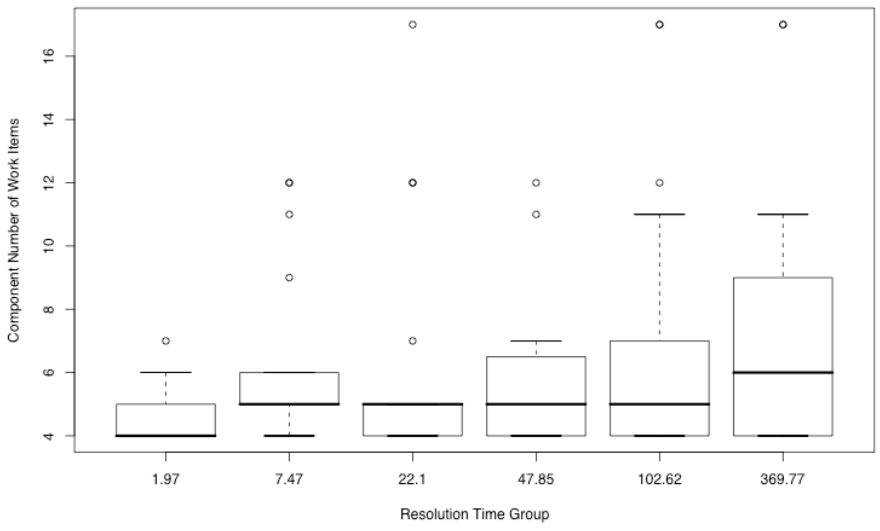

Figure 18: Component number of work items box plots across groups

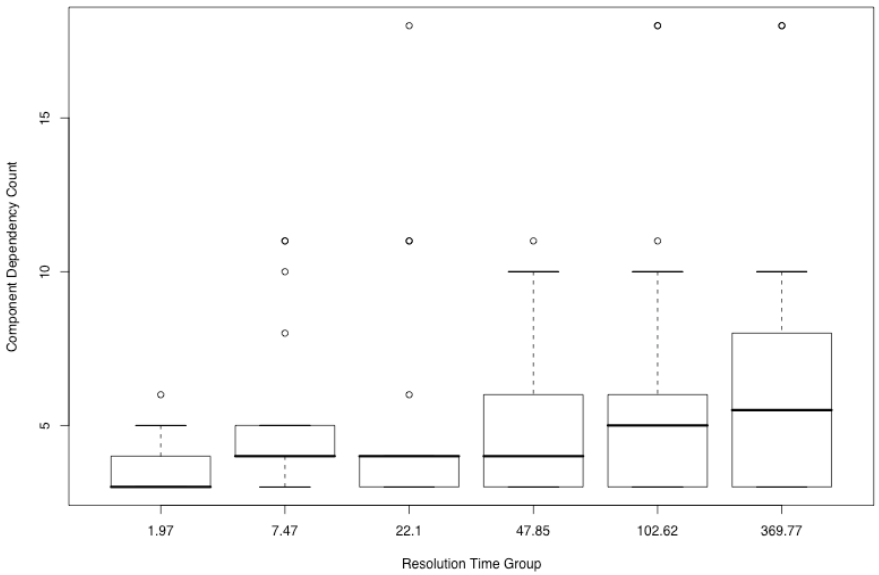

Figure 19: Component dependency link count box plots across groups

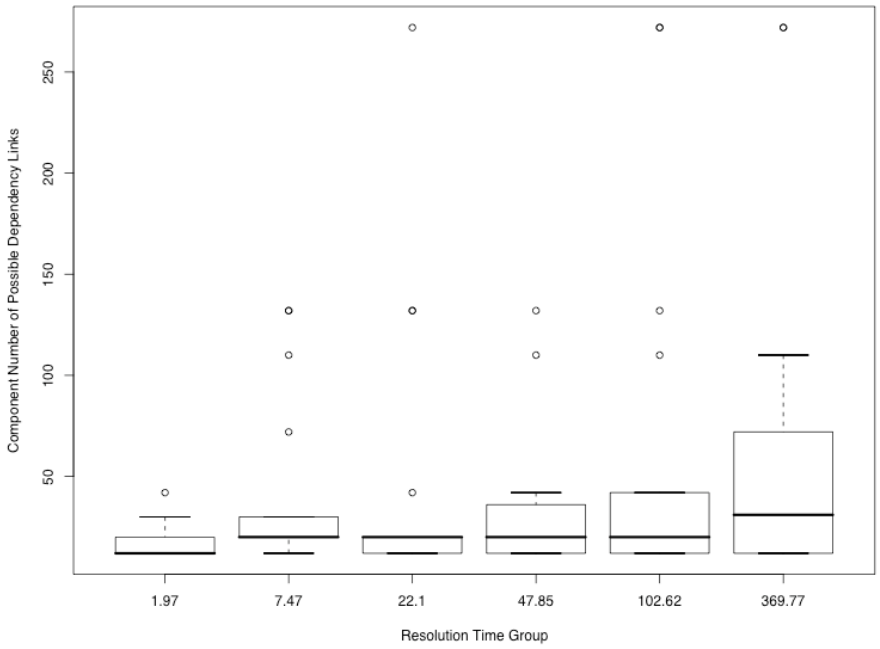

Figure 20: Component number of possible links box plots across groups

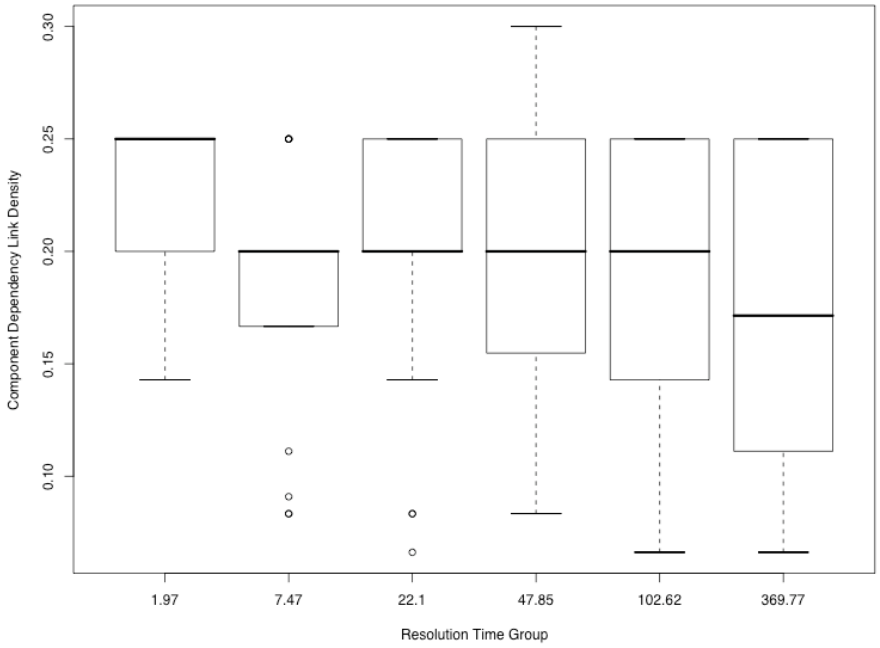

Figure 21: Component dependency link density box plots across groups

5.6.3.5 Closeness Centrality, Component Eigenvector Centrality, and Component Normalized Two-Step Reach

Figure 22, Figure 23, and Figure 24 show box plots illustrating closeness centrality, component eigenvector centrality, and component normalized two-step reach across groups. Closeness centrality and eigenvector centrality do not vary in a practically significant manner in our data set and do not provide any useful insight into the behaviour of resolution times. Component normalized two-step reach shows little variance between groups, but does have greater variance for work items with longer resolution times.

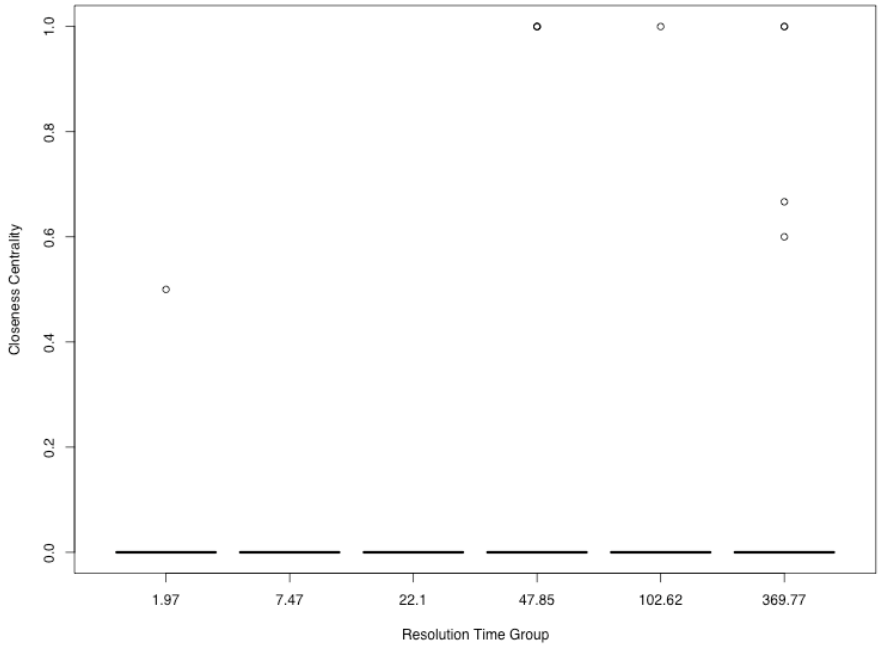

Figure 22: Closeness centrality box plots across groups

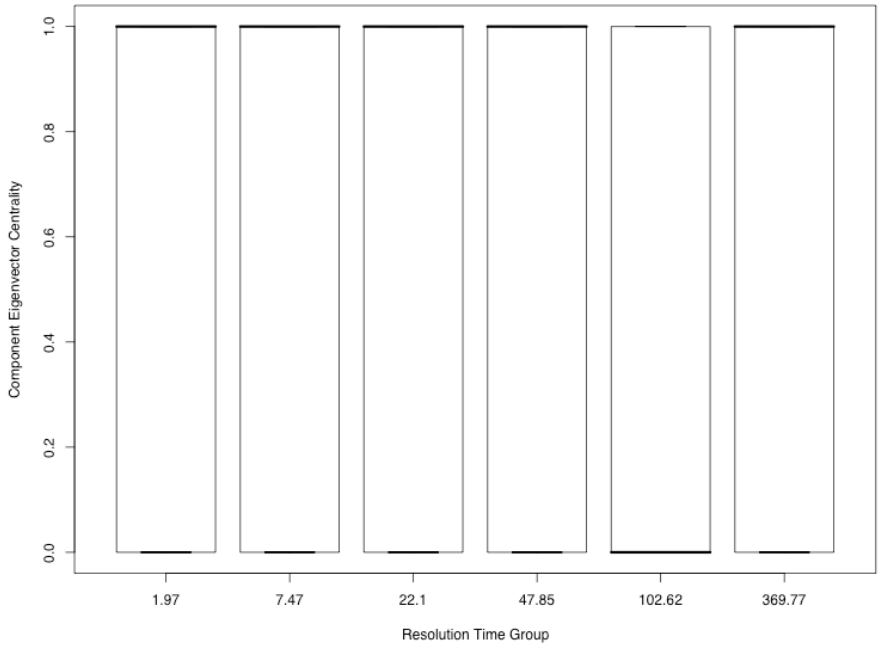

Figure 23: Component eigenvector centrality box plots across groups

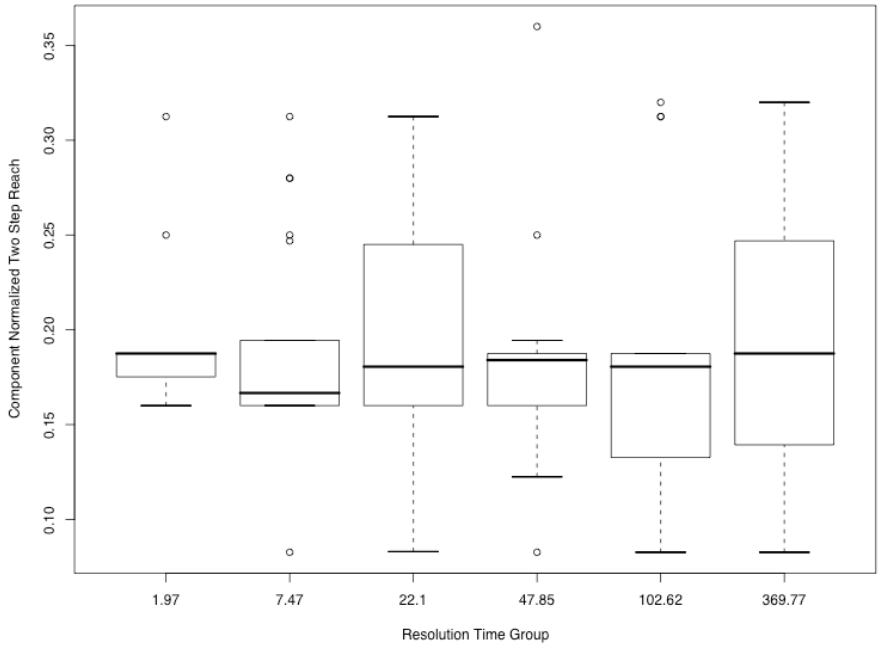

Figure 24: Component normalized two-step reach box plots across groups

5.7 Summary

In this analysis we have approached Proposition #5 from the previous qualitative analysis of interviews with developers, which suggested that explicit dependency structures were used to facilitate coordination and that there was likely some relationship between the structure of explicit dependencies between work items and their resolution times. In this quantitative portion of the analysis, we have derived the work item dependency network components. Then, by using work item attributes, measures of network structure drawn from social network analysis, and measures of the communication across dependencies, we created a set of variables that may explain a relationship between resolution times and dependency

structures. Statistical analysis was used to explore this variable set in relation to resolution time.

Based on this quantitative work, the proposition that the structure of dependencies between work items influences the resolution times of those work items cannot be shown to be strictly true. Simple degree centrality measures such as blocking and dependency relationships have decreasing and increasing associations with resolution times in the 5 lower quantiles. This agrees with our observation that developers respond differently to work items when there is a specific and obvious dependency or blocking relationship. Component dependency link density appear to be lower for longer living work items. Component normalized two-step reach showed little variability between groups, though these results were not statistically strong and were limited to a subset of the data with larger dependency networks. None of the measures derived from work items or their dependency network structure can explain a large proportion of the variance in resolution times. In Chapter 6 we relate and integrate these findings with the original proposition.

CHAPTER 6:
DISCUSSION

6.1 Introduction

In this research we conducted a case study using mixed methods to investigate coordination of interdependent work in a software development team. After using stakeholder interviews and qualitative thematic analysis to generate propositions we further explored Proposition #5 quantitatively, using a statistical analysis of work items and explicit dependencies between those work items. In this chapter, we integrate the statistical findings from Chapter 5 with the initial proposition, that the structure of explicit dependencies between work items reduces the resolution times of those work items. Following the integration of the findings, we discuss the interesting aspects of the differences in results from each method and present a refined proposition. This integration is followed by a discussion of the relation of this research to other theory contributions in the field of coordination in software engineering. Finally, we discuss the implications of these finding for software engineering and several threats to the validity of the research performed thus far.

6.2 Integration of Research Results

The two analysis approaches were combined because the qualitative analysis of interviews that generated propositions inspired the quantitative investigation of the work item database. By relating and comparing the results of the quantitative investigation of the behaviour of work item resolution times in relation to explicit dependency structure back to the original proposition, we can broaden our understanding of coordination and specifically Proposition #5. This stage of the

research allows us to understand the proposition from two perspectives and integrate the results. We compare the proposition with the quantitative results and discuss possible reasons for the differences between results.

6.2.1 Review of Proposition and Repository Analysis Results

After generating nine propositions from the qualitative analysis, Proposition #5 was chosen for further study. This proposition suggested that, due to the patterns of dependency creation and the behaviours of stakeholders who manipulate and use those explicit dependencies, there would be a coordination benefit of reduced resolution time. This proposition seemed important because participants noted that patterns of work item dependency expression were used to aid coordination and that this behaviour was linked to a tangible coordination outcome of reduced resolution times. This benefit was thought to be made possible because the expression of dependencies in the work item database would allow the propagation of dependency knowledge where it would otherwise be hidden, raising awareness and assisting in knowledge management.

Specifically, some participants noted that when they see that they are responsible for a work item that is blocking someone else, they will act more quickly and give that work a higher priority. Similarly, some participants noted that they give work items that originate with a different, or remote team, higher priority and try to be more responsive. These stated behaviours and patterns of explicit work item dependency creation seem to make the underlying technical dependencies more visible, easier to locate, easier to mange, and easier to resolve, potentially leading to lower resolution times.

The exploratory statistical analysis of explicit work item dependencies used work item attributes, network analysis network measures of explicit dependency

networks, and measures of communication across dependencies. Statistical approaches were used to assess and quantify the effect of these variables with respect to resolution times. First, using correlation tests between each independent variable and resolution time, it was found that no single feature strongly correlated with resolution times. In no case, can we say that the independent feature can account for a useful proportion of the resolution time. Then, using a difference of medians test, we found that there is a significant difference of median resolution times between work items that have explicit dependencies and those that have no explicit dependencies. Work items with dependencies have a median resolution time of 22 days, which is 15 days longer than those with no dependencies.

Finally, for each of the independent variables, we tested for significantly different medians across 6 quantiles of resolution times, finding that information accumulates over time and that some simple centrality measures are associated with some but not all of the lower resolution times. As work items age the median number of contributors, number of comments, and related work item relationships increase in general as would be expected as information accumulates through a work item's lifetime.

Some lower and higher, but few mid-range resolution time are associated with blocking relationships, indicating that the stated behaviour of responding to work items quickly when they are obviously blocking other work items may be true in some cases. Resolution times beyond 7 days are associated with work items with explicit dependencies on other work items, indicating that expressing dependencies does not, in itself, ensure a fast resolution.

These results are in partial agreement with the stated practices of responding to work items that are blocked by other work items first resulting in shorter resolution

times. This work practice is used to be responsive to remote teams and customers, and is not used in all cases of expressed dependencies. Further, lower resolution times are associated with higher measures dependency link density. This result shows that centrality and reachability through density in the dependency network may be an important factor in the faster resolution of work items. Though the support is not strong, partially due to limited applicability of network measures to dependency networks of size 4 or greater.

Interestingly, there were no statistically significant differences in medians for communication over dependency variables across the six quantiles. Our measurement techniques, based on several variants of counting the number of work items commented on, in a component, by the same participant, and counting the number of contributors who commented on both work items in a dependency relationship, do not seem to have any significant relationship to the resolution time of work items.

6.2.2 Differences Between Findings from Mixed Methods

The results of the quantitative analysis are interesting because they do not immediately corroborate Proposition #5 which suggested that explicit dependencies should be correlated with decreased resolution times. The expected findings were that some aspects of network structure would be found to be associated with consistently decreased resolution times. This relationship was not found with any measures and was supported in part by simple out-degree (the number of blocked work items). Appearing to run counter to the proposition, at a high level, comparing median resolution times between work items with explicit dependencies and those without, those with dependencies have longer resolution times.

It is not immediately obvious why the results are different between interview and repository work items data sources and analysis methods. Upon further reflection, it became clear that the quantitative analysis could only investigate explicit dependencies between work items and failed to account for implicit dependencies. Explicit dependencies are the dependencies that stakeholders create in the work item database representing dependencies that they have identified, and want to record, between pieces of their work. Implicit dependencies are dependencies that concern work items that are involved in dependent work but have no dependency expressed in the work item database. These work items with implicit or unexpressed dependencies are indistinguishable from work items that have no underlying dependency, technical, organizational, or otherwise. The behaviour of explicit dependency structure affecting resolution time, as suggested by participants, may be further supported through analysis of implicit dependencies between work items.

These results highlight the value of a mixed-methods approach that compares the findings from multiple sources of data collection and analysis techniques. By singularly applying either one of the approaches used in this research we would have had much less insight into the role and behaviours of work item dependencies in coordination. By comparing the results from the two stages of this research we are able to gain much greater insight into the influence of explicit dependencies on resolution times of work items.

6.2.3 Refined Proposition

Based on the statistical investigation further exploring Proposition #5 we now have greater insight into the relationship between explicit work item dependency

structure and resolution times. Using this additional information, a refined proposition was constructed as follows:

- Work items with explicit dependencies take significantly longer, both statistically and in practice, to resolve than work items with no explicit dependencies.
- Decreased resolution times are partially associated with centrality and density in explicit dependency structures.
- Overall, the network structure measures of explicit dependencies between work items only account for a small proportion of the variance in resolution times.
- Implicit or unexpressed work item dependencies may have a significant role in coordination and on resolution times. These dependencies can be technical, scheduling, political or other dependencies, and are less manageable as they have not been articulated.

This refined proposition forms a theoretical proposition that can now be tested further in future case studies, or be used to augment existing theory of coordination in software engineering.

6.3 Relation to Existing Coordination Theory Research

While there are many studies in software engineering that empirically test or quantify different aspects of coordination, several recent works in the software engineering community deal specifically with developing theory of coordination. The relation between our findings and the theories presented in those works is discussed in the following sections.

6.3.1 Theory Formulations from Conway and Parnas

First, Herbsleb and Mockus (2003b) developed a preliminary theory of coordination in software engineering based on hypotheses derived from Conway's Law, that the structure of software systems will model the structure of the organization that produced it (Conway 1968), and from Parnas' modularity suggestions, that coupled engineering decisions be clustered into modules (Parnas 1972). They use the construct of *productivity* that is not present in our analysis. Productivity is the number of work items a developer has closed divided by their total time working on the project. They hypothesize and show that developers with more people assigning work to them have lower productivity. This is similar to our overall finding of work items taking longer to resolve when there are dependencies present, but differs in that our analysis shows that some work items that block other work items have lower resolution times.

Similar to our overall findings with dependencies between work items, Herbsleb and Mockus showed that as modification requests cross modules boundaries the cycle time increases. Our participants stated that module boundaries are where explicit dependencies between work items would often be created to raise awareness and to aid coordination of those dependencies. This finding aligns with our overall findings that work items with explicit dependencies do take longer to resolve than those without. Though, our findings suggest that the theory could be refined to account for behaviours, such as fast or early responses to blocking dependencies, that mitigate coordination difficulties associated with dependencies.

6.3.2 Distributed Constraint-satisfaction Problem

Later work by Herbsleb, Mockus, and Roberts (2006) develops a further theory of coordination, mapping coordination to the more generic problem of a distributed

constraint-satisfaction problem. They used the distributed constraint-satisfaction problem to generate hypotheses and then tested them empirically. They use the technical (call and data) dependencies of the files attached to each work item (modification request) as well as other constructs such as size of change, productivity of developers, and errors. The analysis to support this theory shows that higher code level constraint (technical dependency) density is associated with longer resolution times. Our approach to analyzing Proposition #5 differs from this significantly in that we look only at explicit dependencies between work items and relate these to resolution times. Technical dependencies are likely to imply work item dependencies, but explicit work item dependencies may imply technical, process, infrastructure, or other scheduling dependencies. Further we look at specific network structures that might be related to dependency coordination, whereas Herbsleb, Mockus, and Roberts use a construct of *density* that is composite measure of network measurements.

Our results show that for resolution times using explicit work item dependencies, we tend to agree with previous research, except that we see a relationship between several centrality and density measures with lower resolution times for some, but not all work items. The median resolution times are higher when explicit work item dependencies are present as would be suggested by the simplified density measure, but when comparing across groups the degree centrality structure of the dependency network is associated with some lower resolution times.

6.3.3 Socio-technical Congruence

There is an opportunity to investigate whether the mapping between code dependencies and work item dependencies is isomorphic. Proposition #1 in our qualitative interview findings hypothesizes that code dependencies will match work

item dependencies. This is similar to the concept of socio-technical congruence that is emerging from the coordination research literature (Cataldo et al. 2006; Herbsleb 2007). Though socio-technical congruence assesses the match between social communication patterns of participants and technical dependencies. We did not pursue investigation into Proposition #1 at this time due to lack of appropriate technical dependency data.

Trainer et al., (2005) and Cataldo et al. (2006) presented techniques for mapping technical dependencies to social dependencies. Trainer et al. analyzed source code to generate social dependencies and Cataldo et al. mapped work item dependencies to social interactions, generating dependencies by analyzing the files modified in each modification request. Both discuss the concept of socio-technical congruence to characterize the match between technical dependencies and social communications.

Our results based on work item dependencies are similar, but they rely on explicit relationships between work items to act as the dependencies, in combination with our measures of communication across work items, such as a participant commenting on both work items involved in a dependency relationship. Our approach and findings do not characterize a match between technical dependencies and communication, such as that of socio-technical congruence, but attempt to deduce a relationship between the structure of the dependencies and the resolution time. This approach operates under the assumption that communication of some sort must be used to resolve dependencies.

6.3.4 Notes on Analysis Approaches

It should be noted that much of the existing research that has explored coordination and development of empirical coordination theory uses regressions to conduct statistical analysis. Using regression analysis may have enabled deeper

insight into the explicit dependency structures that affect resolution times. We attempted to use linear regression analysis, but could not meet some regression diagnostics, such as normality of residual errors, so we do not include these results in our analysis and comparison. If we are able to complete a valid regression our work with explicit work item dependencies might be more effectively compared to the existing literature.

Using explicit work item dependencies as a basis of analysis appears to be a novel approach to analyzing outcome constructs such as work item resolution time. The research approach was inspired by reports of coordination strategies involving patterns of work item dependency expression from interview participants. This approach of inspiring hypotheses or theoretical propositions is very useful for generating novel directions to broaden empirical theory. While this thesis has not developed a theory of coordination, our mixed methods approach has developed a number of interesting directions for future investigation using the propositions developed from interviews as inspiration. Once investigated, these refined propositions can be used to augment existing empirical theories, such as those presented above.

6.4 Implications for Software Engineering

The findings from this mixed methods research have several implications for software engineering coordination practice, including tools and process, and for future research. Since distributed software development is only becoming more prevalent and since architectural and organization proximity problems are unavoidable, the tools and data that are used for coordination need to be leveraged as much as possible to maximize effectiveness. Interview participants stated that

tools and processes are used to overcome coordination obstacles such as distance, team structure, or even political agendas. It follows that coordination tools and process, even non-traditional software engineering tools, should be used and evaluated. These tools should be available, configurable, and integrated into the development environment in order to exploit the investment in recorded data.

The use of processes and behaviours to assist in coordination, such as specific patterns of work item dependency creation shows that contributors are willing to invest the time and energy into the management of this data. Our statistical analysis of work item dependencies showed some relationship between dependency structure and resolution time outcomes. We may not have chosen the correct outcome of resolution time or entirely appropriate abstractions for independent variables in our analysis, and so may not be able to see a strong relationship, but the fact that contributors are willing to invest time into coordination management activities implies that they are deriving a positive coordination outcome.

The implication of this study for software engineering research is that there is clearly a benefit of increased insight by using a mixed-methods approach in theory building. Through building towards theory of coordination by generating theoretical propositions and refining them using differing data collection and analysis techniques we have gained much more insight into the relationship between explicit work item dependency relationships and resolution times than would have been possible using a singular data collection and analysis method. Using this approach with the remaining theoretical propositions, or using this approach of refining and exploring hypotheses and propositions from existing theories will yield more accurate and useful theory of coordination in software engineering.

6.5 Threats and Limitations

There are several threats to the validity of this research. These are addressed in two sections; the first addresses the interview data collection and qualitative analysis, and the second addresses the work item data collection and statistical analysis.

6.5.1 Threats to Validity of Qualitative Interview Analysis

This analysis and these findings have been carefully and thoughtfully developed, though that does not remove the risk of invalid results. The results of the qualitative portion of this research are discussed in the context of internal credibility and external credibility as presented in the Qualitative Legitimation Model by Onwuegbuzie and Leech (2007). Along with threats to validity, actions taken to retain credibility or to mitigate threats in the research process are described.

Internal credibility refers to the "truth value, applicability, consistency, neutrality, dependability, and/or credibility of interpretations and conclusions within the underlying setting or group" (Onwuegbuzie and Leech 2007, 234). This refers to whether or not the data collection and analysis makes sense and accounts for researcher actions and design within the context of the case under study. The participants views and observations provided during interviews "must be accepted as a reasonable view of what happened" (Miles and Huberman 2004, 277), but there are threats to the validity of the results based on the ways that they are drawn from the data.

In our analysis of interviews and subsequent development of theoretical propositions there are several threats to internal credibility. First, descriptive validity, where the factual accuracy in the documentation of events is threatened, is a possible threat in this research as the interviews were designed, conducted, and

transcribed by the author. To mitigate this threat, the interview script was reviewed by advisors to mitigate potential problems with the application of the interviews, and word-for-word transcription was used to avoid paraphrasing or rewriting of responses. The second potential threat is observational bias, where the data collected and analyzed is insufficient or inappropriate to provide a useful analysis. This threat was partially mitigated by interviewing as many participants as possible, but it should be noted that the participants were self-selecting in that participation was voluntary.

Third, researcher bias is a potential threat that covers the possibility of "the researcher introducing a priori assumptions that he/she is unable to bracket" (Onwuegbuzie and Leech 2007, 236) into the research design, instruments, and analysis. This threat is partially mitigated through prolonged observation of the interview subjects before and after the interviews. This allowed for a contextualized mindset when conducting the analysis of interviews that was intended to quell subconscious biases that may have otherwise been brought to the analysis. Observation also allowed comparison of the themes and propositions being developed with contextual knowledge of the team under study, allowing only themes and propositions that seemed to have meaning to be put forward.

External credibility, within the qualitative portion of the study, concerns whether the results are applicable outside of the case of study; whether there is replicability across cases, and whether there are reactivity and instrument effects on the findings. Several threats to the external credibility of the qualitative portion of this analysis are presented. First, the Hawthorne and novelty effects (Onwuegbuzie and Leech 2007, 236) may have threatened credibility. The fact that the participants are being studied may have influenced their responses and interest. We have no simple

method to account for such an effect in this study. Second, order bias, or the ordering and presentation of questions may have influenced the responses of participants in ways which made them state false or incomplete observations which could lead to incorrect analyses. This threat is partially mitigated by the use of a semi-structured interview script where ad hoc follow-up questioning was used to clarify the intent of answers. Though, strict ordering concerns cannot be mitigated in this research design.

Third, population generalizability is the ability of the research results to be applied to other settings or other cases for the same types of populations. In the process of theory building, this refers to expanding the theoretical propositions to address the population as a whole too early and not acknowledging that the qualitative analysis is valid only within the context of the participants situation. We protect ourselves from this threat by generating propositions describing coordination from the smaller team originally interviewed, then use comparison techniques to refine the theory with work item data from the larger Jazz development team. This analysis has generated theoretical propositions in the context of one case, or one software development team working on one project. This result cannot yet be generalized to all similar projects or to software engineering as a whole. The Jazz project under study does represent a functional and successful software development project and is composed of developers with a varied experience levels and a variety of skills. The likelihood that these results are generalizable to software engineering in a larger context is probable but cannot be demonstrated using this case alone.

6.5.2 Threats to Validity of Quantitative Repository Analysis

Threats to the internal validity lie in the extraction procedure that was used to gather data from the Jazz team repository, and further threats to the construct validity exist in the semantic interpretation of the data during analysis. There are several factors that threaten the validity of the quantitative analysis described in this section.

Threatening the internal validity of the repository analysis, the data extraction procedure used to extract work item data from the Jazz repository did not return the anticipated number of work items. As can be seen in Figure 25, there is a significant decrease in the number of extracted work items in July, August, and September of 2007. Based on a manual inspection of the number of work items available using queries run through the Jazz.net web client, we know that there are at least 6,502 work items that we could not extract from the Jazz team repository between July and December 2007. It is not possible to gauge the effect of these missing work items, so we can only speculate as to whether there is a pattern among the missing data. In our analysis we do not attempt to correct for missing data points.

A second data extraction concern is the quality or reliability of the extracted data. An example of this concern is the work item type field, which we were not able to use in this work. Work item types, such as “task”, “enhancement”, or “defect”, are available in the user interface upon creation of a work item. It seems obvious that we could have filtered or controlled for the type of work items when relating work items to resolution times. Further, it is possible that there are interesting behaviours that occur for enhancements but not for defects, such as work items that are enhancements having more dependencies and taking longer to execute.

Interview participants and observations of the project noted that the work item type field is not reliable as each new work item defaults to “defect” upon creation, and is often not updated to be accurate by the work item creator. For these reasons, we did not use the work item type field in this analysis.

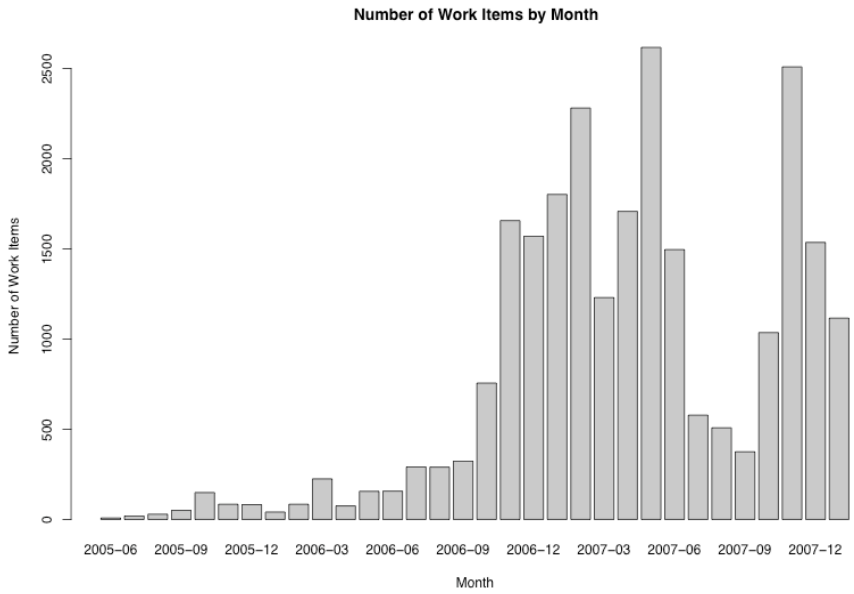

Figure 25: Histogram of work items extracted by month

While calculating the resolution times, it was noted that in some cases the resolution time that we computed, using the difference between resolution date and creation date was negative, implying a negative amount of time to resolve a work item. Further investigation revealed that this was likely due to time zone differences between client and server clocks. It is suspected that in some cases, such as setting the creation time the client’s clock is used, and in other cases, such as the resolution time, the server’s clock is used. This error is obvious and was removed from the data set. The possibility of clock synchronization errors that add time are also possible. The clock synchronization errors we characterized did not account for all of the negative resolution times. Thus, there are likely other timing errors which we cannot detect since they are not obviously erroneous or are not detectable. Since we

do not understand the cause of all the obvious negative timing errors, the extent and severity of this and other potential timing errors is unknown.

There are three threats to construct validity. First, is that we are only considering the explicit work item dependency relations that exist in the Jazz repository. There are potentially many other dependencies between work items that are never expressed or recorded in the repository. This might be because the dependency was easily resolved, or coordination activities took place outside of Jazz and were not recorded. Thus, the structure of the dependencies is partially hidden. This threat is partially mitigated, since the Jazz development team expressed that they try to record as many activities as possible and to use the work item repository as a central communication medium for the project.

Second, only work items that have a resolution time are considered. These are the work items that have been marked as resolved by a process or person in the Jazz development team. There also exist work items that are not resolved, and may never be resolved. These work items were excluded from this analysis as is it difficult to determine if a work item has stalled more than a reasonable amount of time or if there is no intention of ever working on it.

Finally, our choice of using network structure measures is not appropriate in all situations. Since many network measures are not applicable in dependency networks with fewer than 4 work items, parts of our analysis were conducted only with the reduced set of work items that were involved in the larger dependency networks. This reduction makes the results of that portion of the analysis less applicable and less representative of the project as a whole. There may be other abstractions, such as critical dependency paths, which can more effectively model the function of dependency structure in relation to resolution time.

6.5.3 Overall Validity and Limitations

Combining the mixed methods data and analysis, using qualitative approaches to generate preliminary theoretical propositions, and then using quantitative approaches to explore one proposition, has yielded a stronger and more refined theoretical proposition. The findings presented as propositions in Chapter 4 form a description of coordination to be further refined using empirical data. These theoretical propositions are not necessarily applicable beyond the team as the case under study, but form the foundation for future work. The quantitative exploration of Proposition #5 is further exploration in the context of the larger Jazz project and is valid within the Jazz team. Since this is a singular case study we do not have the opportunity to compare these results with other cases, but the Jazz team is a productive and successful industrial software development team, so we suspect that the results of this study will be applicable beyond the scope of this case.

6.6 Summary

This chapter has compared and integrated findings related to Propositions #5 as developed from interview analysis and further explored with statistical analysis of work items. The relationship between explicit dependency structures and resolution times was found to be different than expected based on the proposition developed from interviews. The proposition expected that explicit dependencies would reduce the resolution time of involved work items. In fact, the work items with explicit dependencies take significantly longer to resolve overall, but higher values of some degree centrality measures are related to lower resolution times, and dependency structure overall has little impact on resolution times. A refined proposition is presented demonstrating the integration of the differing research results found using this mixed methods approach, along with implications for

software engineering at large. Finally, a discussion of the threats to the validity of these analyses and limitations of the findings was presented.

CHAPTER 7:
CONCLUSIONS

7.1 Introduction

This research sought to increase our understanding of coordination in software engineering by developing descriptive theoretical propositions that could be used as the basis for broadening or developing theory of coordination in software engineering. Our findings are based on a case study of the practices, views, and experiences of a software development team. Using a mixed methods exploratory approach, a set of theoretical propositions was developed along with an analysis of one of those propositions regarding explicit work item dependency structure and resolution time. In this chapter, we outline avenues for future work to address limitations of this work and to further refine the developed propositions. This is followed by a review the contributions of this work and final words.

7.2 Future Work

This exploratory research used several approaches and methods to explore how developers coordinate interdependent work. From these investigations there are several avenues for further investigation and building of theory of coordination in software engineering. The most significant avenue is to further refine each of the theoretical propositions outlined in Chapter 4, through triangulation, or further exploration. This can be accomplished by exploring each proposition using a data collection and analysis technique that differs from thematic analysis of participant interviews in order to provide alternate views or empirical evidence for comparison. Based on the exploration of Proposition #5 there are opportunities to

conduct further work to address some of the threats to validity, and to develop experimental tooling to support dependency management. The future work in these areas of proposition refinement and dependency networks is presented in the following subsections.

7.2.1 Proposition Refinement

Further work exploring the initial theoretical propositions developed in Chapter 4 will broaden the accuracy and applicability of the propositions. By exploring each proposition from another angle, using empirical evidence, the understanding and detail of the propositions can be increased. The quantitative analysis of Proposition #5, examining work item dependency structure in relation to resolution time in Chapter 5 demonstrates this approach. These refined propositions will then be positioned to be useful for augmenting and expanding existing theory of coordination of interdependent work in software engineering. A short description of a possible avenue for future quantitative exploration of each of the other propositions is outlined below:

Proposition #1: Technical dependencies, such as code dependencies, imply work item dependencies, but work items dependencies do not necessarily imply technical dependencies.

Further Investigation: Compare explicit work item dependency relationships with code dependencies found through static and dynamic code analysis. Assess the match or congruence between the dependency networks.

Proposition #2: Coordination activities are higher for work items with technical dependencies on other work items.

Further Investigation: Measure number of comments and other activity between work items with represent underlying technical dependencies and compare this activity with that of work items that do not represent technical dependencies.

Proposition #3: Contributors working on higher layers in the software stack will have higher coordination needs of their work items than participants working on lower level components.

Further Investigation: Compare the number of relationships, including related, blocks/depends on, and parent/child of work items at different levels in the software architecture. Server level components should have fewer connections overall than user interface level components.

Proposition #4: There is more recorded coordination activity for work items with geographically distributed or cross-team membership.

Further Investigation: Compare coordination activity, such as commenting on, views of, and changes to work items for different levels of cross-team membership and for varying levels of proximity.

Proposition #6: Scheduled and planned work items, such as new features, are resolved more effectively than unplanned work items, such as defect reports.

Further Investigation: Compare resolution times of planned items with resolution times of unplanned items. Explore the differences between planned and unplanned work items to determine why specific work items get planned or prioritized.

Proposition #7: Higher geographic, organizational, and structural distance (low proximity) increases time to resolution due to increased coordination needs.

Further Investigation: Compare work item resolution times, re-opens, associated test failures, and other measures of coordination success with geographic, organizational, and structural proximity measures. Explore the differences across these coordination measures as proximity varies.

Proposition #8: When in times of crisis, coordination tool choices tend towards higher bandwidth, and synchronicity to mitigate distance (low proximity).

Further Investigation: Measure the communication on low geographic proximity work items (those with all participants located at the same site) that are in crisis, such as having critical severity or related to test failures and compare these measures against those from high geographic proximity work items in crisis.

Proposition #9: Expertise recommendations can be made using contributors to previous work items that changed code in the same component.

Further Investigation: Use code authorship and contributors to work items linked to code changes in the component in question to make recommendations for a target set of work items. Survey knowledgeable stakeholders to rate the results.

7.2.2 Future Work on Dependency Networks

There are three interesting avenues for addressing the limitations and exploring the implications of the current investigation into Proposition #5, exploring the

relationship between explicit work item dependency structures and resolution times.

First, there is an opportunity to address dependencies that do exist but were never expressed through work item dependency relationships. This would allow the comparison of resolution times of work items that have explicit dependencies, and work items that have dependencies, but were never expressed through work item relationships. Since, many work items have no underlying dependencies, whether expressed or not, it would be useful to remove these from further analysis. One approach to locate the work items that have dependencies, but no expression of them, is to map work item dependencies to technical dependencies. Technical dependencies could be found through analysis of code or architectural models. Then the match between technical dependencies and expressed work item dependencies can be evaluated. Separating this class of work items with unexpressed dependencies and studying how they are resolved will likely lead to useful insight into the coordination of interdependent work.

Second, since participants noted that they create and use dependency structures to manage interdependency between work, and since there is a small but statistically significant relationship between the degree centrality of a work item in a dependency network and some lower resolution times, then there is an opportunity to create experimental tooling to support awareness, navigation, and creation of dependencies relationships between work items to try to amplify that behaviour and trend for further coordination gains. Prototypes of querying, awareness, and visualization tools that foster the creation of dependency relationships, automatically generate notifications, and specifically exploit dependency relationships should be developed and evaluated.

Finally, there is an opportunity to revisit dependency networks using abstractions, other than social networks, that may more effectively model the behaviour within dependency networks in relation to resolution times. An initial approach that appears interesting, and merits further investigation, is to use concepts, such as critical paths, from operations research. Critical paths, length of critical paths, and fan-in along with fan-out of dependencies, can quantify the level and extent of dependencies around a work item. Approaches similar to Program Evaluation and Review Technique (PERT) and Critical Path Method (CPM) (Winston, 1994) can be used with the addition of estimates or probability distributions of work item completion times either collected through past history or provided by software developers. Using these measures as independent variables in further statistical analysis could yield greater insight into the role of dependencies in relation to resolution time.

7.3 Summary of Contributions

There are three significant contributions from this research that are outlined in this section.

Theoretical Propositions: The first contribution is a set of theoretical propositions describing aspects of coordination of interdependent work in software engineering that were derived from qualitative interview analysis with professional software developers. These propositions form a description of how coordination of interdependent work appears to be managed, and some of the effects of the current management techniques in the case under study. This sets the stage for further proposition refinement by assessing more cases using the same or differing data sources and methods. These propositions can be used to augment existing theory of

coordination or to form the basis of a new theory of coordination in software engineering.

Increased Understanding of Work Dependencies: The second contribution is a deeper understanding and analysis of the relationship between explicit work item dependencies and the resolution times of work items. This focusing and deeper understanding using quantitative statistical methods is an example of how the application of differing data sets and methods from the same case study can yield greater insight into initial theoretical propositions. Deeper understanding of the implications of dependency expression in work item databases allows understanding of how to create better tools and processes for work item management.

Future Work: Finally, this research opens many avenues for future research and demonstrates the value of this mixed methods approach. For each of the propositions developed from the qualitative interview analysis, an approach for future investigation was presented. The investigation of resolution times in the context of dependency network structure was just one of these investigations. By further exploring these propositions using the proposed techniques a much broader and in depth understanding of coordination of interdependent work in software engineering will be developed.

7.4 Final Words

This thesis has explored the coordination of interdependent work in software engineering. A mixed methods approach, anchored in the views and experiences of practicing software developers, was used to develop descriptive theoretical propositions describing coordination of interdependent work and to further refine

one of the propositions. These propositions form the basis for further investigation to develop or broaden theory of coordination in software engineering. The combined results of the mixed methods have broadened our understanding of the management of work item dependencies in software engineering and demonstrated a technique for developing theoretical propositions modeling coordination of interdependent work that are derived from a case study of practicing software engineers.

BIBLIOGRAPHY

Benjamini, Y., and Hochberg, Y. 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society Series B*, 57: 289–300.

Bird, C., A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. 2006. Mining email social networks. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, 137-143.

Bonacich, P. 1987. Power and centrality: A family of measures. *American Journal of Sociology*, 92, 1170-1182.

Butts, C. T. 2007. sna: Tools for social network analysis. R package version 1.5.

Cataldo, M., M. Bass, J. D. Herbsleb, and L. Bass. 2007. On coordination mechanisms in global software development. In *Proceedings of the Second IEEE International Conference on Global Software Engineering,* 71-80.

Cataldo, M., P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. 2006. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work*, 353-362.

Charmaz, K. 2006. Constructing grounded theory: A practical guide through qualitative analysis. Sage Publications.

Cheng, L.-T., S. Hupfer, S. Ross, and J. Patterson. 2003. Jazzing up eclipse with collaborative tools. In *Proceedings of the 2003 OOPSLA Workshop on Eclipse Technology Exchange*, 45-49.

Cheng, L.-T., S. Hupfer, S. Ross, and J. Patterson. 2007. Social software development environments. *Dr. Dobbs Journal*, January 11th, 2007, http://www.ddj.com/dept/cpp/196900164.

Conway, M. E. 1968. How do committees invent. *Datamation* 14, (4): 28-31.

Creswell, J. W. 2003. Research design: Qualitative, quantitative, and mixed method approaches. Sage Publications Inc.

Crowston, K., and J. Howison. 2005. The social structure of free and open source software development. *First Monday* 10, (2) (February).

Damian, D., L. Izquierdo, J. Singer, and I. Kwan. 2007a. Awareness in the wild: Why communication breakdowns occur. In *Proceedings of the 2007 International Conference on Global Software Engineering,* 81-90.

Damian, D., S. Marczak, and I. Kwan. 2007b. Collaboration patterns and the impact of distance on awareness in requirements-centred social networks. In *Proceedings of the 2007 Requirements Engineering Conference*, 59-68.

de Souza, C. R. B., D. Redmiles, D. Millen, and J. Patterson. 2004. How a good software practice thwarts collaboration: The multiple roles of APIs in software development. *ACM SIGSOFT Software Engineering Notes* 29, (6): 221-30.

Easterbrook, S. M., Singer, J., Storey, M, and Damian, D. 2008. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering,* eds. F. Shull, J. Singer and D. Sjøberg, 285-311. Springer.

Eclipsepedia - development process 2006 revision final. http://wiki.eclipse.org/Development_Process_2006_Revision_Final (accessed April 1st, 2008).

Fonseca, S. B., C. R. B. de Souza, and D. F. Redmiles. 2006. Exploring the relationship between dependencies and coordination to support global software development projects. In *Proceedings of the 2006 International Conference on Global Software Engineering,* 16-9.

Freeman, L. C. 1979. Centrality in social networks: Conceptual clarification. *Social Networks* 1, (3): 215-39.

Frost, R. 2007. Jazz and the eclipse way of collaboration. *IEEE Software* 24, (6): 114-7.

Fussell, S. R., R. E. Kraut, J. F. Lerch, W. L. Scherlis, M. M. McNally, and J. J. Cadiz. 1998. Coordination, overload and team performance: Effects of team communication

strategies. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, 275-284.

Gamma, E. and J. Wiegand. 2005. The eclipse way: Processes that adapt. Invited plenary, Eclipsecon 2005. http://eclipsecon.org/2005/presentations/econ2005-eclipse-way.pdf (accessed February 1st, 2008).

Gillham, B. 2000. *Case study research methods.* Continuum International Publishing Group.

Glaser, B. G., and A. L. Strauss. 1967. *The discovery of grounded theory: Strategies for qualitative research.* Aldine Transaction.

Gould, R.V. and Fernandez, R.M. 1989. Structures of mediation: A formal approach to brokerage in transaction networks. *Sociological Methodology*, 19: 89-126.

Halverson, C. A., J. B. Ellis, C. Danis, and W. A. Kellogg. 2006. Designing task visualizations to support the coordination of work in software development. In *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work*, 39-48.

Herbsleb, J. D. 2007. Global software engineering: The future of socio-technical coordination. In *Proceedings of the 2007 Future of Software Engineering*, 188-198.

Herbsleb, J. D., and R. E. Grinter. 1999. Architectures, coordination, and distance: Conway's law and beyond. *IEEE Software* 16, (5): 63-70.

Herbsleb, J. D., and A. Mockus. 2003a. An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering* 29, (6): 481-94.

Herbsleb, J. D., and A. Mockus. 2003b. Formulation and preliminary test of an empirical theory of coordination in software engineering. *SIGSOFT Software Engineering Notes* 28, (5) (September): 138-7.

Herbsleb, J. D., A. Mockus, and J. A. Roberts. 2006. Collaboration in software engineering projects: A theory of coordination. *Proceeding of the 5th IEEE/ACIS International Conference on Computer and Information Science*.

Herbsleb, J. D., A. Mockus, T. A. Finholt, and R. E. Grinter. 2000. Distance, dependencies, and delay in a global collaboration. In *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, 319-328.

Jazz Community Site, http://jazz.net/ (accessed April 1st, 2008).

Johnson, B. R., and A. J. Onwuegbuzie. 2004. Mixed methods research: A research paradigm whose time has come. *Educational Researcher* 33, (7): 14-26.

Katz, L. 1953. A new status index derived from sociometric analysis. *Psychometrika*, 18, 39-43.

Kraut, R., C. Egido, and J. Galegher. 1988. Patterns of contact and communication in scientific research collaboration. In *Proceedings of the 1988 ACM Conference on Computer-Supported Cooperative Work,* 1-12.

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

Kwan, I., D. Damian, and M. A. Storey. 2006. Visualizing a requirements-centred social network to maintain awareness within development teams. *First International Workshop on Requirements Engineering Visualization*, 7-7.

Leech, N., and A. Onwuegbuzie. 2007. A typology of mixed methods research designs. *Quality and Quantity*, published online March 27th, 2007. DOI:10.1007/s11135-007-9105-3 (accessed May 15th, 2008).

Malone, T. W., and K. Crowston. 1990. What is coordination theory and how can it help design cooperative work systems? In *Proceedings of the 1990 ACM Conference on Computer Supported Cooperative Work*, 357-70.

Malone, T. W., and K. Crowston. 1994. The interdisciplinary study of coordination. *ACM Computing Surveys* 26, (1): 87-119.

Maxwell, K. 2002. *Applied statistics for software managers.* Prentice Hall PTR Upper Saddle River, NJ.

Miles, M. B., and A. M. Huberman. 1994. *Qualitative data analysis: An expanded sourcebook.* Sage Publications.

Minto, S., and G. C. Murphy. 2007. Recommending emergent teams. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, 5-5.

Mockus, A., R. T. Fielding, and J. D Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11, (3): 309-46.

Compact Oxford English Dictionary, *s.v. coordination*, http://www.askoxford.com/concise_oed/coordinate (accessed April 1st, 2008).

Nguyen, T., T. Wolf, D. Damian , and A. Schröter. (to appear 2008). Does distance still matter? In *Proceedings of the International Conference on Global Software Engineering*, to appear.

Onwuegbuzie, A. J., and N. L. Leech. 2007. Validity and qualitative research: An oxymoron? *Quality and Quantity* 41, (2): 233-49.

Parnas, D. L. (1972, December). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*. 15 (12), 1053-1058.

Sarma, A., Z. Noroozi, and A. van der Hoek. 2003. Palantir: Raising awareness among configuration management workspaces. In *Proceedings of the 25th International Conference on Software Engineering*, 444-454.

Schümmer, T., and J. M. Haake. 2001. Supporting distributed software development by modes of collaboration. In *Proceedings of the 7th European Conference on Computer Supported Cooperative Work*, 79-98.

Siegel, S. 1956. *Nonparametric statistics for the behavioural sciences.* McGraw-Hill Book Company.

Sim, S. E., S. Easterbrook, and R. C. Holt. 2003. Using benchmarking to advance research: A challenge to software engineering. In *Proceedings of the 25th International Conference on Software Engineering.* 74-83.

Sjøberg, D., T. Dybå, B. Anda, and J. Hannay 2008. Building theories in software engineering. In *Guide to Advanced Empirical Software Engineering,* eds. F. Shull, J. Singer and D. Sjøberg, 312-336. Springer.

Stephenson, K., and M. Zellen. 1989. Rethinking centrality: Methods and applications. *Social Networks* 11, (1): 1-37.

Storey, M.-A. D., D. Čubranić, and D. M. Germán. 2005. On the use of visualization to support awareness of human activities in software development: A survey and a framework. In *Proceedings of the 2005 ACM Symposium on Software Visualization*, 193-202.

Strauss, A. L., and J. M. Corbin. 1998. *Basics of qualitative research: Techniques and procedures for developing grounded theory.* Sage Publications.

Trainer, E., S. Quirk, C. de Souza, and D. Redmiles. 2005. Bridging the gap between technical and social dependencies with Ariadne. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, 26-30.

Van de Ven, A. H., A. L. Delbecq, and R. Koenig. 1976. Determinants of coordination modes within organizations. *American Sociological Review* 41, (2): 322-38.

Wasserman, S., and K. Faust. 1994. *Social network analysis: Methods and applications.* Cambridge: Cambridge University Press.

Winston, W. L.. 1994. *Operations research: applications and algorithms.* Duxbury Press.

Yin, R. K. 2003. *Case study research: Design and methods.* Sage Publications.

Zimmermann, T., and N. Nagappan. 2008. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th International Conference on Software Engineering*, 531-540.

APPENDIX A:
INTERVIEW SCRIPT

Awareness and coordination strategies: Developing a responsive model of artifact relationships and participant communication in a distributed software development environment.

Context

Please tell me a little about yourself and your history with IBM?

Tell me about your roles, responsibilities, and typical activities within the Jazz team(s)?

Workflow & Coordination

During your work, how do you determine what you should be working on?

Are you aware of what your team members are working on?
- How you stay aware of what others are working on?
- Why are you not aware?

Please tell me about a recent scenario where you needed to coordinate your work with another person or team.
- What coordination problems came up during the resolution of this issue?
- How did you solve those problems?

Do you need to coordinate differently with remote members of the team?
- Describe them for me.

When assessing new work how do you determine if it is duplicating, blocking, or related to other work?

Do you find that there are dependencies between your work and work by others in your team or from other teams?
- Describe how you find these dependencies.

How do you manage these dependencies throughout their lifecycle?

Feature Conceptualization

At what level of granularity does it become meaningful to maintain issue awareness?

If you are working on an issue, how do you know that you understand the whole issue?

Describe your communication with others when assessing work and finding related work.

When you need help with some work, how do you find the experts in a certain issue?

APPENDIX B:
RECRUITMENT MATERIAL

Examining team awareness and coordination strategies: Developing a responsive model of artifact relationships and participant communication in a distributed software development environment.

Research Description

Developing software in a distributed development environment has specific challenges and obstacles due to complex technical and social components involved. Changes and dependencies in the software development process and the timeliness of the propagation of changes to concerned stakeholders have a direct impact on development success. Within distributed projects, the problem can be compounded by distance, time zone difference, language, and cultural differences. A lack of awareness of changes of software development artifacts, such as features, and the participants involved can further create problems in the software development process.

I am conducting research to inform the design of future awareness and notification tools used in distributed software development. This research seeks to characterize the existing ways that participants organize features in software engineering repositories and to examine how participants coordinate, communicate, and maintain awareness of dependencies in distributed teams.

Participation

As a member of the Jazz team your participation is requested in the form of a one hour interview with the researcher. This interview will be an open-ended recorded discussion of techniques and practices you use to manage your work. The interviews will be used as a data source to form a descriptive model of awareness and coordination in distributed software development. Interviews can be scheduled any weekday before August 28th, 2007. If you are interested in participating, please contact Lucas Panjer (Lucas_Panjer@ca.ibm.com) for further details.

About the Researcher

Lucas Panjer is a graduate student in the Department of Computer Science at the University of Victoria studying software engineering practices. He is currently working under the supervision of Dr. Daniela Damian in the SEGAL group (http://segal.cs.uvic.ca) and with Dr. Margaret-Anne Storey in the CHISEL group (http://thechiselgroup.org).

APPENDIX C:
PARTICIPANT CONSENT FORM

Department of Computer Science
The University of Victoria

Engineering/Computer Science Building
(ECS), Room 504
PO Box 3055, STN CSC
Victoria, BC
Canada V8W 3P6

Participant Consent Form

Examining team awareness and coordination strategies: Developing a responsive model of artifact relationships and participant communication in a distributed software development environment.

You are being invited to participate in a study entitled "Examining team awareness and coordination strategies: Developing a responsive model of artifact relationships and participant communication in a distributed software development environment." that is being conducted by Lucas Panjer.

Lucas Panjer is a graduate student in the department of Computer Science at the University of Victoria and you may contact him if you have further questions by email at ldp@cs.uvic.ca or phone at 250.472.5781.

As a graduate student, I am required to conduct research as part of the requirements for a Masters degree in Computer Science. It is being conducted under the supervision of Dr. Daniela Damian and Dr. Margaret-Anne Storey. You may contact my supervisor at danielad@cs.uvic.ca (250.472.5717) or at mstorey@cs.uvic.ca (250.472.5713).

This research is partially funded by the IBM Centre for Advanced Studies. The purpose of this research project is to identify (1) collaboration patterns that emerge during a software development project and (2) how project contributors are alerted of changes made in the project repository artifacts.

Research of this type is important because the Software Engineering community is recognizing the importance of the implementation of awareness mechanisms in distributed settings. Alerting contributors timely and providing rationale of features changes enables contributors to maintain awareness of the team workspace.

You are being asked to participate in this study because your role and involvement in the Jazz project. If you agree to voluntarily participate in this research, your participation will include individual interviews, as well as group meetings which will be observed by the researcher. The data collection methods includes interviews and observations, as well the examination of project documents such as requirements, design documents, and communication logs. Lucas Panjer will be conducting the interviews.

Participation in this study may cause some inconvenience to you, including brief distractions from your current activities. In order to avoid this inconvenience the researcher will coordinate with you with anticipation when is the best time to participate in the research. Furthermore, please note that this research does not seek to collect information about individual performance in any way, but to understand how the collaboration process among team members is in practice in software development at IBM. Thus, given your knowledge of the project, your participation is voluntary as much as it is valuable in this research.

The potential benefits of your participation in this research include (1) a better understanding of the collaboration patters in your own project and (2) to understand how you are alerted about changes in artifacts that occur in your project and could potentially affect your own development work.

Your participation in this research must be completely voluntary. If you do decide to participate, you may withdraw at any time without any consequences or any explanation. If you do withdraw from the study your data will be used in the research articles only with your agreement and otherwise will be destroyed. The handwritten notes and audio recordings from interviews will be destroyed.

Apart from identifying information collected by the researcher, your anonymity will be protected. The researcher will be the only person that has access to this information. It is important that the researcher maintains a way to identify each piece of the data collected to enable follow-up with participants in subsequent phases of the research. This identifying information will be protected in the possession of the researcher.

Your confidentiality and the confidentiality of the data will be protected by data being kept electronically on a password protected computer in the SEGAL offices at the University of Victoria; the paper-based information collected during interviews will be kept in a locked cabinet in the researcher’s office at SEGAL at the University of Victoria.

No one other than the researcher will have access to information that identifies you with the research data.

It is anticipated that the results of this study will be shared with others in the following ways presentations at scholarly meetings, published articles at conferences or journals as well as reports made available to IBM. Other planned uses of this data include reports to IBM and research papers submitted to conferences or research journals. None of these reports will provide information to identify the participants or the company name unless permission is granted from IBM management. Data from this study will be disposed after 5 years.

In addition to being able to contact the researchers Lucas Panjer, Dr. Daniela Damian, and Dr. Margaret-Anne Storey at the above phone numbers, you may verify the ethical approval of this study, or raise any concerns you might have, by contacting the Associate Vice-President, Research at the University of Victoria (250.472.4545).

Your signature below indicates that you understand the above conditions of participation in this study and that you have had the opportunity to have your questions answered by the researcher.

_____	_____	_____
Name of Participant	*Signature*	*Date*

A copy of this consent will be left with you, and a copy will be taken by the researcher.