THREE CASE STUDIES ON BUSINESS COLLABORATION AND PROCESS MANAGEMENT

by

Shaokun Fan

Copyright © Shaokun Fan 2012

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF MANAGEMENT

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY
WITH A MAJOR IN MANAGEMENT INFORMATION SYSTEMS
In the Graduate College

THE UNIVERSITY OF ARIZONA

2012
THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Shaokun Fan entitled Three Case Studies On Business Collaboration And Process Management and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

_________________________________________________ Date: 08/01/2012
J. Leon Zhao

_________________________________________________ Date: 08/01/2012
Jay F. Nunamaker, Jr.

_________________________________________________ Date: 08/01/2012
Paulo Goes

_________________________________________________ Date: 08/01/2012
Daniel Zeng

Final approval and acceptance of this dissertation is contingent upon the candidate’s submission of the final copies of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

_________________________________________________ Date: 08/01/2012
Dissertation Director: J. Leon Zhao

_________________________________________________ Date: 08/01/2012
Dissertation Director: Jay F. Nunamaker, Jr.
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the copyright holder.

SIGNED: Shaokun Fan
ACKNOWLEDGEMENTS

I especially thank my advisor, Dr. J. Leon Zhao, for his guidance and encouragement over the past five years. Through his guidance, coaching, and determined insistence on the utmost quality in my work, Dr. Zhao has helped me to develop and advance as a scholar, far exceeding what is expected of an advisor. I believe that what I have learned from him will significantly benefit my future career.

Special thanks to my dissertation co-advisor, Dr. Jay F. Nunamaker, for his valuable suggestions and feedback on my research and continuous support during my doctoral program. Dr. Nunamaker contributed greatly to not only this dissertation, but also to my professional development.

I also would like to give special thanks to my dissertation committee members, Dr. Paulo Goes and Dr. Daniel Zeng, who were always willing to make time for all my questions. They provided me with invaluable advice on how to conceptualize research questions and improve research quality.

I must also thank my minor committee members Dr. Salim Hariri and Dr. Sandiway Fong. I also thank all other faculty members for their support. Additional thanks belong to my colleagues and friends at the University of Arizona, Xin Li, Daning Hu, Manlu Liu, Ping Yan, Runpu Sun, Noyan Ilk, Kunpeng Zhang, and Jiesi Cheng, who made these five years much easier and more colorful.
DEDICATION

This dissertation is dedicated to my father, my mother, my sister, my wife, and my little son for their love, support, and sacrifice over the years.
TABLE OF CONTENTS

LIST OF FIGURES ............................................................................................................ 8
LIST OF TABLES .............................................................................................................. 9
ABSTRACT ...................................................................................................................... 10
CHAPTER 1. INTRODUCTION ..................................................................................... 12
CHAPTER 2. A FRAMEWORK FOR BUSINESS PROCESS MODELING VIA
MODEL TRANSFORMATION ...................................................................................... 20
  2.1 Introduction ........................................................................................................ 20
  2.2 Literature Review ............................................................................................... 23
    2.2.1 Business Process Modeling ..................................................................... 23
    2.2.2 Business Process Model Transformation ................................................ 26
  2.3. Conceptual, Logical, and Physical Models ....................................................... 28
  2.4. An Example of a Conceptual Process Model .................................................... 38
  2.5. A Framework for Business Process Model Transformation ............................. 40
    2.5.1 Transforming Conceptual Model to Logical Model ............................... 40
    2.5.2 Consistency between Conceptual and Logical Models ......................... 54
  2.6 A Case Study and Prototype Implementation .................................................... 60
    2.6.1 A Case Study ........................................................................................... 60
    2.6.2 Prototype System Implementation .......................................................... 63
  2.7 Conclusion .......................................................................................................... 66
    2.7.1 Discussion ............................................................................................... 66
    2.7.2 Limitations and Future Work .................................................................. 68
CHAPTER 3. COLLABORATIVE INTERACTION PROCESS PATTERNS IN OPEN
SOURCE SOFTWARE ISSUE RESOLUTION AND THEIR IMPACT ON TEAM
EFFICIENCY ................................................................................................................... 69
  3.1 Introduction ........................................................................................................ 69
  3.2 Literature Review ............................................................................................... 72
    3.2.1 Dependency and Interaction Patterns in Collaboration ......................... 72
    3.2.2 Team Efficiency in Software Development ............................................ 74
    3.2.3 Pattern Mining from Sequences .............................................................. 75
  3.3 Dataset ................................................................................................................ 78
  3.4 Interaction Patterns ............................................................................................. 80
  3.5 Interaction Pattern Mining .................................................................................. 85
  3.6 Pattern Impact Assessment ................................................................................. 97
  3.7 Results ............................................................................................................... 102
    3.7.1 Quantitative Analysis ............................................................................ 102
    3.7.2 Qualitative Analysis .............................................................................. 110
  3.8 Conclusion ........................................................................................................ 114
    3.8.1 Implications ........................................................................................... 114
    3.8.2 Limitations and Future Work ................................................................. 117
CHAPTER 4. A THEORY OF COLLABORATION VIRTUALIZATION ..................... 120
TABLE OF CONTENTS

4.1 Introduction ...................................................................................................... 120
4.2 Literature Review ............................................................................................. 121
4.3 Collaboration Virtualization Theory ................................................................. 123
   4.3.1 Definitions and Overall Conceptual Model .............................................. 123
   4.3.2 Task Characteristics .............................................................................. 126
   4.3.3 Technology Characteristics ................................................................... 129
   4.3.4 Team Characteristics ............................................................................. 130
   4.3.5 Moderating Effect of Multi-task Degree ............................................... 132
4.4 Applying Collaboration Virtualization Theory to an Empirical Setting .......... 133
4.6 Discussions and Conclusions ........................................................................... 137
   4.6.1 Contributions ......................................................................................... 137
   4.6.2 Limitations ............................................................................................ 138
   4.6.3 Future Work .......................................................................................... 139

CHAPTER 5. CONCLUSIONS AND FUTURE WORK .............................................. 141
  5.1 Contributions .................................................................................................... 141
  5.2 Future Directions .............................................................................................. 143
REFERENCES ............................................................................................................... 146
LIST OF FIGURES

Figure 1. The Workflow Management Lifecycle ............................................................. 30
Figure 2. Diagrammatic Representation of DND ............................................................. 38
Figure 3. DND Model of the Insurance Claim Process .................................................... 40
Figure 4. The Model Transformation Procedure .............................................................. 41
Figure 5. Pattern-based Model Fragments ........................................................................ 51
Figure 6. Workflow Model Validation Procedure ............................................................ 54
Figure 7. Ontology Classes of Workflow Models ............................................................ 56
Figure 8. Ontology Instances of the Petri Net Model ....................................................... 57
Figure 9. Ontology Instances of the DND Model ............................................................. 57
Figure 10. Process Fragments and the Corresponding Derived Petri Net Model ............. 63
Figure 11. Architecture of Prototype System ................................................................. 64
Figure 12. Logical Model Generator Interface ............................................................... 65
Figure 13. Model Validator Interface ............................................................................... 65
Figure 14. An Interaction Process in Issue Resolution in Jira .......................................... 79
Figure 15. Interaction Process Instances and Their Patterns ............................................ 85
Figure 16. The Interaction Pattern Mining Procedure ..................................................... 87
Figure 17. An Example of Graph Inclusion ..................................................................... 91
Figure 18. An Example of Interaction Pattern Mining Procedure .................................... 97
Figure 19. The Effect of a Pattern on the “Hazard” .......................................................... 99
Figure 20. The Distribution of Patterns ......................................................................... 104
Figure 21. Extracted Patterns in Software Issue Resolution ........................................... 106
Figure 22. Conceptual Model of CVT ............................................................................ 126
Figure 23. A Typical Software Development Process .................................................... 134
LIST OF TABLES

Table 1. Process Model Classification Criteria ................................................................. 36
Table 2. SWRL Constraint Rules ...................................................................................... 60
Table 3. Functional Units and the Derived Logical Tasks ................................................ 61
Table 4. Conceptual Dependencies and the Derived Control Dependencies ................. 61
Table 5. Control Dependencies among Tasks Within the Same Role .............................. 61
Table 6. Conceptual Dependencies and the Derived Data Dependencies ....................... 62
Table 7. Data Dependencies among Tasks within the Same Role .................................... 62
Table 8. Comparison of Different Pattern Mining Approaches ....................................... 77
Table 9. Summary Statistics ........................................................................................... 103
Table 10. Regression Results of Software Issue Resolution Processes .......................... 109
Table 11. Case Summary ................................................................................................ 111
Table 12. Differences between Virtual and Physical Collaboration ............................... 124
Table 13. Comparison between Requirement Analysis and Coding and Testing........... 135
ABSTRACT

The importance of collaboration has been recognized for more than 2000 years. While recent improvement in technology creates vast opportunities for collaboration, effective collaboration remains challenging as ad hoc teams work across time, geographical, language, and technical boundaries, and suffer from process inefficiency. My dissertation addresses part of these challenges by proposing theoretical frameworks for business collaboration and process management. Case study is used as a research strategy for this thesis and it consists of three studies.

The first study proposes a process modeling framework to support efficient process model design via model transformation and validation. First, we divide process modeling into three layers and formally define three layers of workflow models. Then, we develop a procedure for transforming a conceptual process model into its corresponding logical process model. Third, we create a validation procedure that can validate whether the derived logical model is consistent with its original conceptual model.

The second study proposes a framework for analyzing the relationship between interaction processes and collaboration efficiency in software issue resolution in open source community. We first develop an algorithm to identify frequent interaction process structures referred to as interaction process patterns. Then, we assess patterns’ impact through a time-dependent Cox regression model. By applying the interaction process analysis framework to software issue resolution processes, we identify several patterns
that are significantly correlated with collaboration efficiency. We further conduct a case study to validate the findings of pattern efficiency in software issue resolution.

The third study addresses the issue of suitability of virtual collaboration. Virtual collaboration seems to work well for some cases, but not for others. We define collaboration virtualization as the suitability for a task to be conducted virtually and propose a Collaboration Virtualization Theory (CVT) to explain collaboration virtualization. Three categories (i.e., task, technology, and team) of constructs that determine the suitability of collaboration virtualization are derived from a systematic literature review of related areas.

In summary, this dissertation addresses challenges in collaboration and process management, and we believe that our research will have important theoretical and practical impacts on the development of collaboration management systems.
"For each individual among the many has a share of virtue and prudence, and when they meet together, they become in a manner one man, who has many feet, and hands, and sense.” The great Greek philosopher Aristotle described the importance of collaboration among people more than 2000 years ago. The recent improvement in information technology has provided companies with collaboration tools like micro blogs, and wikis, and has as such created a “mass collaboration” (Gobillot, 2011; Tapscott and Williams, 2008). According to a recent survey, 73% of knowledge workers reported working with people in different locations within their own company and 67% of knowledge workers reported working with people in other companies at least monthly (Forrester, 2009). Effective collaboration is no longer a competitive advantage for companies; now it is imperative for business success (Handoll et al., 2012). However, effective collaboration remains challenging as ad hoc teams work across time, geographical, language, and technical boundaries (Forrester, 2009). My dissertation addresses several challenges faced by effective collaboration and proposes theoretical models on business collaboration and process management.

Collaboration is broadly defined as a “process through which parties who see different aspects of a problem can constructively explore their differences and search for solutions that go beyond their own limited visions of what is possible” (Gray, 1989), p.5. During the collaboration process, people share resources and expertise, and join their efforts to deliver outputs beyond what individuals can achieve (Nunamaker et al., 2001).
The process of collaboration includes various types of interactions among people, such as communication, cooperation and coordination (Gerosa et al., 2006). As a result, collaboration processes are dynamic and complicated in nature and collaboration process management is critical to group productivity and performance (Thomson and Perry, 2006).

Recent years have witnessed a trend of collaboration virtualization: geographically or temporally dispersed people collaborate with each other via collaboration technologies or other virtual means (Overby, 2008). Group-based collaboration and technologies to support a broad range of interaction have proliferated and are increasingly a central component in organizations (Smith and McKeen, 2011). A review of the literature concludes that “with rare exceptions all organizational teams are virtual to some extent” (Martins et al., 2004). Thus, information technologies are playing a more and more important role in teams nowadays.

Two areas are closely related to the research in this dissertation: team collaboration (Wood and Gray, 1991) and process management (Weske, 2010). Research on team collaboration focuses on finding important factors for team performance. Researchers have examined the impact of personalities of team members, the characteristics of team structure, and communication modes on team productivity (Drexler et al., 1988). In the IS field, the use of information technologies in team collaboration has been intensively studied and researchers have proposed theories on technology choice and adoption for teams (Briggs, 2006). These theories have shed
important insights on collaboration management and led to the development of various collaboration systems, such as group decision support systems, social network systems, collaborative learning environment, etc.

Research on process management is mainly concerned with process modeling, analysis and automation in various business contexts (Stohr and Zhao, 2001). For the purpose of business process management, processes are normally modeled with modeling languages such as Petri nets, UML activity diagrams, BPMN and so on (van eer Alast W. M, 1998). In order to formally verify the logical and functional correctness of business process models, many model verification approaches that can be applied to specific process modeling languages have been developed (Bi and Zhao, 2004; van der Aalst and ter Hofstede, 2000). Once a process is formally modeled, it can be automated via workflow management systems.

Although previous research results in two areas mentioned above are fruitful, the collaboration process is normally treated as a “black box” in previous studies (Thomson and Perry, 2006). The deep and dynamic nature of interaction process structure has not been appropriately addressed. Even with the support of many types of collaboration systems, collaboration process management is still a challenging problem because most existing collaboration systems cannot fully satisfy the requirements of managing the dynamic and complicated processes that may occur among team members (Baker et al., 1999). So far, collaboration management has faced several challenges.
The first challenge is about how to model processes correctly and efficiently. Automating collaboration processes requires formally defined process models. Further, many verification mechanisms can be used to check the logical correctness of formal process models so that logical errors can be identified before process execution. Since collaboration processes include many types of interactions among people, building a correct collaboration process model is a knowledge-intensive effort that requires detailed understanding of all process aspects (Stohr and Zhao, 2001). When people collaborate in a virtual environment with the support of information technologies, the situation becomes even more complicated. Compared with traditional workflow models, collaboration process models tend to be recursive, dynamic, ad hoc and uncertain (Sheth and Kochut, 1998). Traditional workflow modeling approaches do not explicitly consider collaboration requirements in business processes. Few mechanisms allow users to explicitly derive process models from business requirements. Moreover, there is no systematic method to support model reuse, and model designers have to redesign the whole process model when the context is changed.

The second challenge is how to analyze collaboration processes and their efficiency. The collaboration process that a team adopts has an important impact on the outcome of the team. Current collaboration process management approaches mainly rely on ad hoc and “best practice” type of experiences that are synthesized by experts who have gone through the process many times (Briggs et al., 2003). While this approach provides very useful guidelines to collaboration management, it is hard to apply
experiences from one domain to another. People have to spend a huge amount of effort on “trial and fail” to gain experience for their collaboration contexts.

The third challenge is how to assess the suitability of using IT in collaboration management. With widespread use of collaboration technology and increasing dispersion of teams due to globalization of companies, IT is playing an important role in collaboration process management (Overby, 2008). However, information technologies-based collaboration works well for some cases, but not for others (Zigurs and Buckland, 1998). While information technologies facilitate communication among people, they also introduce problems such as distrust, information overload and so on (Dewett and Jones, 2001). The decision on when to use virtual collaboration is critical for team success.

Facing these challenges, I aim to achieve the following objectives in this dissertation:

- Develop methodologies and techniques to facilitate collaboration process modeling and automation.
- Propose methodologies for analyzing collaborative interaction processes and their relationship with team efficiency.
- Propose theoretical models to guide management of virtual collaboration processes.

Each of the next three chapters will address one or multiple objectives above. The case study research strategy will be used in all three studies in this dissertation. Here, case study is a research strategy that investigates a business collaboration problem within
its real-life context (Yin). Case study research can mean single and multiple case studies, include qualitative and quantitative evidence, and benefit from the prior theoretical propositions (Tietje and Schol, 2002).

In Chapter 2, we propose a three-layer process modeling approach to support process automation via workflow systems. Conceptual models are normally used to document the generic business process requirements in the company. Logical models are generally used for defining technology specific requirements, where software modules as well as their behavioral patterns should be clearly specified. Physical models are only used for system execution. However, the transformation from conceptual models to logical models can be a tedious task, often causing errors in the resulting logical model. In this chapter, we propose a formal approach that can be used to support efficient and accurate model transformation. First, we develop a procedure for transforming a conceptual process model into its corresponding logical process model. Business requirement analysis, dependency mapping, and workflow pattern-based model transformation are the major components of this transformation procedure. Second, we create a validation procedure that can validate whether the derived logical model is consistent with its original conceptual model. Business process ontologies are employed in our approach to describe both conceptual and logical models. We also implement a prototype system and conduct a demonstrative case study to show the feasibility of our approach.
In Chapter 3, we propose a framework for analyzing the relationship between interaction processes and collaboration efficiency. Previous research on collaboration posits the collaboration process as a key factor for team performance. However, it is not fully understood which characteristics of a process make collaboration more efficient. In this chapter, we investigate the effect of interaction patterns on teamwork efficiency (e.g., time cost) in the software development setting. We propose a framework to identify frequent process structures referred to as interaction patterns and study their impact on the efficiency of software development. For purposes of pattern mining, we propose an algorithm to extract process graphs from software development processes stored in a software project tracking system. To analyze the effect of different interaction patterns, we conduct an empirical study to examine their correlation with issue resolution time using data from an open source software community. As a result, we identified several interaction patterns that are positively (or negatively) correlated with issue resolution time. We further conduct a case study to validate the findings of pattern efficiency and provide a rich and contextualized explanation of pattern efficiency in software issue resolution. Our research helps identify interaction patterns that are preferable or detrimental for process efficiency, thus providing insights for designing tools and mechanisms for collaboration management.

In Chapter 4, we address the issue of suitability of virtual collaboration. The move towards collaboration virtualization is faster than ever before. However, some tasks are more successful with virtual collaboration while others are more successful with physical
collaboration. This phenomenon motivates these research questions: What factors determine the suitability of collaboration virtualization, and how do those factors affect the design of effective collaboration systems? Our literature study yielded little theoretical work in this regard. As such, we believe that research on collaboration virtualization theory (CVT) is critically needed. To this end, we present our preliminary findings on the purpose and composition of collaboration virtualization theory based on the literature. Essentially, our CVT contains three categories of constructs: task, team, and technology characteristics. Our main objective in this chapter is to initiate a new theoretical perspective for research in the field of collaboration technology and management. We do a case study on collaboration tasks in software development to illustrate how CVT can be used to guide strategic decision making on collaboration virtualization.

Finally, we conclude this dissertation, stating its contributions to the collaboration process management domain, and present directions for further research in Chapter 5.
CHAPTER 2. A FRAMEWORK FOR BUSINESS PROCESS MODELING
VIA MODEL TRANSFORMATION

2.1 Introduction

Because the focus of process management is often collaboration between departments or between companies, process management offers great support for collaboration management (Weske, 2010). Process modeling is the first step for process management. After a process is modeled, we can analyze, improve, and automate the process via different techniques. There are various process modeling languages to describe processes (Stohr and Zhao, 2001). While much attention has been paid to the logical correctness of these models (Bi and Zhao, 2004; van der Aalst and ter Hofstede, 2000), developing a workflow application that can fulfill given requirements is also very important (Jørgensen et al., 2008; van der Aalst et al., 2005). The terms “business process model” and “workflow model” are both found in the literature. The business process model is often used when communicating with managers, while the workflow model is commonly used at the system level. In this chapter, we will use both terms interchangeably.

This research is motivated by the need to resolve a real-world problem in the context of the Kuali project (Liu et al., 2007). Kuali is a community source project to develop a comprehensive suite of administrative software that meets the needs of all Carnegie Class institutions. There are currently more than twenty development partners in the Kuali project. In this context, we need to develop a workflow model to support
software change management based on a conceptual process model. Further, we need to validate that the workflow model we develop is consistent with the given conceptual process model. However, we could not find an existing approach for systematically transforming a conceptual business process model to a physical process model.

Over the past twenty years, a lot of work has been done in the area of business process modeling. Much research has been devoted to model expressiveness (Russell et al., 2005; van der Aalst et al., 2003), and some research has focused on business process model verification (Bi and Zhao, 2004; Stohr and Zhao, 2001; van der Aalst and ter Hofstede, 2000; Wang and Zhao, 2011). However, these approaches stop at logical correctness. Only a few approaches (Jørgensen et al., 2008; Mans et al., 2009; van der Aalst et al., 2005) in the literature explicitly capture business requirements in the workflow design process even though doing so was suggested more than ten years ago (Georgakopoulos and Tsalgatidou, 1998). Further, for formal verification, some workflow models are very difficult for managers to understand, which often results in a gap between managerial users and technical developers of workflow applications. For example, in order to add a new task to a Petri net-based workflow model, one must manipulate the model in terms of transitions, places, arcs, and tokens, which can be done correctly and efficiently only by someone well-versed in Petri nets, a skill not normally possessed by ordinary managers.

Designing a workflow model is a knowledge-intensive endeavor because creating a typical workflow model requires detailed understanding of various process components,
such as business process logic, the organizational chart, and the information systems accessed by the workflow. The whole design process may require collaboration between an enterprise’s functional and technical departments. More importantly, the model is subject to frequent modification due to changes in the process components. As has been done in the database field, dividing the design process into three phases (i.e., conceptual, logical, and physical design) should enhance the efficiency of modeling as well as the quality of the design output. A conceptual, logical, or physical business process model is the output of each design phase respectively. The conceptual business process model has a higher level of abstraction then the other two types of models. The transformation from conceptual business process model to logical business process model and then to physical business process model is very important in terms of mapping business requirements to system implementation. The terms “conceptual model,” “logical model,” and “physical model” are used to represent the three models in the rest of this chapter.

In this chapter, we present a detailed transformation procedure from conceptual to logical models. We choose Dependency Network Diagrams (DND) (Tillquist et al., 2002) as the conceptual model because of its simplicity and expressiveness. Further, Petri nets is chosen as the sample logical modeling language because of the availability of abundant verification techniques (van der Aalst, 1998). Here, the conceptual model is mainly used to capture the business requirements in enterprises; the logical model is used for information system (e.g., workflow) design purposes; and the physical model is only used for system execution. In particular, the key challenges in this research are how to derive a
logical model from a given conceptual model and how to validate that the derived logical model is consistent with the given conceptual model. To provide a flexible modeling framework, model designers can derive logical models iteratively based on conceptual models in the logical design process. That is, logical design of a workflow model can involve multiple logical models.

The main contributions of this chapter are threefold. First, we propose a three-layer modeling approach that differentiates among conceptual, logical, and physical models. Second, we develop a methodology for transforming a conceptual model into a logical model. Third, we create an approach for validating whether the derived logical model is consistent with its corresponding conceptual model. The rest of this chapter is organized as follows. In Section 2.2, we review the related areas of research. In Section 2.3, we define conceptual, logical, and physical models in detail, and we address relationships among the three types of models. Section 2.4 gives an example conceptual model in DND. Section 2.5 presents the transformation procedure for deriving a logical model from a conceptual model and introduces the validation procedure. Section 2.6 validates our approach by a case study and prototype system development. The conclusion in Section 2.7 includes discussion and limitations of our research.

2.2 Literature Review

2.2.1 Business Process Modeling

Business process modeling has been a subject of study from both managerial and technical perspectives. From the managerial perspective, business process modeling is
about the understanding and analysis of business processes. Over the past twenty years, business process modeling became an important aspect of Business Process Redesign (BPR) for business management in order to improve business efficiency (Aguilar-Savén, 2004; Phalp and Shepperd, 2000). The focus of business process modeling in BPR is on “Why” a particular process activity is undertaken (Bradley et al., 1995). For instance, the previous study identified seven principles that should guide any business process re-engineering exercises undertaken (Hammer, 1990). The unique contribution of BPR over past organizational change approaches is its primary focus on the business process. Typically, the result of business process modeling is a model at the conceptual level since no consideration is given to what technology to use in the implementation of the given business processes. Popular modeling languages from this area include GED (Katzenstein and Lerch, 2000), i* model (Yu, 1997), and DNDs (Tillquist et al., 2002).

From the technical perspective, business process models provide a blueprint for the development of information systems, leading to model-driven system development (Atkinson and Kuhne, 2003). A business process model is also referred to as a workflow model, although a workflow model typically requires detailed information in five perspectives, namely, functional, behavioral, informational, operational, and organizational (Stohr and Zhao, 2001). This is because a workflow model needs to be deployed and executed in workflow management systems (WFMS) while a business process model might not. Consequently, a workflow model requires specific information related to the workflow technology (logical level) or even specific workflow software.
(physical level). Important modeling languages from this area include Petri nets, UML activity diagrams, and BPMN.

The two perspectives mentioned above previously have been separated, and little work has been done to explore the relationship between them (Dreiling et al., 2008). Existing process modeling languages that feature different degrees of abstraction for different user groups exist and are used for different purposes in business process management (Dreiling et al., 2008). In a case study on process modeling, three levels of process models are identified: the abstract level, the organizational level, and the operational level (Glassey, 2008). Similarly, Dreiling et al. (2008) distinguished three perspectives in process modeling: management, business process analyst, and technical analyst. Our work is similar to these studies because we emphasize the difference between different types of models. However, our approach also tries to formalize three levels of workflow models and facilitate model transformation.

A couple of methods have been proposed recently to develop business process models based on particular business requirement documents. In (Jørgensen et al., 2008; Mans et al., 2009; van der Aalst et al., 2005), Colored Petri nets (CPN) were used as a requirement model to specify, validate, and elicit user requirements. Then the requirement in CPN is transformed to a workflow model and to an implementation of the new system. These approaches hold similar objectives, but they use the same model language (i.e., CPN) to describe both user requirements and workflow models. The drawback of this method is that managers do not understand CPN. Our approach chooses
conceptual models in the business process analysis domain as the starting point and helps business process modelers design business models to meet the business requirements.

In this chapter, DNDs and Petri nets are chosen as examples of conceptual and logical modeling languages, respectively. DNDs (Tillquist et al., 2002) were recently proposed as a new representation methodology, which allows the essential elements governing organizational relations to be captured, communicated, and evaluated under changing conditions. By depicting important features of organizational relations, information systems can be designed explicitly for the control and coordination of organizational activities. Petri nets, as a state-based graphical modeling language, have become one of the most popular workflow modeling languages (van der Aalst, 1998). Many analysis techniques are available for Petri nets. Thus, DND and Petri nets are chosen as the example modeling languages in this chapter.

2.2.2 Business Process Model Transformation

The transformation between models of different levels of abstraction such as platform-independent models and platform-specific models is a critical step of system development in model-driven architecture (Atkinson and Kuhne, 2003). While model transformation techniques have attracted lots of attention (Czarnecki and Helsen, 2006), defining a transformation between any two workflow modeling languages is still a difficult task as several domain-specific problems remain to be solved. In (Murzek and Kramler, 2007), seven issues about defining business process model transformations are
identified based on the observations of four business process modeling languages.

Some approaches have been proposed for transforming one workflow modeling language to another. In order to perform formal analysis on BPEL, both BPEL2PN (Hinz et al., 2005) and WofBPEL (Ouyang et al., 2005) provide the functionality of transforming BPEL to Petri nets. BPMN, as a popular workflow modeling language, can be translated to Petri nets through certain mapping rules (Dijkman et al., 2007). BPMN can also be translated to BPEL for system implementation purposes (Ouyang et al., 2006). Other transformation approaches for workflow modeling languages can also be found in the literature. However, these approaches are mostly done in an ad hoc way.

In addition to the language-specific approaches mentioned above, there are a number of other approaches to developing a general framework for business process model transformation. Lohmann et al. (2008) proposed a strategy, relying on Triple Graph Grammars (TGG), for implementing a model transformation based on workflow patterns. In their approach, TGGs allow structural relationships between different model elements to be elegantly expressed in graphical, declarative rules. Murzek and Kramler (2007) also proposed a general approach to business process model transformation based on workflow patterns. Using the example of translating EPC to BPMN, Vanderhaeghen et al. (2005) presented an XML-based approach for model transformation of business process models. However, without being aware of the difference between models of different abstraction, these approaches tend to treat transformations between any two modeling languages in the same way.
A common weakness of most existing approaches, which is a core feature of this work, is that the differences between levels of abstraction in the existing models are ignored. Those methods usually assume that there is equivalent semantic information in the source model and target model. However, as is pointed out by researchers (Recker and Mendling, 2006), different workflow models contain different semantic information. These differences can lead to semantic mismatches when transforming one model to another. In this chapter, we first clearly define three levels of abstraction, namely, conceptual model, logical model, and physical model. Then the transformations between conceptual and logical model are studied and a general framework is proposed based on workflow pattern analysis.

2.3. Conceptual, Logical, and Physical Models

In this section, we explore the concepts of conceptual, logical, and physical models. As in the database field, dividing workflow modeling tasks into three stages has the following advantages. First, conceptual modeling tasks can be performed before decisions are made on the selection of workflow technology and software. Second, workflow analysts can specialize in different types of modeling tasks, some of which require more knowledge about business while others require knowing more about technology. This is particularly true for large organizations with complex business processes that may take months to analyze and model. Third, a workflow model resulting from the conceptual modeling stage can be reused multiple times in the logical modeling
stage if changes to technology occur later on. Similarly, the same logical model might be used for different workflow software and for specific software versions. Fourth, changes to business requirements can be easily mapped into the system implementation through the transformation from conceptual model to logical model, and then to physical model.

The current business process literature tends to lump all these models into a single “workflow model” or “business process model.” However, there are many types of workflow models, each of which may play a different role in the business process management lifecycle (Dreiling et al., 2008). Conceptual models are mainly used for managers and business analysts to analyze business processes, while physical models are mainly used to implement business processes. Here, logical models are the bridge between conceptual and physical models, where logical formalization and verification can be realized. When the three models are linked together, we can have a better understanding between the managerial users and technical developers of business processes. Figure 1 illustrates the relationships among the three different types of models.
Conceptual, logical, and physical models are derived from theoretical work on three perspectives of process modeling (Dreiling et al., 2008) and three levels of process modeling (Glassey, 2008) in previous studies. We extend the previous work by defining generic definitions of different types of processes models that are not restricted to a particular process modeling language. By formally defining the three layers of models, we divide the complex process modeling problem into three tasks so that workflow modeling can be done successively from simple to complex in three steps in terms of information contents. This helps simplify the process of system analysis and design by means of the so-called “divide and conquer” approach. At the conceptual level, we do not consider logical and physical constraints. By applying the theory of information hiding from software engineering to model building, we can simplify the modeling tasks by focusing on the most important concepts without worrying about unnecessary details.
In addition, changes to the workflow models can be started in any of the three layers first and then propagate the changes to other layers through model transformation and validation procedures.

Since the existing process modeling procedure tends to lump all three layers of models into one model, most of the process modeling languages cannot be perfectly mapped to our definition of models in each layer. Therefore, our layered approach requires future work on development of process modeling languages in each layer. Further, whether all of the three layers of models are required in process modeling may depend on the complexity of the business environment. For simple and stable business processes, one or two layers modeling might be more effective. Further studies have to examine the benefits and costs of layered modeling approaches and provide practical guidelines for effective process modeling.

Conceptual models are created for at least four purposes: (1) providing a way for developers and users to communicate, (2) increasing analysts’ understanding of the context, (3) serving as the basis of design, and (4) serving as documentation of the requirements (Kung and Solvberg, 1986). Based on the literature in business process modeling at the conceptual level (Katzenstein and Lerch, 2000; Tillquist et al., 2002; Yu, 1997), we identify four key constructs as the scope of conceptual modeling:

1. **Goal.** A goal is a desirable or suitable objective.
2. **Functional unit (or conceptual activity).** A functional unit is the means or procedure to provide material or informational resources necessary to achieve a goal.
3. Role. A role is a bundle of actions, obligations, perspectives, and other concerns that characterize an individual or group of individuals in organizations.

4. Dependency. A dependency is comprised of logistic, financial, informational, or managerial (e.g., evaluation) relationships that one role establishes with another in the process to achieve their goals.

These four elements are found as essential concepts in most of the modeling languages that are used for process analysis and design. A conceptual model usually reflects knowledge about the application domain rather than about the implementation of information systems (Wand et al., 1995). It presents exactly what the process is expected to do, but includes no technology-dependent specifications. Therefore, a true conceptual model designed by business analysts should be independent of any implementation techniques or platforms.

**Definition 2.1: (Conceptual Process Model).** A conceptual process model is a derived representation of a real-world business process without concern for specific workflow technology. It can be represented as a tuple \((\text{ROLE}, \text{FU}, \text{GOAL}, \text{DEPENDENCY})\), where

1. **ROLE** is a non-empty finite set of roles;
2. **GOAL** is a non-empty finite set of goals;
3. **FU** is a non-empty finite set of functional units;
4. **DEPENDENCY** \(\subseteq (\text{FU} \times \text{FU})\), is a binary relation.

A logical model is developed based on the business requirements found in the
conceptual model to describe the business process logic needed to fulfill the conceptual process model with a particular workflow technology. In our context, a workflow technology can be message-based or event-based (Swenson and Irwin, 1995). The correctness and completeness of the logical model should also be verified before moving on to the development of the physical model.

When designing a logical model, detailed information about tasks should be presented. Specification of workflow routing conditions should also be given in the logical model. By examining current workflow model languages (e.g., Petri nets, UML AD, BPMN and so on) that are designed for system design and implementation, we identify four key constructs as the scope of the logical model:

1. Task. A task is an individual implementable module of any workflow system.
2. Data. Each data is input or output of tasks.
3. Control flow. Control flow is the execution order and constraints among tasks.
4. Dataflow. Dataflow describes the flow of business data (e.g., file and document) among tasks.

Logical tasks are usually derived from conceptual models through functional decomposition. In the derivation process, control flow and dataflow should conform to the given conceptual dependencies. Compared with conceptual models, logical models provide detailed information needed to support system design, and consequently the execution logic should be clearly specified.

**Definition 2.2: (Logical Process Model).** A *logical process model* is a
representation of system design, which is independent of WFMSs. It can be formalized as a tuple \((\text{TASK}, \text{DATA}, \text{CF}, \text{DF})\), where

1. \(\text{TASK}\) is a non-empty finite set of logical tasks;
2. \(\text{DATA}\) is a non-empty finite set of data objects;
3. \(\text{CF} \subseteq (\text{TASK} \times \text{TASK})\), is a binary relation, which describes dependency relationships between tasks;
4. \(\text{DF} \subseteq (\text{DATA} \times \text{TASK} \times \text{TASK})\), is a trinary relation.

The logical design of a workflow model is very complicated because it requires much more information than what is embedded in the conceptual model. It usually takes several rounds of interaction to achieve the logical model that is ready for physical implementation. To provide a flexible modeling framework, we need to consider the iterative nature of model analysis and design. Depending on the granularity of available information, logical models for a workflow instance can vary a lot. Therefore, our definition of logical model allows flexibility in logical design by enabling model designers to derive logical models iteratively. Model designers can first derive a preliminary logical model on the basis of a loosely or partially specified modeling requirement, and the full specification of the model is achieved through several rounds of interaction. For each step of logical model design, more information is added to the previously designed logical model and finer granularity can be achieved.

Once a specific workflow system is chosen, a physical model can be derived from the logical model. The physical model is obtained by converting the logical model to a
language that can be directly used as input by the chosen WFMS. Physical models are usually machine-level languages (e.g., XML), which are easily interpreted by a WFMS. A complete physical model should include all the workflow artifacts required to build the software application, such as data format, constraint definitions, protocols used for communication between different tasks, and security constraints. We identify two concepts as the scope of physical process models:

1. Procedure. A procedure is a section of programs that perform a specific task;
2. Message. A message is a piece of information that is passed from one procedure to another.

**Definition 2.3: (Physical Process Model).** *A physical process model is a representation of software design that takes into account the facilities and constraints of a given workflow management system. It can be formalized as a tuple (PROCEDURE, MESSAGE), where*

1. PROCEDURE is a non-empty finite set of procedures;
2. MESSAGE is a non-empty finite set of messages.

In summary, the three types of workflow models mainly differ in the following three aspects: (1) they have different purposes; (2) they are used by different users; and (3) they describe a business process at different levels of abstraction. The definitions above can be used to judge whether a modeling technique is considered as a conceptual model or a logical model. Six criteria are proposed in this chapter to classify existing modeling languages.
1. **Purpose.** Why is the modeling language used?

2. **User.** Who should use the modeling language?

3. **Scope.** What are the constructs (or entities) addressed by the modeling language?

4. **Formality.** Can the modeling language be easily interpreted by computers?

5. **Independence.** Is the modeling language independent of system implementation?

6. **Understandability.** Can the modeling language be easily understood by a professional?

<table>
<thead>
<tr>
<th>Table 1. Process Model Classification Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Purpose</strong></td>
</tr>
<tr>
<td><strong>User</strong></td>
</tr>
<tr>
<td><strong>Scope</strong></td>
</tr>
<tr>
<td><strong>Formality</strong></td>
</tr>
<tr>
<td><strong>Independence</strong></td>
</tr>
<tr>
<td><strong>Understandability</strong></td>
</tr>
</tbody>
</table>

These six criteria can be used to characterize workflow modeling languages as summarized in Table 1. Currently, there is a significant divergence in workflow modeling paradigms. Different models have been developed with various objectives, and each modeling paradigm has its own strengths and limitations. Based on the six criteria, we can measure the fitness scores of existing modeling languages and classify them into
different categories. The detailed method for calculating the fitness score is part of our future work. Nevertheless, we can conduct high-level analysis of the classification by referring to (Recker and Mendling, 2006). In their approach, business process modeling languages are classified based on their ability to describe four constructs.

Recker and Mendling’s work included 12 modeling languages, most of which were invented for workflow automation instead of business process analysis. Some models such as i*, GED, and DND were invented for business process analysis and were not included in their work (Recker and Mendling, 2006). However, in this chapter, we consider them as good candidates of conceptual modeling languages because they are easy to understand. Other models such as Petri nets (van der Aalst, 1998), PI-Calculus (Puhlmann and Weske, 2005), and event-based workflow model (Kumar and Zhao, 1999) are mainly used in the software modeling domain and they are more suitable for logical modeling. Script languages, such as BEPL and XPDL, are mainly used for system execution and they are more suitable for physical modeling.

The transformation from logical models to physical models will not be addressed in this chapter. We defer this topic to future research. In fact, some existing WFMSs can provide the transformation from logical models to physical models. For example, jBPM (jBPM) allows users to model workflow via a graphical logical model and transform the graphical model to JPDL, a workflow execution script which can be considered as a physical model. Nevertheless, our research in this chapter sheds new light on model transformation by developing a formal framework and some basic concepts.
2.4. An Example of a Conceptual Process Model

In this section, we briefly introduce a conceptual modeling language—DND, which will be used to demonstrate our approach in the follow sections. A vehicle insurance example represented by DND is also presented in this section. The DND, as a model of management action and IT design, is a good example of a conceptual model because of its expressiveness and parsimoniousness. It has a graphical representation with five basic elements: activity, role, goal, dependency, and governance control. More importantly, DNDs are very useful for business process analysis (Tillquist, 2005), the primary focus of conceptual modeling.

DNDs (Tillquist et al., 2002) diagrammatically depict the exchange channels, governance controls, and roles among different participators in business processes. The DND is essentially a model of management action and IT design. Activity, goal, role, and dependency can be mapped to the four concepts we defined in Definition 2.1. Beside these four concepts, governance control is also used in DND to describe a prescription for acceptable actions to fulfill a dependency. The five concepts are represented graphically in Figure 2.
A typical insurance claim process can be modeled with DND. A detailed construction algorithm and rules for DNDs can be found in (Tillquist et al., 2002). Figure 3 depicts the relationships between a claimant, an insurance company, and a vehicle repair shop in the process. This example is a simpler version of the process mentioned in (Tillquist et al., 2002). The claimant files a claim with the insurance company and submits the relevant vehicle to a repair shop to restore the vehicle function. The shop repairs vehicles for customers and gets payment from the insurance company, with the intention of generating a profit. The insurance company processes claims, work orders, and payments, with the intention of resolving claims. Four key dependencies are included in the vehicle repair process.

1. The vehicle repair shop depends on the insurance company for claim payment.
2. The claimant depends on the vehicle repair shop for vehicle repair.
3. The shop depends on the insurance company to authorize the repair.
4. The claimant depends on the insurance company to resolve the claim.

According to Definition 2.1, we can formally represent the DND model in Figure 3 as follows:

\[
\text{ROLE} = \{\text{Claimant, Insurance Company, Vehicle Repair Shop}\}
\]

\[
\text{GOAL} = \{\text{ResolveClaim, RestoreVehicle, GenerateRevenue}\}
\]

\[
\text{FU} = \{C1, I1, I2, V1, V2\}
\]

\[
\text{DEPENDENCY} = \{(C1, I1), (V2, I2), (I1, V1), (C1, V1)\}
\]

Next, we choose Petri net as an example logical model and link DND with Petri
net in two respects: deriving a Petri net model from a given DND and checking whether the derived Petri net conforms to the given DND model.

Figure 3. DND Model of the Insurance Claim Process

2.5. A Framework for Business Process Model Transformation

2.5.1 Transforming Conceptual Model to Logical Model

In this section, we present a transformation procedure that takes a conceptual model as input and generates a logical model. If the logical design requires several iterations of logical modeling, the conceptual model and previously achieved logical modeling elements can be used as input and the transformation procedure can also be applied. Note that this transformation is not restricted to DND or Petri nets. It can be applied to any conceptual or logical modeling languages as long as they conform to our definitions of conceptual and logical models. Some steps in the procedure require human
intervention while other steps can be entirely automated. As shown in Figure 4, the model transformation procedure contains eight steps, which are defined next.

**Figure 4. The Model Transformation Procedure**

**Step 1: Derive logical tasks.** We derive a logical task set \( \text{LTS} = \{\text{LT}_1, \text{LT}_2, \ldots, \text{LT}_n\} \) based on a given set of conceptual functional units \( \text{FUS} = \{\text{FU}_1, \text{FU}_2, \ldots, \text{FU}_n\} \).

The conceptual model usually describes the tasks at a higher level of abstraction than logical and physical models do because it does not need to consider any system implementation details. Activities in conceptual models are typically functional units. It is hard to directly implement each of these functional units with just one task. The first step of transformation is to analyze the conceptual activities and develop them into appropriate logical tasks. Sometimes, observations and interviews in the field are needed to gather more information. This step generally relies on human intelligence and domain knowledge. Derived logical tasks should maintain some properties, such as coupling and
cohesion. A complex functional unit can be decomposed into multiple logical tasks. A simple functional unit that has only some simple functions can be modeled with a single logical task.

Step 2: Derive control dependencies. We derive a set of control dependencies CDS = \{CD_1, CD_2, \ldots, CD_n\} in this step. In this chapter, we use the notation \((SourceTask, TargetTask)\) to represent control dependency. A control dependency \((SourceTask, TargetTask)\) means that SourceTask should be executed before TargetTask. In this step, two types of control dependencies need to be considered: within the same role and among different roles.

Control dependencies among logical tasks that are assigned to different roles can be produced by mapping conceptual dependencies to the logical level. Conceptual dependencies are found among functional units, while logical dependencies are found among logical tasks. The dependency mapping rule is represented in Algorithm 2.1.

<table>
<thead>
<tr>
<th>Algorithm 2.1: Mapping Control Dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A set of conceptual dependencies and a set of functional units FUS={FU_1, FU_2, \ldots, FU_n}, with their corresponding logical task set LTS_1, LTS_2, \ldots, LTS_n</td>
</tr>
<tr>
<td><strong>Output:</strong> A set of derived control dependencies, CDS</td>
</tr>
<tr>
<td>CDS=∅</td>
</tr>
<tr>
<td>For every element FU_i in FU</td>
</tr>
<tr>
<td>For every other element FU_j in FU (i\neq j)</td>
</tr>
<tr>
<td>If there exists conceptual dependency (FU_i , FU_j)</td>
</tr>
<tr>
<td>Add LTS_i \times LTS_j to CDS</td>
</tr>
<tr>
<td>Return CDS</td>
</tr>
</tbody>
</table>

Control dependencies among logical tasks that are assigned to the same role are typically not specified in the conceptual model. More information from users is required.
to define these dependencies. Similar to task decomposition in Step 1, generating dependencies among tasks corresponding to the same functional unit also relies greatly on human intelligence.

**Definition 2.4: (Redundant Control Dependency).** If a group of three control dependencies is in the form of \((T_1, T_2) (T_1, T_3) (T_2, T_3)\), the control dependency \((T_1, T_3)\) is redundant.

Redundant control dependencies need to be removed when generating logical models. For example, based on dependencies \((T_1, T_2)\) and \((T_2, T_3)\), we can get a sequential model with tasks \(T_1, T_2,\) and \(T_3\). Adding an extra dependency \((T_1, T_3)\) can only give the same model as before. Therefore, redundant control dependencies are removed in this step in order to reduce the complexity of the subsequent transformation steps.

<table>
<thead>
<tr>
<th>Algorithm 2.2: Removing Redundant Control Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> A set of control dependencies (CD_{S} = {CD_{1}, CD_{2}, \ldots, CD_{n}})</td>
</tr>
<tr>
<td><strong>Output:</strong> A set of control dependencies (CD_{S} = {CD_{1}, CD_{2}, \ldots, CD_{n}}) without redundant control dependencies</td>
</tr>
<tr>
<td>For every element (CD_{i}) in (CD_{S})</td>
</tr>
<tr>
<td>For every element (CD_{j}) in (CD_{S}) ((i \neq j))</td>
</tr>
<tr>
<td>For every element (CD_{k}) in (CD_{S}) ((k \neq i, k \neq j))</td>
</tr>
<tr>
<td>Check redundancy among (CD_{i}, CD_{j}, CD_{k})</td>
</tr>
<tr>
<td>If (CD_{i}) is redundant, remove (CD_{i}) and goto 1</td>
</tr>
<tr>
<td>If (CD_{j}) is redundant, remove (CD_{j}) and goto 2</td>
</tr>
<tr>
<td>If (CD_{k}) is redundant, remove (CD_{k})</td>
</tr>
<tr>
<td>Return (CD_{S})</td>
</tr>
</tbody>
</table>

At the end of this step, we derive a complete set of control dependencies with on redundancy.

**Step 3:** Derive data dependencies. We derive a set of data dependencies \(DDS = \)
Here, we use the notation \( (\text{DataObject}, \text{SourceTask}, \text{TargetTask}) \) to represent data dependencies. A data dependency \( (\text{DataObject}, \text{SourceTask}, \text{TargetTask}) \) means that \text{DataObject} is the output of \text{SourceTask} and input of \text{TargetTask}.

In this step, data dependencies among logical tasks are identified. As is done in Step 2, data dependencies among tasks derived from different functional units can be generated by mapping conceptual dependencies to the logical level. However, data dependencies among tasks derived from the same functional unit can only be defined based on further requirement analysis.

**Algorithm 2.3: Data Dependency Mapping**

**Input:** A set of conceptual dependencies and a set of functional units \( \text{FUS} = \{\text{FU}_1, \text{FU}_2, \ldots, \text{FU}_n\} \), with their corresponding logical task set \( \text{LTS}_1, \text{LTS}_2, \ldots, \text{LTS}_n \)

**Output:** A set \( \text{DDS} \) of derived data dependencies

\[ \text{DDS} = \emptyset \]

For every element \( \text{FU}_i \) in \( \text{FU} \)

For every other element \( \text{FU}_j \) in \( \text{FU} \) (\( i \neq j \))

If there exists conceptual dependency \( (\text{FU}_i, \text{FU}_j) \)

For all data objects \( \text{D}_i \) in the dependency

Identify the task \( \text{T}_x \) using \( \text{D}_i \) as input

Identify the task \( \text{T}_y \) using \( \text{D}_i \) as output

Add \( (\text{D}_i, \text{T}_x, \text{T}_y) \) to \( \text{DD} \)

Return \( \text{DDS} \)

**Step 4: Business rule enforcement.** We derive a set of rules \( \text{RULES} = \{\text{RULE}_1, \text{RULE}_2, \ldots, \text{RULE}_n\} \). We use the notation \( \text{RULE} = (\text{CD}, \text{Condition}) \) to represent rules, where \( \text{CD} \) is a control dependency and \( \text{Condition} \) is comprised of predicates combined by AND and OR operators. A rule \( (\text{CD}, \text{Condition}) \) means that the control dependency \( \text{CD} \) holds only when \( \text{Condition} \) is true.

In this step, we introduce business rules that are extracted from various
documentation sources to constraint the model transformation process so that the derived logical model can better satisfy business requirements. A business rule is a statement that defines or constrains some aspect of the business (van der Aalst et al., 2011). It allows managers or business analysts to specify policies in small, stand-alone units using explicit statements. It is usually extracted from business policies and rule documents or through user interviews (Wang et al., 2009). Since a conceptual model does not provide all the necessary information for building the corresponding logical model, logical model designers have to dig more detailed business information through user interviews.

Because data are normally used as parameters of routing constraints (or conditions), we represent a rule as the mapping relationship between a control dependency and a predicate that asserts the values of data objects. Rules are very important for workflow routing when implementing business processes. The procedure to generate rules is to examine every control dependency and see whether it is “unconditional” or “conditional.” Here, a “conditional” control dependency means that the target task may or may not be executed after the source task is finished. An “unconditional” control dependency means that the target task must be executed after the source task is finished.

Step 5: Pattern recognition. Formally, we retrieve a set of patterns $PATTERNS = \{\text{PATTERN}_1, \text{PATTERN}_2, \ldots, \text{PATTERN}_n\}$. We use the notation $\text{PatternName(Task}_1,\text{Task}_2,\ldots,\text{Task}_m)$ to represent a workflow pattern with $m$ tasks. Based
on the information provided by previous steps, patterns involved in the workflow model are identified in this step. Based on (van der Aalst et al., 2003), five basic workflow patterns are discussed here.

**Lemma 2.1.** (SEQUENTIAL pattern). Given tasks A and B, SEQUENTIAL(A,B) holds if and only if the following conditions are satisfied: (1) (A,B) ∈ CDS; (2) (B,A) ∉ CDS; (3) ∀X, X ≠ A → (X,B) ∉ CDS; (4) ∀Y, Y ≠ B → (A,Y) ∉ CDS.

**Discussion.** If there is one and only one control dependency (A, B) that uses task A as the source task (condition (1), (3), and (4)), then task B must be executed when task A is finished. Condition (2) guarantees that A will not be executed after B is finished. Therefore, tasks A and B consist of a SEQUENTIAL pattern.

**Lemma 2.2.** (AND-SPLIT pattern). Given tasks A, B, and C, AND-SPLIT(A,B,C) holds if and only if the following conditions are satisfied: (1) (A,B) ∈ CDS ∧ (A,C) ∈ CDS; (2) (B,A) ∉ CDS ∧ (C,A) ∉ CDS ∧ (B,C) ∉ CDS ∧ (C,B) ∉ CDS; (3) ∀X (((A, B), X) ∉ RULES ∧ ((A, C), X) ∉ RULES.

**Discussion.** If two control dependencies use task A as the source task (condition (1)), and there is no rule associated with any of these control dependencies (condition (3)), then tasks B and C must be executed when task A is finished. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B, and C consist of an AND-SPLIT pattern.

**Lemma 2.3.** (XOR-SPLIT pattern). Given tasks A, B, and C, XOR-SPLIT(A,B,C) holds if and only if the following conditions are satisfied: (1) (A,B) ∈ CDS ∧ (A,C) ∈ CDS; (2) (B,A) ∉ CDS ∧ (C,A) ∉ CDS ∧ (B,C) ∉ CDS ∧ (C,B) ∉ CDS; (3) ∀X (((A, B), X) ∉ RULES ∧ ((A, C), X) ∉ RULES.
CDS; (2) (B,A)∉CDS ∧ (C,A)∉CDS ∧ (B,C)∉CDS ∧ (C,B)∉CDS; (3) ∃X ((A, B), X)∈RULES ∧ ∃Y ((A, B), Y)∈RULES.

Discussion. If two control dependencies use task A as the source task (condition (1)), and there is a rule associated with each of these control dependencies (condition (3)), then tasks B and C can be executed if and only if task A is finished and the related rule is true. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B, and C consist of an XOR-SPLIT pattern.

Lemma 2.4. (AND-JOIN pattern). Given tasks A, B, and C, AND-JOIN(A,B,C) holds if and only if the following conditions are satisfied: (1) (A,C)∈CDS ∧ (B,C)∈CDS; (2) (C,A)∉CDS ∧ (C,B)∉CDS ∧ (A,B)∉CDS ∧ (B,A)∉CDS; (3) Output(A) ∩ Output(B) = Ø, where Output(T) = {Data object i | ∀ T_x (i, T, T_x) ∈ DDS}.

Discussion. If two control dependencies use task C as the target task (condition (1)), and the output data of all source tasks A and B have no intersection, then both tasks A and B should be executed before executing task C. Otherwise, there will be missing data since tasks A and B generate different data. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B, and C consist of an AND-JOIN pattern.

Lemma 2.5. (XOR-JOIN pattern). Given tasks A, B, and C, XOR-JOIN(A,B,C) holds if and only if the following conditions are satisfied: (1) (A,C)∈CDS ∧ (B,C)∈CDS; (2) (C,A)∉CDS ∧ (C,B)∉CDS ∧ (A,B)∉CDS ∧ (B,A)∉CDS; (3) Output(A) ∩ Output(B) ≠ Ø, where Output(T) = {Data object i | ∀ T_x (i, T, T_x) ∈ DDS}. 
Discussion. If two control dependencies use task C as the target task (condition (1)), and the output data of both source tasks A and B have at least one common data item, then only one of these tasks should be executed. Otherwise, there will be duplicated data in the process. Condition (2) excludes the situation that a loop exists among these tasks. Therefore, tasks A, B, and C consist of an XOR-JOIN pattern.

Theorem 2.1: (Pattern confliction free). If both tasks A and B are in a pattern and 
\((A,B)\in\text{CDS}\), then tasks A and B cannot both be included in another pattern.

Proof. Let Lemma 2.2, 2.3, 2.4 and 2.5 be the only patterns that can be recognized based on given control dependencies, logical dependencies, and conditions. We use enumeration to prove that every pattern is exclusive.

(1) Based on Lemma 2.1, if SEQUENTIAL\((A,B)\) holds, then only one control dependency uses task A as the source task. However, if tasks A and B are included in AND-SPLIT, AND-JOIN, XOR-SLIP or XOR-JOIN patterns, then there must be either more than one control dependency (condition (1) in Lemma 2.2, 2.3, 2.4 and 2.5) or no control dependency including tasks A and B (condition (2) in Lemma 2.2, 2.3, 2.4 and 2.5). If there is no control dependency between A and B, it contradicts condition (1) in Lemma 2.1. If there is more than one control dependency between A and B, it contradicts conditions (3) or (4) in Lemma 2.1. In conclusion, if tasks A and B are included in a SEQUENTIAL pattern, they cannot be included in other patterns at the same time.

(2) Based on Lemma 2.2, if AND-SPLIT\((A,B,C)\) holds, then two or more control dependencies use task A as the source task, and there is no condition assigned to any of
these control dependencies. SEQUENTIAL(A,B) and SEQUENTIAL(A,C) cannot hold because they contradict condition (1) in Lemma 2.2. SEQUENTIAL(C,B), SEQUENTIAL(B,C), SEQUENTIAL(C,A), and SEQUENTIAL(B,A) cannot hold because they contradict condition (2) in Lemma 2.2. Therefore, any two tasks from an AND-SPLIT pattern cannot be included in a SEQUENTIAL pattern at the same time. Similarly, by enumerating all other possible patterns, we can conclude that any two tasks from AND-SPLIT(A,B,C) cannot be included in other patterns at the same time.

(3) Similarly, we can prove that when two tasks are included in XOR-SPLIT, AND-JOIN, or XOR-JOIN, they cannot be included in other patterns at the same time. Detailed proof is omitted due to space limitation.

Theorem 2.1 is very important because it guarantees that any two logical tasks can have at most only one kind of relationship in terms of workflow model patterns. This opens up the possibility of performing a deterministic transformation from patterns to a workflow model. If two tasks are included in two different patterns, then there is a conflict between these two patterns. As a result, it is hard to determine which pattern should be used when the logical model is generated. For example, if tasks A, B, and C are recognized as an AND-SPLIT pattern and tasks A, B, and D are recognized as a XOR-SPLIT pattern, then the relationship between tasks A and B is nondeterministic based on patterns.

Lemma 2.1, 2.2, 2.3, 2.4, and 2.5 provide us the mathematical foundations for automating the pattern recognition process. Based on our discussion above, the pattern
The sequential pattern involves only two tasks. The algorithm chooses any two tasks from the logical task set and matches them with conditions in Lemma 2.1. The other four patterns involve three tasks. So the algorithm chooses any three tasks and matches the dependencies and conditions with their definitions. Based on Theorem 2.1, we can get a set of patterns without any conflicts.

Only five basic workflow patterns are discussed because of space limitations. The recognition rules for other advanced workflow patterns can be defined accordingly if needed. Nevertheless, these five patterns represent what the commercial workflow
engines can directly support for the most part (van der Aalst et al., 2003). In addition, those advanced patterns (e.g., multi-choice) that can be decomposed into several basic patterns can also be addressed by our approach.

If not all logical tasks are included in at least one pattern, it means that the control dependency, data dependency, or rules are not correctly analyzed. In this case, we need to go back to step 1 and redo the dependency analysis for the tasks that are separated from any pattern.

![Pattern-based Model Fragments](image)

**Figure 5. Pattern-based Model Fragments**

**Step 6: Pattern-based model transformation.** We generate a workflow model fragment for each pattern. In this step, patterns identified are translated into workflow model pieces according to the pattern-model transformation rules, which are available for most current logical modeling languages. This step relates to the specific logical
modeling language that is chosen in the application. For example, we used Petri nets for our sample logical model. Five basic patterns of Petri net model are shown in Figure 5. Note that this step should not be a barrier to applying our approach to other logical modeling languages. This step can be easily migrated to other modeling languages because we can directly plug in the structures of workflow patterns in those languages and transform patterns into workflow model pieces.

**Step 7: Model assembling.** In this step, we assemble workflow model fragments into an integrated workflow model. According to Theorem 2.1, if the patterns of the logical model are supported by the pattern recognition procedure, all tasks should be included in the set of model pieces. Because there is no confliction among these model pieces, it is straightforward to assemble them into a logical model. Without loss of generality, we present this algorithm at the conceptual level. The model assembling procedure selects any two model pieces that have common node(s) and connects separate pieces into an integrated model. All separate pieces should be integrated into a single model without any confliction.

**Algorithm 2.5: Model Assembling**

**Input:** A set of model pieces  
**Output:** An integrated model  
1. Put all model pieces into a queue Q  
2. Select two items in the queue that have the same task node(s)  
3. Assemble the two pieces into one by linking them through the shared nodes  
4. Put the newly generated pieces into Q  
5. Go to step 2 until the models in the queue cannot be assembled any more

**Step 8: Refine.** This is the last step of transformation where integrity of the logical
model is addressed. For Petri net-based logical models, tokens need to be added to the start state. The derived model needs to be checked against specific rules for different modeling languages.

The transformation procedure described in this section takes the high-level conceptual model as input and generates the detailed logical model. Mapping conceptual-level business requirements to the logical model helps workflow model designers design models that are functionally correct. Compared with building a logical mode from scratch, our approach allows users to design a logical model just by analyzing control and dataflow between any two tasks that are derived from different functional units. This can greatly reduce the complexity of workflow model design. Further, our approach allows designers to build logical models iteratively based on a given conceptual model. For each round of logical model design, the 8-step approach can be applied iteratively until the finest granularity is achieved.

Although we use DNDs and Petri nets as two example modeling languages in this section to illustrate our approach, the transformation method proposed above is independent from those modeling languages. It can be generated with any other conceptual and logical modeling languages as long as those models conform to our definitions of a conceptual model and a logical model. For example, if we want to perform transformation from an i* model to a BPMN model, we can follow the steps proposed in this section and perform the transformation. The only thing depending on a particular modeling language is the mapping from patterns to a specific language. Since
lots of work has been done on the patterns of different modeling languages, this issue is not addressed in this chapter.

2.5.2 Consistency between Conceptual and Logical Models

Since the whole transformation procedure in Section 2.5 involves much human effort, such as in Step 1 to Step 4, mistakes are possible during the transformation. For example, in Step 1, people may specify the subtask relationship incorrectly because they do not have enough specific knowledge about the business domain. Or, in Step 2, people may miss important control dependencies or add unnecessary control dependencies by mistake. Such mistakes are also possible in Steps 3 and 4. These errors sometimes are unavoidable and may cause severe problems in workflow execution. In this section, we will demonstrate how to check these errors by validating the consistency between logical and conceptual models in order to guarantee that the logical model does not violate any constraints defined in the conceptual model.

![Figure 6. Workflow Model Validation Procedure](image)
We use the Web Ontology Language (OWL) to build ontologies for conceptual and logical models, respectively, and use the properties of OWL classes to describe their mapping relationships. Further, to ensure the two models are consistent, some consistency rules are checked by using the KAON2 inference engine (KAON2). Reasoning in KAON2 is implemented by novel algorithms which reduce a SHIQ(D) knowledge base to a disjunctive data log program, which makes it very efficient. An overview of this validation process is depicted in Figure 6.

The ontologies describe (the) three main parts—the logical model, the conceptual model, and the mapping relations between the two models. The ontology designed and used as input for checking consistency depends on the modeling languages chosen for the logical and conceptual modeling. Our design of a Petri net ontology is based on previous work (Gašević and Devedžić, 2006). It contains the classes for places, transitions, and arcs. Similarly, the DND ontology is defined based on core model elements in the DND model. Relationships among model elements are also captured by the ontology. Figure 7 illustrates a sample ontology describing logical and conceptual models. Properties are symbolized by arrows, which correlate the OWL classes to one another. For example, the property dependOn is used to describe dependencies in Petri nets.

**Definition 2.5: (Model Element Correspondence).** There is a correspondence between a logical model element $L$ and a conceptual model element $C$ if $L$ is derived from $C$. Formally, $\text{Correspondence}(L, C)$. 
The correspondence relationship can be identified in the transformation process proposed in Section 2.5.1. The correspondence relationships for all logical model elements derived from the conceptual model are identified and used as input for the consistency check. For example, if logical tasks $LT_1$ and $LT_2$ are derived from the conceptual task $C_1$ in Step 1 of the transformation process, the correspondence relationships $(LT_1, C_1)$ and $(LT_2, C_1)$ should be identified.

For certain business processes, we can build the workflow model ontology by generating ontology instances based on the ontology classes in Figure 7. The ontology classes specify only a set of constraints that all business process models have to satisfy. With detailed ontology instances for certain business processes, domain-specific constraints can be defined and validated by the inference engine. Figure 8 and Figure 9 show two fragments of the OWL file for the insurance claim process example, where ontology instances of the Petri net model and DND model are defined, respectively.
Figure 8. Ontology Instances of the Petri Net Model

```xml
<!-- http://process.arizona.edu/ontologies/Process.owl#C-1-2 -->
<Transition rdf:about="#C-1-2">
    <fromTrans rdf:resource="#A3"/>
    <toTrans rdf:resource="#A4"/>
    <output rdf:resource="#Insurance info"/>
    <input rdf:resource="Claim method ">
</Transition>

<!-- http://process.arizona.edu/ontologies/Process.owl#C-1-3 -->
<Transition rdf:about="#C-1-3">
    <fromTrans rdf:resource="#A5"/>
    <toTrans rdf:resource="#A6"/>
    <dependON rdf:resource="#I-1-1"/>
    <output rdf:resource="#Insurance Info"/>
    <input rdf:resource="Claim method ">
</Transition>

<!-- http://process.arizona.edu/ontologies/Process.owl#P2 -->
<Place rdf:about="#P2">
    <fromPlace rdf:resource="#A3"/>
    <fromPlace rdf:resource="#A5"/>
    <toPlace rdf:resource="#A2"/>
</Place>
```

Figure 9. Ontology Instances of the DND Model

```xml
<!-- http://process.arizona.edu/ontologies/Process.owl#C-4 -->
<DNDFU rdf:about="#C-1">
    <Correspondence rdf:resource="#C-1-1"/>
    <Correspondence rdf:resource="#C-1-2"/>
    <Correspondence rdf:resource="#C-1-3"/>
    <source rdf:resource="#D1"/>
    <source rdf:resource="#D3"/>
    <hasGoal rdf:resource="#C-G1"/>
</DNDFU>

<!-- http://process.arizona.edu/ontologies/Process.owl#C-1 -->
<DNDGoal rdf:about="#C-G1">
    <hasRole rdf:resource="#Claimant"/>
</DNDGoal>

<!-- http://process.arizona.edu/ontologies/Process.owl#C-1 -->
<DNDDependency rdf:about="#D1">
    <dataObject rdf:resource="#Insurance info"/>
</DNDDependency>
```
Once we have the ontology for given conceptual and logical models, we can conduct validation with the help of a rule reasoning engine. Rule languages allow a significant extension of the machine-processable semantics. Here, Semantic Web Rule Language (SWRL) is used to describe rules (W3C, 2004). It was proposed based on a combination of the OWL DL and OWL Lite sublanguages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sublanguages of the Rule Markup Language. Rules in SWRL are in the form of an implication between an antecedent (body) and consequent (head). The intended meaning can be read as: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold. Atoms in these rules can be of the form C(x), P(x,y), sameAs(x,y), or differentFrom(x,y), where C is an OWL description, P is an OWL property, and x,y are either variables, OWL individuals, or OWL data values. Note that the consistency rule should be restricted to the so-called DL-safe subset (Motik et al., 2005) of the SWRL, in order to make reasoning decidable. Applied to business process modeling, such rules can be used to enrich the constraints that a specific business process model should fulfill. If some mistakes happen during the transformation, the consistency will no longer be maintained. Here, we define the concept of consistency.

**Definition 2.6: (Consistent Workflow Model).** Workflows model ontology is consistent if and only if:

1. It satisfies all the constraints defined by the ontology;
2. It satisfies all the constraints defined by the rules.
The constraints defined by the ontology itself are the general constraints that a business process model must maintain as a complete and correct model. The domain, range, and characteristics of a property, as well as the subclass relationships, can restrict the business processes from various kinds of mistakes.

The constraints defined by rules are used to validate whether the derived logical model is consistent with the conceptual model. By checking this kind of constraints, we can guarantee that business requirements in conceptual models will be realized in logical models, which will be implemented by physical models. In this chapter, we identify rules from four different perspectives:

1. **Operational rule:** Each functional unit must have at least one corresponding logical task.

2. **Functional rule:** Each logical task can belong to one and only one functional unit.

3. **Informational rule:** Each data object must be generated by the correct logical task.

4. **Behavioral rule:** Conceptual dependencies are preserved in the logical model.

More constraints might be possible for the rule set according to specific user requirements. By translating the two consistency requirements into SWRL rules, automatic validation can be carried out by the KAON2 inference engine. The SWRL representation of the above rules is shown in Table 2. These rules, together with model ontologies, are used as input for consistency checks. If inconsistencies are found between conceptual and logical models, the derived logical model should be revised so that it can conform to all requirements in the conceptual model.
### Table 2. SWRL Constraint Rules

<table>
<thead>
<tr>
<th>Constraints</th>
<th>SWRL rules</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operational</td>
<td>FunctionalUnit(?F) =&gt; (1&lt;= numberOfLogicalTask) (?F)</td>
<td>A functional unit must be implemented by at least one logical task.</td>
</tr>
<tr>
<td>Functional</td>
<td>FunctionalUnit(?F1) ∧ FunctionalUnit(?F2) ∧ Transition(?T) ∧ Correspondence(?F1, ?T) ∧ Correspondence (?F2, ?T) =&gt; sameAs(?F1,?F2)</td>
<td>If a Petri net transition node implements two functional units, we can conclude that the two conceptual tasks are actually the same task.</td>
</tr>
<tr>
<td>Informational</td>
<td>Depencecey (?D)∧ DataObject(?Data)∧ Transition(?T)∧ FunctionalUnit(?F)∧ source (?F,?D)∧ output (?T, ?Data)∧ hasData(?D,?Data) =&gt; Correspondence (?F, ?T)</td>
<td>If a Petri net transition node has an output data object that is generated by a functional unit, we can conclude that this transition node must be derived from the functional unit.</td>
</tr>
<tr>
<td>Behavioral</td>
<td>FunctionalUnit(?F1)∧ FunctionalUnit(?F2)∧ Transition(?T1)∧ Transition(?T2)∧ Dependency(?D)∧ source(?F1,?D)∧ target (?F2,?D)∧ Correspondence (?F1, ?T1)∧ Correspondence (?F2, ?T2) =&gt; dependOn(?T1,?T2)</td>
<td>If functional unit F1 depends on F2, F1’s corresponding logical task T1 must depend on F2’s corresponding logical task T2.</td>
</tr>
</tbody>
</table>

### 2.6 A Case Study and Prototype Implementation

#### 2.6.1 A Case Study

In order to validate our workflow model transformation approach, we conducted a case study on insurance claim processing. The conceptual model of the insurance claim process is the example described in Section 2.4. In this section, we show how to apply the model transformation procedure and derive a logical model that satisfies the requirements in the conceptual model. The first step of model transformation is to derive logical tasks. As we mentioned in Section 2.5.1, this step is mainly based on user input. The functional units and their derived logical tasks in the example are shown in Table 3.
Table 3. Functional Units and the Derived Logical Tasks

<table>
<thead>
<tr>
<th>Functional units</th>
<th>Logical tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1.File claim</td>
<td>C1-1 Choose claim method C1-2 Online claim C1-3 Phone claim</td>
</tr>
<tr>
<td>I1.Process claim</td>
<td>I1-1 Estimate claim I1-2 Manager signature I1-3 Send repair authentication</td>
</tr>
<tr>
<td>I2.Process payment</td>
<td>I2-1 Approve payment I2-2 Pay repair shop</td>
</tr>
<tr>
<td>V1.Repair vehicle</td>
<td>V1-1 Vehicle diagnose V1-2 Repair</td>
</tr>
<tr>
<td>V2.Submit for payment</td>
<td>V2-1 Calculate price V2-2 Send payment information</td>
</tr>
</tbody>
</table>

Then we derive control dependencies by following the methods in Step 2. We first generate control dependencies among tasks that are assigned to different roles by mapping conceptual dependencies to the logical level. Further, control dependencies among tasks that are assigned to the same role are generated based on user input. Tables 4 and 5 show the two types of control dependencies, respectively.

Table 4. Conceptual Dependencies and the Derived Control Dependencies

<table>
<thead>
<tr>
<th>Conceptual dependencies</th>
<th>Derived control dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C1, I1)</td>
<td>(C1-1, I1-1) (C1-1, I1-2) (C1-1, I1-3) (C1-2, I1-1) (C1-2, I1-2) (C1-2, I1-3) (C1-3, I1-1) (C1-3, I1-2) (C1-3, I1-3)</td>
</tr>
<tr>
<td>(V2, I2)</td>
<td>(V2-1, I2-1) (V2-1, I2-2) (V2-2, I2-1) (V2-2, I2-2)</td>
</tr>
<tr>
<td>(I1, V1)</td>
<td>(I1-1, V1-1) (I1-1, V1-2) (I1-2, V1-1) (I1-2, V1-2) (I1-3, V1-1) (I1-3, V1-2)</td>
</tr>
<tr>
<td>(C1, V1)</td>
<td>(C1-1, V1-1) (C1-2, V1-1) (C1-3, V1-1) (C1-1, V1-2) (C1-2, V1-2) (C1-3, V1-2)</td>
</tr>
</tbody>
</table>

Table 5. Control Dependencies among Tasks Within the Same Role

<table>
<thead>
<tr>
<th>Roles</th>
<th>Control dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Claimant</td>
<td>(C1-1, C1-2) (C1-1, C1-3)</td>
</tr>
<tr>
<td>Insurance company</td>
<td>(I1-1, I1-2) (I1-2, I1-3) (I2-1, I2-2)</td>
</tr>
<tr>
<td>Vehicle repair shop</td>
<td>(V1-1, V1-2) (V2-1, V2-2) (V1-2, V2-2)</td>
</tr>
</tbody>
</table>

After removing redundant dependencies from Table 4 and Table 5, we get the set
of control dependencies \( \text{CDS} = \{(C1-2, I1-1), (C1-2, I1-2), (C1-2, I1-3), (C1-3, I1-1), (C1-3, I1-2), (C1-3, I1-3), (V2-1, I2-1), (V2-2, I2-1), (I1-1, V1-1), (I1-2, V1-1), (I1-3, V1-1), (C1-1, C1-2), (C1-1, C1-3), (I1-1, I1-2), (I1-2, I1-3), (I2-1, I2-2), (V1-1, V1-2), (V2-1, V2-2), (V1-2, V2-2)\} \). The business policy specifies how the claim should be handled based on different claim channels (e.g., online or phone). Two business rules are generated according to the policy: \(((C1-1, C1-2), \text{ClaimMethod(online)})\) and \(((C1-1, C1-3), \text{ClaimMethod(phone)})\). Based on the data dependency mapping algorithm, the logical dependencies in Table 6 are derived from conceptual dependencies in the vehicle insurance example. Table 7 contains the data dependencies between logical tasks derived corresponding to functional units. Figure 10 shows the nine patterns that are recognized by the pattern recognition algorithm and the final Petri net model for the vehicle insurance example.

### Table 6. Conceptual Dependencies and the Derived Data Dependencies

<table>
<thead>
<tr>
<th>Conceptual dependencies</th>
<th>Derived data dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>(C1, I1)</td>
<td>(Insurance Info., C1-2, I1-1) (Insurance Info., C1-3, I1-2)</td>
</tr>
<tr>
<td>(V2, I2)</td>
<td>(Price, V2-1, I2-1) (Payment Info., V2-2, I2-1)</td>
</tr>
<tr>
<td>(I1, V1)</td>
<td>(Authorization, I1-3, V1-1)</td>
</tr>
</tbody>
</table>

### Table 7. Data Dependencies among Tasks within the Same Role

<table>
<thead>
<tr>
<th>Roles</th>
<th>Data dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Claimant</td>
<td>(ClaimMethod, C1-1, C1-2) (ClaimMethod, C1-1, C1-3)</td>
</tr>
<tr>
<td>Insurance company</td>
<td>(ClaimEstimation, I1-1, I1-2) (Signature, I1-2, I1-3) (AuditReport, I2-1, I2-2)</td>
</tr>
<tr>
<td>Vehicle repair shop</td>
<td>(Price, V2-1, V2-2) (DiagnoseReport V1-1, V1-2) (Repaired vehicle V1-2, V2-2)</td>
</tr>
</tbody>
</table>
2.6.2 Prototype System Implementation

We implemented the prototype system for model transformation based on the approach proposed in this chapter. As shown in Figure 11, our web-based system includes two components: logical model generator and model validator. The model validator uses KAON2 as the back end reasoning engine. The conceptual and logical models, as well as their correspondence relationships, are stored in a database. The model ontology is also stored in the database for model validation.

The logical model generator is a process modeling environment that provides step-by-step instructions for building logical models based on a given conceptual model. First, users are required to input a conceptual model and then they will be required to go
through each step of model transformation and input additional information for
generating the conceptual model. After enough information is collected from users, the
system can generate the logical model automatically. The model validator relies on
KAON2 as the reasoning engine. The consistency checking function is implemented by
the function Reasoner.isSatisfiable() in KAON2 that checks if the knowledge base is
satisfiable. That is, we can validate whether a derived logical model is consistent with the
original conceptual model. Figure 12 and Figure 13 show the runtime interface of the
logical model generator and workflow validator.

![Figure 11. Architecture of Prototype System](image-url)
Figure 12. Logical Model Generator Interface

Figure 13. Model Validator Interface
2.7 Conclusion

2.7.1 Discussion

Important design science contributions create and evaluate IT artifacts intended to solve identified organizational problems (Hevner et al., 2004). In order to facilitate workflow model design, this research presented a formal approach for transformation from conceptual to logical process models. We first proposed a semi-automated procedure to add information to the conceptual model and transform it to a logical model. Further, the consistency between a conceptual model and its corresponding logical model is checked via an ontology-based approach. To the best of our knowledge, our study represents the first attempt to (a) formally define three layers of workflow models, (b) transform conceptual models to logical models, and (c) check consistency between conceptual and logical models. While this study has direct practical implications for workflow model designers, it may also have theoretical implications.

The model-driven architecture (Atkinson and Kuhne, 2003) is a software design paradigm that provides a set of guidelines and standards for the structuring of specifications, which are expressed as models. Our study adds to the literature on model-driven architecture, and emphasizes that more than two levels of models (i.e., platform-independent vs. platform-dependent) may be needed as modeling tasks become more complicated. We extend the model-driven architecture to the domain of workflow modeling and propose to use conceptual and logical models as platform-independent models and physical models as platform-specific models. The transformation between
models of different levels of abstraction such as platform-independent models and platform-specific models is a critical step of system development in model-driven architecture.

Our work also serves to motivate a theory of workflow model design. At present, little is known about workflow modeling practice and workflow modeling procedure (Recker and Mendling, 2006). As a relatively young field in software engineering, workflow design in the industry has not adopted a standard process similar to what has been done in database design. This is unfortunate since the three-layer approach in database design has been widely adopted in the industry with great benefits. However, how to standardize the workflow design process remains an open question since it is not yet widely agreed if workflow design should adopt a layered approach and if workflow design tasks should be layered as we suggested in this chapter. We believe that we have laid the groundwork for extensive theoretical and empirical research into workflow model design. Some of the conjectures that can be derived from our research (e.g., efficiency of the layered workflow modeling approach, the relationship between different layers of models, the design principles and guidelines for workflow modeling, etc.) call for further investigation. In particular, future research could address the potential benefits and tradeoffs of the layered workflow modeling approach that has been identified in this chapter.
2.7.2 Limitations and Future Work

We identify two limitations of this study. First, we limited this study to popular process modeling patterns (e.g., five basic workflow patterns) and techniques (e.g., Petri nets and DND). We believe it is representative of the most popular techniques based on earlier studies. The smaller scope enables us to focus our work and to avoid too many extra constraints. Although the general model transformation and validation framework can be applied to other modeling languages, the detailed steps might need to be altered according to different modeling languages. Second, our approach is semi-automatic and relies heavily on user input of additional information. This could introduce potential errors and inaccuracies to the transformation process. More effort may be needed to automatically extract additional information from other sources such as business policies and rules.

In our future research, we intend to extend our work in two directions. First, while the correctness of the proposed model transformation approach is validated by a case study, user studies are needed to assess the effectiveness of such systems in practice. These studies must address a bevy of issues, including appropriate user interface design, methods for enhancing the perceived usefulness of the system, mechanisms for error alerting messages when interacting with users, etc. Second, we will explore the relationships between logical models and physical models that are system-dependent and propose a design theory for layered workflow model design.
3.1 Introduction

Collaboration is ubiquitous in human activities. Through teamwork, people share resources and join their efforts to deliver outputs beyond what individuals can achieve (Nunamaker et al., 2001). It has always been a critical problem on the understanding and management of collaborations for teamwork success (Forrester, 2009). Previous literature has accounted teamwork success to several organizational, operational, and communicational factors, including team composition (Gladstein, 1984), social network structure (Faraj and Johnson, 2011), communication media selection (Dennis and Valacich, 1999), among others. From the perspective of collaboration processes, existing studies generally admit its critical role in teamwork (Morgeson et al., 2010). However, they provide limited instruments to capture characteristics of collaboration process. It remains a challenging problem to understand and model collaboration processes due to its complex and ad hoc nature (Thomson and Perry, 2006).

Software development is one critical collaborative task that worth studying from this perspective (Dean et al., 1997). In software development projects, project managers usually coordinate team collaboration processes based on best practices and/or personal experience (Grol and Grimshaw, 2003; Wagner et al., 2006). Such a subjective approach
can be effective in some projects and ineffective in others depending on projects’ circumstances (Jones, 2004). There is a lack of formal quantitative methods to assess the effect of such “best practices.” In fact, many software development projects face challenges and are late and/or over budget from their original plans (Crowston et al. 2003; Wallace and Keil 2004). Among the various reasons that cause these failures, the effectiveness of interaction process is one factor that is always under researchers’ inspection (Antolić, 2008).

In practice, the collaboration process of a project is often documented as a sequence of interactions among participants (Bertram et al., 2010). However, the sequential order captures individuals’ back-and-forth interactions, i.e., how the collaboration process is organized. According to the coordination theory, people interact since there are dependencies among their activities and resources (Malone and Crowston, 1994; Malone et al., 1999). The dependencies are due to organization restrictions, resource limitations, and task requirements, which may be shared by multiple collaboration instances. Under their influences, people’s interaction and collaboration processes across projects should show some recurrent structures. Such routines are usually summarized qualitatively by domain experts (Halverson et al., 2006).

In this research, we aim to discover patterns from interaction sequences in project tracking logs and use it as a quantitative instrument to understand collaboration. We conceptualize frequent-occurred structures hidden in interaction sequences as “collaborative interaction process patterns” (or “interaction patterns”). A pattern is “the
abstraction from concrete form which keeps recurring in specific non-arbitrary contexts" (Riehle and Züllighoven, 1996). For example, enterprise design patterns are used to describe reusable solutions to recurring problem in software development (Grand, 2002). While interaction patterns share the spirits in reusability and abstraction with other types of patterns, they specifically focus on the process perspective and capture the fundamental dependencies among people in processes. Interaction patterns are components to build up each interaction process, which reflect fundamental dependencies among participants. The patterns discovered from data may not be as detail as ones extracted from interview or other qualitative methods. However, they provide fast and direct assessment of collaboration process.

We extend previous efforts on sequence and graph mining (Han et al., 2000; Yan and Han, 2002) and design an analytical framework for collaboration pattern extraction. We examine the impact of interaction patterns on the time efficiency in software issue resolution in open source community (i.e., time cost on software issue resolution), which is one important dimension of teamwork success (Kirkman and Mathieu, 2005). We apply the framework on a dataset collected from open source software projects and identify interesting interaction patterns that affect teamwork efficiency, which has not been fully addressed in exiting studies.

As one major contribution of this paper, our proposed analytical framework supports the identification and assessments of interaction patterns on collaboration time efficiency. In the context of software development, our identified collaboration patterns
can be employed in future software development management and collaboration tool design. The rest of this chapter is organized as follows. Section 3.2 reviews related work in the literature. Then we introduce the dataset that is used in this study in Section 3.3. Section 3.4 describes what interaction patterns are, and Section 3.5 proposes the procedure for interaction pattern mining. In Section 3.6, we introduce the methodology for pattern impact assessment. Then, we apply this methodology to the software issue resolution dataset and empirically analyze the effects of identified interaction patterns in Section 3.7. Finally, we conclude this chapter by discussing the implications and limitations in Section 3.8.

3.2 Literature Review

3.2.1 Dependency and Interaction Patterns in Collaboration

A collaboration process usually contains a sequence of interactive activities among team members. From the perspective of coordination theory, a group of people interact with each other because there are dependencies between activities, between resources, and between activities and resources (Malone and Crowston, 1994; Malone et al., 1999). Such dependencies cause people’s complicated interactions and recurrent interaction structures. For example, previous research (Salomon, 1992) found that there is genuine interdependence in collaborative learning; genuine interdependence is described as the necessity of resource sharing and joint thinking. Thus, the inherent dependency of collaboration may lead to certain recurrent structures of interaction activities or
"patterned relations" (McGrath, 1984). Recently, researchers have proposed to process patterns in different collaboration contexts. For instance, van der Aalst et al. (2003) abstract workflow patterns as routines in dealing with dependencies in business tasks, which can be used as basic building components for workflow modeling. Briggs et al. (2003) use thinkLet to encapsulate communication patterns in decision making on group decision support systems. As compared with ad hoc activities, these recurrent interaction structures reflect activities that may happen in future collaboration processes and thus are worth more effort to manage and optimize (Briggs et al., 2003). Liu and Ram (2011) have investigated how the collaboration pattern derived from data provenance clustering may affect the quality of their co-edited wiki pages.

In software development, researchers have proposed several types of process patterns from different perspectives. Aranda and Venolia (2009) studied the bug-fixing cases at Microsoft and identified common coordination patterns from dimensions such as communication media and meeting type. Guo et al. (2011) did a large-scale quantitative and qualitative analysis of the bug reassignment process in the Microsoft Windows Vista operating system project and studied the effect of task assignment patterns. Through interviews with industry and open source programmers, Halverson et al. (2006) identified patterns of problematic behavior in software development. Social network structures that are derived from email logs are also proposed as interaction patterns to study knowledge sharing in virtual teams (Gloor et al., 2003). However, such process patterns have not been fully conceptualized quantitatively and investigated as an instrument to understand
collaboration management. Existing literature still uses static and abstract features to characterize the dynamic collaboration processes. To the best of our knowledge, we are the first to study the process structure patterns in software development.

3.2.2 Team Efficiency in Software Development

Previous research suggests that interaction structure caused by dependencies in collaboration may affect collaboration outcomes (Horton, 1993). Collaboration performance can be measured in different ways. In this research, we inspect the efficiency perspective of process performance. Efficiency refers to producing a specific outcome with a minimum amount of expense, waste or effort (Bstieler, 2005; Rhee et al., 2007). Time is a common metric of efficiency in software development (Antolić, 2008).

Traditional research (Gladstein, 1984) on group performance outlines many factors that affect collaboration process efficiency, including team composition, high-level process characteristics (e.g., leadership style, well-organized or not), tool support (Lowry et al., 2003; Lowry and Nunamaker, 2003), and interaction media. Role clarification is one fundamental rule that should be applied during the collaboration process. The team performance model (Drexler et al., 1988) argues that role clarification helps collaborators know what to do and what the goal is, which are essential for group commitment. Among the different roles in a team, team leaders’ behavior is very important in collaboration processes. They affect the effectiveness of virtual teams,
especially for dispersed teams. Team leaders’ actions, such as technology chosen and team members’ training (Kirkman et al., 2002), are critical to the success of virtual teams.

Software development is one inherent collaborative task that requires software engineers to coordinate while producing software systems (Faraj and Sproull, 2000). In software projects, project managers usually coordinate team collaboration based on best practice and/or personal experience (Grol and Grimshaw, 2003; Wagner et al., 2006). Studies of OSS projects have found that many factors strongly affect the success of collaboration in software development, such as motivation, goal clarification, membership size, the degree of supervision by community owners (Nakakoji et al., 2005). In addition, the importance of collaboration processes in software development has also been recognized by previous researchers (Florac et al., 2000).

However, existing research on collaboration efficiency mainly focuses on the impacts of team and high-level process characteristics on team performance. Few studies have explicitly considered the effect of the collaboration process structure (Lowry et al., 2005), which is a key construct we will study in this chapter.

3.2.3 Pattern Mining from Sequences

Interaction pattern mining is extracting a set of process patterns for a set of process instances. Few studies have addressed this problem directly. From the technical perspective, extraction of interaction patterns is related to three areas of work: sequential
pattern mining (Han et al., 2000), process mining (van der Aalst and Weijters, 2004), and graph pattern mining.

Sequential pattern mining (Agrawal and Srikant, 1995) seeks to discover frequent sequential occurrences of activities in transactional data. The Apriori algorithm was first proposed to address the problem in 1995 (Agrawal and Srikant, 1995). Since then, this problem has received a great deal of attention and many extensions have been introduced (Han et al., 2007; Han et al., 2000). Sequential pattern mining has been widely applied to many practical contexts, such as intrusion detection (Hu and Panda, 2004) and DNA analysis (Han et al., 2001). However, collaboration processes usually include complicated structures, such as loops and branches, as in business process models. It is necessary to identify substructures from such a graph-based collaboration process, which is not covered by sequential pattern mining.

Process mining deals with the problem of rebuilding a process model from process logs representing different process instances. In process mining, process modeling languages such as directed graphs (Hwang and Yang, 2002), finite state machines (Datta, 1998), and Petri nets (van der Aalst et al., 2004) are used for representing process models. The goal of process mining is to discover a process model that best describes the set of process instances. This differs from our study because we look for common and frequent structures across multiple process instances while process mining assumes the existence of a unified process model that can describe all process instances.
Graph pattern mining can be classified into two categories according to the definitions of the problem (Koyutürk et al., 2004). The first type of graph pattern mining algorithms try to find isomorphic substructures that are independent of labeling in a collection of graphs (Yan and Han, 2002). This approach is well-suited to applications focused on the structure of relationships between entities. This is a particularly challenging problem as it relates to the NP-hard subgraph isomorphism problem (Inokuchi et al., 2003). The second type of graph pattern mining algorithms focus on finding frequent patterns that have both the entities (node labels) and relationships between entities (graph structure) in common (Cook and Holder, 1994). Many approaches have been developed to mine interesting subgraph patterns from graph datasets. These include mathematical graph theory-based approaches such as gSpan (Yan and Han, 2002) and greedy search-based approaches like Subdue (Cook and Holder, 1994). In interaction pattern mining, we need to consider labels of nodes because nodes can represent the types of activities, which are essential to collaboration management. The major difference between graph pattern mining and interaction pattern mining is that not all subgraphs of a process graph are subprocesses.

Table 8 compares the three areas mentioned above with interaction pattern mining. Although none of these pattern mining approaches can fully satisfy the requirements of interaction pattern mining, they provide a solid theoretical basis for our pattern mining approach.

Table 8. Comparison of Different Pattern Mining Approaches

<table>
<thead>
<tr>
<th></th>
<th>Sequential</th>
<th>Process mining</th>
<th>Graph pattern</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 3.3 Dataset

For this research, we collected collaboration process data from an open source software community dedicated to administrative software development. The community had successfully released four software packages by 2010. The community uses an online collaboration platform, Jira (www.jira.com), to track its software issue resolution processes. The system runs following an issue resolution procedure, where someone raises bugs in the software and someone else fixes the issue. Each issue has some predetermined attributes, such as priority and issue type. For each issue, multiple users may be involved in a specific sequence. In this process, developers record their “actions,” such as coding changes, module modifications, and idea exchanges using the system. An action can also be a simple comment. If an action is a concrete change, the developer can fill in multiple “change items” to describe the things done under the name of one action. Different developers may play different roles in the process. We take this software issue resolution process as a representative collaboration process in software engineering and analyze how interaction patterns may affect process efficiency (i.e., issue resolution time). Figure 14 shows an example of issue tracking in the context of open source software development.

<table>
<thead>
<tr>
<th><strong>Input</strong></th>
<th><strong>Output</strong></th>
<th><strong>Input</strong></th>
<th><strong>Output</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A set of activity sequences</td>
<td>A set of sequential patterns</td>
<td>A graph or a set of graphs</td>
<td>A set of graph patterns</td>
</tr>
<tr>
<td>A set of activity sequences</td>
<td>A graphical process model</td>
<td>A set of graph patterns</td>
<td>A set of graph patterns</td>
</tr>
</tbody>
</table>
Our dataset contains the community’s process data from February 2005 to June 2009. In this period, there are 14,049 issues on the Website. Each issue takes from a few days to a couple of years to fix. Since some issues have dependency relations, we preprocessed the data and kept only the atomic issues whose resolution does not depend on other issues’ resolution. We also removed issues of duplicate reporting (by different testers), wrong reporting, and features that will not be included in the software. After preprocessing, the dataset contains 7,991 unique atomic issues that the development team intends to address, including 6,755 fixed issues and 1,236 in-progress issues.
3.4 Interaction Patterns

Collaboration is a process in which autonomous individuals interact through formal and informal interactions (Thomson, 2001). Because of the dynamic nature of collaboration processes, the structure of such a process is very complex and ad hoc. As a result, it is very difficult to analyze entire collaboration processes. The research in pattern mining proposes to identify frequent substructures from a set of dynamic and complicated datasets that can be used for various analyses (Inokuchi et al., 2003). Recently, some studies pointed out that traditional pattern mining approaches only consider the frequency of substructures and this restricts the pattern mining algorithms to discovering only small patterns (Han et al., 2007). If a pattern is frequent, all of its subpatterns are frequent as well; thus a large pattern will contain an exponential number of smaller, frequent subpatterns.

In this research, we formalize each interaction sequence for a project/task as a process instance. Each interaction instance is composed of a sequence of actors, which tracks how people get a collaboration task done through interactions. While actors and can be described from many aspects, in this paper, for the sake of simplicity, we focus on the organizational perspective of actors and describe an actor from two dimensions: the role and the actor identifier that uniquely represents an actor (e.g. name or id number).

**Definition 3.1: (Actor).** An actor in collaboration can be represented a tuple $a_i = (r_i, m_i)$, where $r_i$ is the role of the actor, $m_i$ is the actor identifier.
Definition 3.2: (Interaction Process Instance). A process instance PI is a sequence of interactions among actors to accomplish a particular task, which can be represented as an ordered list \( \text{PI} = \langle a_1, \ldots, a_n \rangle \).

Actor’s role is a fundamental reason on what one can do and how one may interact with others. The effect of actor identity, however, is tricky. On one hand, actor identity is needed to differentiate individuals’ efforts within a process instance. On the other hand, role is a more substantial measure than identity when mining patterns from process instances. In most of the collaboration tasks, a specific individual should not be considered as a part of business routine/logic. Thus, in this research actor identity is only used to differentiate actors and not to characterize the semantics of patterns. For example, process instance \( \langle (\text{Alan}, \text{Manager}, \text{Modification} \ldots), (\text{Billy}, \text{Worker}, \text{Modification} \ldots), (\text{Charlie}, \text{Worker}, \text{Modification} \ldots), (\text{Billy}, \text{Worker}, \text{Modification} \ldots), (\text{Charlie}, \text{Worker}, \text{Modification} \ldots) \rangle \) contains five activities conducted by three individuals. It starts with a manager’s work followed by two workers’ iterative work. This understanding can be gain without knowing the specific names of the three individuals.

Definition 3.3: (Process Segment). For a given process instance \( \text{PI} = \langle a_1, \ldots, a_n \rangle \), a process instance segment is one of its subsequence \( \text{SEG} = \langle a_m, a_{m+1}, \ldots, a_{m+k} \rangle \) which meets the condition \( m \geq 1 \) and \( m+k \leq n \).

Here we take process instance segments as a basis to extract interaction pattern. In practice, interaction process instances are seldom identical. However, due to the inherent dependency among tasks, people and resources, collaborators often use some routines to
solve similar problems. There should be significant among of segments reflecting such routines in process instances. It is obvious that a process instance is also a process segment.

An organization may have multiple interaction process instances. An actor or certain combinations of actors can appear multiple times in an interaction process instance or across process instances. The iterative appearances of certain interaction structure generally reflect the requirement of some fundamental dependencies. Thus, we abstract the interaction relations between actors as a representation of dependencies from multiple (segments of) process instances.

Definition 3.4: (Actor Dependency Graph). An actor dependency graph of a process segment ($SEG$) is a directed graph $ADS = (V; E; L)$, where
- $V$ is a set of activities in $SEG$
- $E = \{(a_i, a_j)\}$, where $a_i, a_j \in V$, $a_i \neq a_j$, and $a_j$ directly follows $a_i$ in $SEG$
- $L: L(a_i) \rightarrow \mathbb{R}$ is the label function that maps nodes to roles.

Actor dependency graphs are graphical representation of process segments and abbreviate how actors interact with each other to conduct the task. To address the actor identity concern, we re-label each actor’s identity during this process. The generated identity reflects the temporal sequence one gets involved into collaboration. This relabeling process allows us to match people with different roles across process instances while keeping the ability to differentiate individuals in a process. Although the actor dependency graph ignores some details of the process instances, we consider it highlights
the temporal relationship between actor interactions, which is critical for this study. It is obvious that multiple different process instance segments can be mapped to a same actor dependency graph.

**Definition 3.5: (Interaction pattern).** An interaction pattern is an actor dependency graph satisfying two conditions:

- Its process instance segments frequently occur across processes instances.
- Its appearance is independent from other interaction patterns.

“Frequent” means patterns should appear a sufficient amount of times that is normally defined by a threshold value. Although the definitions of patterns in different contexts vary dramatically, frequent occurrence is always considered as the most important feature of patterns (Han et al., 2007; Hwang et al., 2004). The “frequent occurrence” feature of patterns allows us to focus on analyzing the most important parts of collaboration processes that have a high probability to occur in team collaboration.

“Independent” means a pattern should have sufficient occurrences where the pattern is embedded in other patterns. If a small pattern is always embedded within one or several large patterns, its impact on the collaboration process is dominated by the large patterns. Thus, in order to reduce the interdependencies among patterns, we propose that interaction patterns should have independent occurrences. Moreover, the process structure that is always embedded within large patterns has high correlations with the large patterns. Therefore, if we analyze highly correlated process patterns in the same statistical model, the result might not be accurate. These two requirements will be
incorporated into threshold values in the pattern mining procedure so that meaningful patterns can be identified.

Next, we will use an example to show the relationship between interaction process instances and their possible patterns in the context of software issue resolution in open source community. Assume that we have five process interaction instances and we use the threshold value of 0.5 for the pattern frequency requirement, which means a pattern should at least occur in two out of four processes. All the four process instances are for bug fixing and two types of roles (i.e., manager and workers) are involved in the processes. In the first process, one manager and four workers work together and the manager only initiates the process and leaves the rest to the four workers. Among the four workers, Bob and Aaron interact with each other a few times. In the second process, one manager and three workers are involved. The manager Jeff interacts with Aaron and David a few times. In the third process, only four workers work for this case and only one back-and-forth interaction occurs. The manager did not perform any activity in this process. In the fourth process, one manager and two workers interact with each other most of the time.

If we use the threshold value of 0.5, ten frequent graphs can be extracted from the four interaction processes, but only three of the ten frequent graphs are independent from other patterns. For example, although process structure M→W occurs in three process instances, all the three occurrences are depending on patterns M→W↔W or W→W↔>M. Thus, we do not consider process structure M→W as a pattern because it does not
satisfy the requirement of “independent”. Therefore, only three patterns can be identified from Figure 15. Dashed lines on interaction processes outline the frequent and independent occurrences of patterns in processes.

**Figure 15. Interaction Process Instances and Their Patterns**

3.5 Interaction Pattern Mining

Finding all the frequent patterns from huge datasets is a very time-consuming task. In the general case, the examination of all possible combinations is intractable, and
efficient algorithms are required to focus on those sequences that are considered important to an organization. In order to achieve our goal of pattern mining, we divide the pattern mining procedure into three phases (see Figure 16). In phase 1, we extract process graphs that occur frequently in interaction processes. Specifically, we indentify the frequent graph structures by examining all possible subsequences of interaction processes and counting the occurrence of each process graph. In phase 2, we divide frequent graphs into groups so that independent occurrences of patterns in the same group can be calculated in parallel. Given that the time complexity of calculating independent occurrences is very high, this will enhance the performance of our algorithm dramatically. In phase 3 we select a set of frequent graphs that are not embedded in other frequent graphs in interaction processes as patterns. We generate the pattern set by adding frequent graphs identified in phase 1 iteratively and calculating their occurrences that are independent of other graphs in the set. Process graphs that always occur as a portion of other patterns will be removed in pattern mining.
For instance, graph A and B are identified as frequent graphs in phase 1. Graph A is a subgraph of graph B and graph A is always embedded in graph B in interaction processes. In that case, we want to exclude graph A from the pattern set because it is just a portion of graph B. Since graph B is already included in our analysis, analyzing graph A will be meaningless. Next, we will formally define the pattern mining problem and related concepts.

**Definition 3.6: (Support).** An actor dependency graph ADG is supported by a process instance PI if ADG is a graph of a subsequence of PI.

If an actor dependency graph ADG is supported by a process instance PI, then a portion of the interaction process is carried out by following the dependency defined by ADG. When supported by many process instances, this actor dependency graph will be considered as a frequent graph that is used in many interaction processes. We consider...
these frequent graphs as the common interesting units for process analysis. These graphs are more important than those of non-regular processes because they are adopted a lot by teams and have significant impact on team collaboration.

**Definition 3.7: (Frequent Graph).** Given a set of process instances, an actor dependency graph ADG is said to be frequent if it is supported by no less than $\alpha\%$ of the process instances, where $\alpha\%$ is a user-defined minimum support threshold.

Algorithm 3.1 implements the phase of extracting frequent graphs in Figure 16. It scans all interaction process instances and finds out all process graphs in each process instance. If a graph is already in the graph set, its count will be increased by one. Otherwise, it will be added to the graph set and its count will be initialized as one. If a graph is supported by many process instances and its count exceeds the predefined threshold value, it will be identified as the frequent graph.

**Algorithm 3.1 Frequent Graph Extraction**

\[\text{Extract\_frequent\_graph}(\text{PIS, } \alpha)\]

//Enumerate all subgraphs that are supported by PI in PIS
Frequent Graph Set FGS=$\Phi$
Count $C=0$
For each process instance PI
  ADGS$_{PI}=$(ADG\_Extraction(PI)) \text{//see algorithm 3.2}
  FGS=FGS $\cup$ ADGS$_{PI}$
  For each ADG in ADGS$_{PI}$
    $C_{ADG,PI}=1$
  Next ADG
Next PI
//Calculate support and remove infrequent pattern candidates
For each graph ADG in FGS
  Support $S_{ADG}=$ $\sum_{PI}(C_{ADG,PI})$
  If $S_{ADG}<\alpha$
    FGS=FGS - ADG
Algorithm 3.2 is an important part of Algorithm 3.1. It describes how to extract all subgraphs of each process instance. It enumerates all subsequences of a process and converts them to graphs. It returns a set of graphs that support the process instance.

Algorithm 3.2 ADG Extraction

\[
\text{ADG\_Extraction}\ (\text{PI})
\]

\[
\text{ADGS}=\Phi
\]

//enumerate all subsequences in a process instance
For \text{i}\text{th} actor in PI (from 0 to n)
  For \text{j}\text{th} actor in PI (from \text{i}+2 to n)
    \text{SQ}=\text{Sub-seq}(\text{PI}, \text{i}, \text{j})
    \text{ADG}=\text{Convert}(\text{SQ}) \quad //\text{convert process segment to graph according to Definition 3.4}
    \text{If ADG} \notin \text{ADGS}
      \text{ADGS}=\text{ADGS} \cup \text{ADG}
    \text{End If}
  Next j
Next i
Return \text{ADGS}

End ADG\_Extraction

Our goal of pattern mining is to find out the set of graphs that have enough independent occurrences in the interaction process. Algorithms 3.1 and 3.2 cannot guarantee that occurrences of graphs are independent. The graphs identified in phase 1 may be subgraphs of each other. We define the relationship of dependency subgraph as follows:
Definition 3.8: (Dependency Subgraph). An actor dependency graph $ADG' = (V', E', L')$ is a subgraph of another actor dependency graph $ADG = (V, E, L)$ if and only if $V' \subseteq V$, $E' \subseteq E$ and $\forall a_i \in V'$, $L'(a_i) = L(a_i)$.

Among those frequent process graphs identified by Algorithm 3.1, we are only interested in those process graphs that exist in interaction processes independent of other frequent graphs. Therefore, with Definitions 3.9 and 3.10, we can further filter out actor dependency graphs that appear within other graphs.

Definition 3.9: (Graph Inclusion). In a process instance $PI$, an actor dependency graph $ADG$ is considered to be included in an actor dependency graph set $ADGS$ ($ADG \not\in ADGS$), if for any segment $SEG$ of $PI$ mapped to $ADG$ there exists a segment $SEG'$ mapped to $ADG' \not\in ADGS$ so that $SEG$ is a subsequence of $SEG'$.

A graph $ADG$ is included in another graph $ADG'$ means the occurrence of graph $ADG$ depends on the occurrence of graph $ADG'$. Further, graph $ADG$ only represents a portion of graph $ADG'$. When graph inclusion happens, we do not count the included graph as a supported pattern. This is consistent with the pattern mining literature that often tries to find the maximal common subunits out of a set of instances (Agrawal and Srikant, 1995; Hwang et al., 2004). Figure 17 shows an example of graph inclusion.
Definition 3.10: (Independent Occurrence). Given a set of actor dependency graphs ADGS, an actor dependency graphs ADS $\not\in$ ADGS independently occur in the process instance PI, if and only if:

- ADG is supported by PI;
- ADG is NOT included by ADGS in PI.

With definition of independent occurrence of patterns, the interaction pattern mining problem can be defined as follows:

**Definition 3.11: (Interaction pattern Mining).** Given a set of process instances PIS and frequent graphs ADGS, the problem of pattern mining is to find the complete set of interaction patterns (PTS) where each interaction pattern independently occur no less than $\beta\%$ of the process instances; $\beta\%$ is a user-defined minimum support threshold.

Because the complexity of counting independent occurrences is very high, we propose a grouping approach to divide frequent graphs into groups and count their independent occurrences. Lemma 3.1-3.4 proves the correctness of the grouping method.
Lemma 3.1. For two process segment SEG and SEG’, if SEG’ is a subsequence of SEG, then ADG(SEG’) is a subgraph of ADG(SEG).

Proof. Suppose ADG(SEG’)=(V’, E’, L’) and ADG (SEG)=(V, E, L). Because SEG’ is a subsequence of SEG, V’⊆V, E’⊆E and ∀aᵢ∈V’, L’(aᵢ)=L(aᵢ). According to Definition 3.8, Graph(S’) is a ADG(SEG’) is a subgraph of ADG(SEG).

Based on Lemma 3.1, we know that graphs of subsequences of a process are subgraphs that constitute a portion of the whole interaction process. These subgraphs are fragments of the interaction process structure and each of them describes how part of the interaction process is carried out. Note that the opposite of Lemma 3.1 does not necessarily hold because there is no one-one mapping relationship between process subsequence and graph.

Lemma 3.2. If an actor dependency graph ADG’ is a subgraph of another actor dependency graph ADG, adding ADG’ to pattern set ADGS (ADG’∉ ADGS) will not affect the count of independent occurrence of ADG.

Proof. Without loss of generality, we use PI to represent an arbitrary process instance. We divide into two cases:

Case 1: If ADG does not have independent occurrence in PI under ADGS, according to Definition 3.10, ADG is either not supported by PI or included by ADGS in PI. Adding ADG to ADGS will not change any of these conditions. So ADG does not have independent concurrence in PI under ADGS∪ADG’.
Case 2: If ADG has independent occurrence in PI under ADGS, according to Definition 3.10, ADG is supported by PI and not included by ADGS in PI. Since ADG$' \subseteq$ ADG, ADG$\not\subseteq$ ADG$. So ADG cannot be included in ADG$. Therefore, ADG cannot be included in ADGS$\bigcup$ADG$. ADG satisfies the definition of independent occurrence in PI under PI under ADGS$\bigcup$ADG$'$.

Lemma 3.2 means that if there is a subgraph relationship between two graphs, we should count the unique occurrence of the super graph pattern first.

**Lemma 3.3.** If an actor dependency graph ADG$'$ is NOT a subgraph of ADG, whether ADG is in ADGS will not affect the count of independent occurrence of ADG.

**Proof.** Proof of Lemma 3.3 is similar to that of Lemma 3.2 and detailed proof is omitted because of space limit.

**Lemma 3.4.** If there is no subgraph relationship between two actor dependency graphs, their independent occurrence support will not be affected by each other.

**Proof.** Assume we have two actor dependency graph ADG and ADG$, ADG\not\subseteq ADG$ and ADG$'\not\subseteq ADG$, by applying Lemma 3.3 twice and Lemma 3.4 is achieved.

According to Lemma 3.4, we know that actor dependency graphs that are not subgraphs of each other can be considered together for independent occurrence calculation. Thus, we can divide frequent graphs into groups according to their subgraph relationship and iteratively examine independent occurrences of graphs. We select all graphs that are not subgraphs of any other graphs in frequent graph sets as the first group.
Then the first group of graphs is removed from the frequent graph set. The above steps are repeated to select the rest of the groups until the frequent graph set is empty.

Algorithm 3.3 implements Phase 2&3 in Figure 16. It first divide frequent graphs into groups and then extracts a set of interaction patterns that have frequent independent occurrences. It iteratively adds groups of graphs to the pattern set. For each round, the count of each graph in the pattern set will be recalculated and graphs with low independent occurrences will be removed from the pattern set.

**Algorithm 3.3 Pattern Mining**

```
Pattern_Mining(PIS, FGS, β)

Patten Set PS= PG.0
//Divide graphs in FGS into groups according to the subgraph relationship
Graph group set GS= Φ
Group index i=0
While (FGS is not empty)
    For all graphs ADG in FGS
        If ADG is not a subgraph of any other graph ADG’ in FGS
            Add ADG to ith group in GS
            Remove ADG from FGS
        End IF
    Next ADG
    i++
// Add groups of graphs to pattern set iteratively
For kth group of graphs in GGS
    PS = PS ∪ GGS_k
    Count C=0
    For each process instance PI
        Subgraphs IGS_{PI}= Independent_Occur (PI, PS) // see algorithm 4
        For each ADG in IGS_{PI}
            C_{ADG,PI}=1
        Next ADG
    Next PI
//Recalculate support and remove infrequent unique candidates
For ADG in PS
    Calculate support: S_p=Σ_p(C_{p,PI})
```
Algorithm 3.4 is an important step in Algorithm 3.3, and it calculates whether a graph is uniquely supported by a process instance. It examines all frequent graphs supported by the instance and compares the indexes of different graphs. If all indexes of a graph are included in indexes of other graphs, it will not be uniquely supported by this instance. Otherwise, it will be considered as uniquely supported by the instance.

**Algorithm 3.4 Calculating Independent Occurrence**

\[
\text{Independent}_\text{Occur} \left( \text{PI}, \text{PS} \right) \\
\text{IGS} = \emptyset \\
\text{Set of indexed Subgraphs INDEXS} = \emptyset \\
\text{//enumerate all subsequences in a process instance and store indexed subgraphs} \\
\text{For } i \text{th actor in PI (from 0 to n)} \\
\text{For } j \text{th actor in PI (from i+2 to n)} \\
\text{SQ} = \text{Sub-seq(PI, } i, j) \\
\text{ADG} = \text{Convert(SQ)} \text{ // convert process segments to graph according to Definition 3.4} \\
\text{If } \text{ADG} \in \text{PS} \\
\text{INDEXS} = \text{INDEXS} \cup \text{(ADG, } i, j) \\
\text{End If} \\
\text{Next } j \\
\text{Next } i \\
\text{// Remove indexed subgraphs that are included in other subgraphs} \\
\text{For } i \text{th INDEX in INDEXS (0...m)} \\
\text{If } \text{INDEX} \_i \text{ is embedded in any other subgraphs in PS} \\
\text{INDEXS} = \text{INDEXS} \setminus \text{INDEX} \_i \\
\text{End If} \\
\text{Next } i \
\]
// Patterns that have indexed subgraphs remaining will be counted.
// For each subgraph ADG in PS
// If (ADG ∉ IGS && (ADG,x,y) ∈ INDEXS)
// IGS = IGS ∪ ADG
// End If
Next ADG
Return IGS
End Independent_Occur

Next, we use the example in Figure 18 to illustrate the pattern mining procedure. Note that this example is significantly easier than the one in Figure 15 in terms of process complexity and length. We use simple examples here because we want to show the whole procedure of pattern mining. The complexity of the procedure will increase dramatically if the complexity of process increases.

Assume that we have three process instances in our dataset and the each of them contains a sequence of interaction activities among participants. Assume we use 2/3 as our threshold value for pattern mining. In phase 1, we find five frequent graphs that occur more than once in the three instances. In phase 2, we first divide the five frequent graphs into three groups. In the first iteration of pattern mining, graph M→W↔W is counted twice and they satisfy the threshold value. In the second iteration, two more graphs are added to the pattern set. But graphs M→W→W and W↔W have lower frequencies than the threshold value. Therefore, they are removed from the pattern set. In the third iteration, we add W→W and M→W and calculate the independent occurrences. The final result only contains two patterns that satisfy the threshold value.
97

![Diagram of interaction pattern mining procedure](image)

**Phase 1**

<table>
<thead>
<tr>
<th>Frequent Graphs</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>W → W</td>
<td>3</td>
</tr>
<tr>
<td>M → W</td>
<td>3</td>
</tr>
<tr>
<td>W → W</td>
<td>2</td>
</tr>
<tr>
<td>M → W → W</td>
<td>2</td>
</tr>
<tr>
<td>M → W → W</td>
<td>2</td>
</tr>
</tbody>
</table>

**Phase 2**

<table>
<thead>
<tr>
<th>Group ID</th>
<th>Process Graphs</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>M → W → W</td>
</tr>
<tr>
<td>G2</td>
<td>W → W</td>
</tr>
<tr>
<td>G3</td>
<td>W → W</td>
</tr>
</tbody>
</table>

**Phase 3**

**Result of Iteration 1**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>M → W → W</td>
<td>2</td>
</tr>
</tbody>
</table>

**Result of Iteration 2**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>M → W → W</td>
<td>2</td>
</tr>
<tr>
<td>W → W</td>
<td>0</td>
</tr>
<tr>
<td>M → W</td>
<td>0</td>
</tr>
</tbody>
</table>

**Result of Iteration 3**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>M → W → W</td>
<td>2</td>
</tr>
<tr>
<td>W → W</td>
<td>2</td>
</tr>
<tr>
<td>M → W</td>
<td>0</td>
</tr>
</tbody>
</table>

**Figure 18. An Example of Interaction Pattern Mining Procedure**

3.6 Pattern Impact Assessment

Our research focuses on time efficiency to assess the impact of interaction patterns on software development. Efficiency is widely used to describe the extent to which time or effort is well-used for the intended task (Faraj et al., 2011). It often refers
to producing a specific outcome with a minimum amount of expense, waste, or effort (Bstieler, 2005; Rhee et al., 2007). Efficiency in software development can be measured by various metrics, such as time and project cost (Antolić, 2008). In this research, we focus on the time issue and employ task-processing time (i.e., issue resolution time in our dataset) as the representative variable. The more time used in finishing a task, the less efficient the process is. This measure has been used in several previous studies (Faraj and Sproull, 2000; Jones, 1996).

Here we employ the time-dependent Cox regression to model the impact of interaction patterns. The time-dependent Cox regression model extends the Cox regression model by considering covariates that change over time and assessing how the covariates affect the hazard for an event to happen. It is appropriate for our research for two reasons. First, our problem is naturally a survival analysis task. In the dataset, there are closed and open issues, both associated with a processing time. While each issue will eventually be finished, their processing time may be affected by different methods of management. (The open issues are right-censored data.) Second, our problem has variables that naturally change over time. In a time-independent Cox regression, the independent variables do not change during the span of the observations. However, in our context, the resolution of an issue needs a series of actions under certain interaction patterns. Each action and each interaction pattern only lasts for a while and exerts some influence on the finishing of the issue. If we use a variable to capture the appearance of a certain type of actions, it will change over time, which is time-variant.
The standard setting of the time-dependent Cox regression model is:
\[ h(t) = h_0(t) e^{\beta_i X_i + \beta_j X_j(t)} \], where \( X_i \) indicates the time-invariant covariates and \( X_j(t) \) indicates the time-variant covariates. The major effect we want to capture is a time-variant variable on interaction patterns. Noticing that each interaction pattern may not only affect the issue resolution during its appearance time period but also influence all follow up actions in a subtle way, we create an “occurrence” variable to capture the effect of the pattern.

![Figure 19. The Effect of a Pattern on the “Hazard”](image)

This variable captures the first appearance of a pattern in an issue. Before the pattern’s first appearance, the variable is 0. After that, the variable is 1. Thus, this pattern occurrence variable is one during the time span when the pattern may have an effect. So the hazard associated with this variable can be represented as being \( h_0(t) \) and \( h_0(t) e^{\gamma} \) before and after the pattern’s occurrence. If we draw a hazard curve (assuming an...
arbitrary baseline hazard curve), we get Figure 19. Such types of variables have been used in previous studies (Morita et al., 1993). (Please note that a higher hazard means the issue may be addressed faster, which is what a project manager prefers.) In this research, we also group similar patterns together and measure their occurrence’s impact.

We include several control variables in addition to the variables on interaction patterns. First, we notice the different nature of issues and employ user-generated issue priority and issue type as two time-invariant control variables. Priority of the issue is an ordinal variable that can be directly included as a covariate. Issue type is a nominal variable with about 40 values. Thus, we stratified the model based on this variable. The stratified model assumes different types of issues have different baseline hazard functions but the covariates have simple impact on all types of issues. Second, we capture the basic characteristics of collaboration in a set of time-invariant control variables. Specifically, we capture whether the issue has an assignee, and whether the issue resolution involves collaboration (i.e., with more than one person). These variables provide us the basic idea of the impact of assignees and the appearance of collaboration on process efficiency. The third type of variables is time-dependent, which captures the impact of different interactions (assuming the impact is not related to the interaction pattern these actors belong to). Specially, we look at the appearance of comments and changes made by assignees. The hazard corresponding to these variables thus switches between \( h_b(t) \) and \( h_b(t)e^{\gamma} \) over time. If some actors appear multiple times, their cumulative effect will
increase the overall hazard for the issue to finish. Eventually, the model we used in this study is:

\[
h_g(t) = h_{0g}(t)\exp[\beta_1 \times \text{Priority} + \beta_2 \times \text{hasAssignee} + \beta_3 \times \text{isCollaboration} + \beta_4 \times \text{isComment}(t) + \beta_5 \times \text{isAssigneeAction}(t) + \sum \gamma_i \gamma_i P(t)]
\]

(1)

where \(g\) is the stratification on issue type, and variables with \(t\) indicate the time-variant covariates and \(P\) indicates the interaction pattern (group) covariates.

One concern of this time-dependent Cox regression model is that some issues may be more complicated, need more action, cost more time, and have a higher chance to introduce complicated interaction patterns. To address the concern, we stratify the model with the complicity of issues. We take the number of change items, which counts the multiple change items for all change actions during the issue resolution span to capture complicity of issues. Since each change item reflects the developers’ efforts from some perspectives, the number of change items shows the amount of effort spent on concrete changes. After stratification, the time-dependent Cox regression can provide us accurate assessments on how much each pattern may change process efficiency even on issues that requires similar efforts to finish. The model is:

\[
h_{g,a}(t) = h_{0g,a}(t)\exp[\beta_1 \times \text{Priority} + \beta_2 \times \text{hasAssignee} + \beta_3 \times \text{isCollaboration} + \beta_4 \times \text{isComment}(t) + \beta_5 \times \text{isAssigneeAction}(t) + \sum \gamma_i \gamma_i P(t)]
\]

(2)

where \(a\) is number of change items.
3.7 Results

3.7.1 Quantitative Analysis

We apply our pattern mining approach to the dataset described in Section 3.5. In this process, we investigate interaction patterns composed of different roles of individuals. In the dataset, most issues have an individual named as “assignee” who has the major responsibilities to coordinate efforts in addressing the issue. The “assignee” is the major developer or team leader in many cases. We project s/he would put more effort into the issue than regular developers or team members. We call other developers “workers.” Therefore, our extracted patterns are transition structures between nodes that have one of the two roles.

In the dataset, after setting alpha and beta to 10%, we identified 16 unique frequent interaction patterns. Table 10 reports the descriptive statistics of the data. In our dataset, about 85% of issues are resolved issues. Each issue on average involves about 3.25 individuals with about 10 actions. In general, 4 of these actions are comments, which show up in 84% of the issues. Six of the actions are changes, which are composed of 10 change items, on average. In addition, 80% of the issues have assignees and 89% of the issues involve more than one person (i.e., has collaboration among participants). The Table also summarizes the appearance of top patterns in the issues. Taking pattern 1 as an example, this pattern uniquely appears in 1,708 issues. In these issues, on average, it appears 1.22 times and each appearance occupies 4.06 consecutive actions (since one individual may conduct a series of actions before handing the task to another person, the
actions for each pattern are different across issues). As we can see, most patterns appear only once in most issues. Each appearance may occupy 4 to 8 actions. This simple characteristic partially alleviates the concern that the pattern occurrence variables may capture multiple patterns’ effect in one coefficient. In most cases, each occurrence variable series only matches to one pattern.

### Table 9. Summary Statistics

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>mean</th>
<th>stddev</th>
<th></th>
<th>n</th>
<th>mean</th>
<th>stddev</th>
<th>length</th>
<th>n</th>
<th>mean</th>
<th>stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolved</td>
<td>7991</td>
<td>0.850</td>
<td>0.360</td>
<td>pid1</td>
<td>1708</td>
<td>1.22</td>
<td>0.69</td>
<td>4.06</td>
<td>2.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Priority</td>
<td>7991</td>
<td>2.710</td>
<td>0.860</td>
<td>pid2</td>
<td>1340</td>
<td>1.24</td>
<td>0.72</td>
<td>4.13</td>
<td>2.95</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoOfPeople</td>
<td>7991</td>
<td>3.250</td>
<td>1.540</td>
<td>pid3</td>
<td>1319</td>
<td>1.1</td>
<td>0.35</td>
<td>6.98</td>
<td>2.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoOfActions</td>
<td>7991</td>
<td>10.290</td>
<td>8.740</td>
<td>pid4</td>
<td>1123</td>
<td>1.07</td>
<td>0.27</td>
<td>7.04</td>
<td>2.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoOfComments</td>
<td>7991</td>
<td>3.980</td>
<td>5.170</td>
<td>pid5</td>
<td>1088</td>
<td>1.05</td>
<td>0.22</td>
<td>6.5</td>
<td>3.13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HaveComments</td>
<td>7991</td>
<td>0.840</td>
<td>0.360</td>
<td>pid6</td>
<td>1045</td>
<td>1.17</td>
<td>0.46</td>
<td>7.33</td>
<td>2.68</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoOfChangeActions</td>
<td>7991</td>
<td>6.310</td>
<td>4.830</td>
<td>pid7</td>
<td>1031</td>
<td>1.3</td>
<td>0.72</td>
<td>8.59</td>
<td>5.17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoOfChangeItems</td>
<td>7991</td>
<td>9.980</td>
<td>7.720</td>
<td>pid8</td>
<td>1024</td>
<td>1.11</td>
<td>0.36</td>
<td>8.59</td>
<td>5.12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>HaveAssignee</td>
<td>7991</td>
<td>0.800</td>
<td>0.400</td>
<td>pid9</td>
<td>994</td>
<td>1</td>
<td>0.03</td>
<td>4.18</td>
<td>1.94</td>
<td></td>
<td></td>
</tr>
<tr>
<td>is_collaboration</td>
<td>7991</td>
<td>0.890</td>
<td>0.310</td>
<td>pid10</td>
<td>959</td>
<td>1.72</td>
<td>1.3</td>
<td>6.81</td>
<td>2.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Figure 20, we draw the distribution of patterns. The horizontal axis represents the length of process instances and the vertical axis represents the number of pattern occurrences. We further cluster these patterns into three groups according to the distribution of their occurrences: 1) P1, P2, P9; 2) P5, P13; 3) the others. The three groups of patterns have the most occurrences in processes that have 5, 8, and 11 actions, respectively. The distribution of patterns allows us to differentiate complexities of
different patterns. The pattern complexity is related to two aspects: number of nodes and number of edges. The number of nodes in a pattern graph represents how many people are involved, and the number of edges represents how many types of interactions occur in the pattern. Thus, the more nodes and edges a pattern has, the more complicated the pattern is.

**Figure 20. The Distribution of Patterns**

We visualize all identified patterns in Figure 21. The three groups of patterns are consistent with their pattern’s complexity. The more complicated a pattern is, the more likely that it will occur in longer processes. Patterns in Group 1 (P1, P2, P9) have the least complexity among all identified patterns because they only have two nodes and one edge. Group 1 is a collection of patterns that only involve single-direction collaboration between two people. This is the simplest interaction process. Patterns in Group 2 (P5,
P13) have the middle-level complexity among all identified patterns. Pattern 5 only has two people, but it is more complicated than patterns in Group 1 because it is a bi-directional interaction. Pattern 13 has three people, but it is less complicated than patterns in Group 3 because it has the simplest type of interactions (sequential process among three people). Patterns in Group 3 are the most complicated because they involve either more people or more interactions than patterns in Groups 1 and 2. This makes sense because patterns that are more complicated are more likely to occur in longer processes.

The pattern distribution is also consistent with patterns’ average lengths. The average lengths of patterns in Group 1, 2, and 3 are about 4, 6, and 8. The lower average length a pattern has, the more likely that it will occur in short processes. This is because lengths of patterns are closely related to pattern complexity. The minimum length of a pattern must be longer than the number of its edges, which is an important measurement of pattern complexity.

We also organize patterns according to their structural semantics in Figure 21. The major differences between these patterns are on two dimensions, i.e., whether the patterns contain a manager and whether the patterns contain a loop. As we know, the simplest patterns in business process and collaboration are sequential, which indicates one individual completely finish his/her task before handing it to others. In the patterns we extracted, there are relative shorter ones (with 2 or 3 workers) and longer ones (with more than 3 workers). (We make the distinction between 3 and 4 since the average
number of people in an issue is 3.25.) This applies to both patterns with and without managers.

For patterns with managers, we further differentiate whether the manager is participating at the beginning or at the end. If the manager appears at the beginning, he is likely to play a planning role. Otherwise, he is likely to play an evaluation and quality control role. For the patterns with a loop, we notice that all worker-based patterns are 2-worker loops plus a pre or post actor (if any). For loop patterns with managers, one major type is a manager-worker loop with a pre or post actor. Another one is a three-actor loop including a manager. After inspecting pattern structures, we identified 9 types of patterns.

Figure 21. Extracted Patterns in Software Issue Resolution
(some types have only one pattern). We thus create 9 occurrence variables to represent these patterns in the time-dependent Cox regression.

Table 11 reports the regression results of our model. In general, the effect of the control variables is consistent with the previous studies (Mockus et al., 2002). First, we notice that issues with a high priority (i.e., a low priority index) are completed faster. This meets our intuition. If the issue has an assignee it also is finished faster. In general, issues with collaboration take more time to complete. That may be because the tasks involving collaboration are more difficult to address. In terms of the effect of individual actions, we notice that both comments and assignee actions have a negative coefficient. The negative coefficient of comments means comments may slow down issue resolution (or complicated issues need more discussion). However, the effect of assignee action needs to be interpreted with the hasAssignee variable. For issues with an assignee, the effect of assignee action is to slow down issue resolution. As compared with issues without an assignee, an assignee action still increases hazard $e^{(1.170-0.373)}$ times than other actions (using model 1 as an example).

The results on interaction patterns are also interesting. First, worker-only sequential patterns are generally not very effective. Long_Worker_Seq shows negative coefficients on all models. In most of the cases, the coefficient of Short_Worker_Seq is not significantly different from 0. If the sequential patterns involved managers, the results are quite different. We found that Long_PreManage_Seq and Short_PostManage_Seq have positive coefficients on all models, and Long_PostManage_Seq is efficient if we consider
the different effort put into each issue. For operations with a larger number of workers, a manager should be involved early and do enough planning. For operations needing a small number of workers, managers can evaluate and summarize at the end; this can also accelerate the process. Managers can also conduct evaluation and summarization at the end of long operations, although its effect is not as significant. For patterns that involve iterative interactions among actors, we notice that patterns with managers in the iterations will make the process more efficient. However, worker-only loops in general reduce the process efficiency.
## Table 10. Regression Results of Software Issue Resolution Processes

<table>
<thead>
<tr>
<th></th>
<th>No Stratification</th>
<th>Stratified on # ChangeItems</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Priority</strong></td>
<td>-0.3493***</td>
<td>-0.3672***</td>
</tr>
<tr>
<td></td>
<td>(0.0232)</td>
<td>(0.0256)</td>
</tr>
<tr>
<td><strong>HaveAssignee</strong></td>
<td>1.1703***</td>
<td>1.3651***</td>
</tr>
<tr>
<td></td>
<td>(0.0505)</td>
<td>(0.0579)</td>
</tr>
<tr>
<td><strong>is_collaboration</strong></td>
<td>-0.7321***</td>
<td>-0.2787***</td>
</tr>
<tr>
<td></td>
<td>(0.0661)</td>
<td>(0.0848)</td>
</tr>
<tr>
<td><strong>is_comment</strong></td>
<td>-1.6935***</td>
<td>-1.8602***</td>
</tr>
<tr>
<td></td>
<td>(0.0663)</td>
<td>(0.0698)</td>
</tr>
<tr>
<td><strong>is_AssigneeAction</strong></td>
<td>-0.3734***</td>
<td>-0.5211***</td>
</tr>
<tr>
<td></td>
<td>(0.0529)</td>
<td>(0.0597)</td>
</tr>
<tr>
<td><strong>Short_Worker_Seq</strong></td>
<td>0.0963**</td>
<td>-0.0994</td>
</tr>
<tr>
<td></td>
<td>(0.0394)</td>
<td>(0.0447)</td>
</tr>
<tr>
<td><strong>Long_Worker_Seq</strong></td>
<td>-0.6089***</td>
<td>-0.2992***</td>
</tr>
<tr>
<td></td>
<td>(0.0535)</td>
<td>(0.0624)</td>
</tr>
<tr>
<td><strong>Short_PreManage_Seq</strong></td>
<td>0.0430</td>
<td>-0.3171***</td>
</tr>
<tr>
<td></td>
<td>(0.0499)</td>
<td>(0.0553)</td>
</tr>
<tr>
<td><strong>Long_PreManage_Seq</strong></td>
<td>0.1744***</td>
<td>0.2853***</td>
</tr>
<tr>
<td></td>
<td>(0.0514)</td>
<td>(0.0603)</td>
</tr>
<tr>
<td><strong>Short_PostManage_Seq</strong></td>
<td>0.1462***</td>
<td>0.4801***</td>
</tr>
<tr>
<td></td>
<td>(0.0525)</td>
<td>(0.0580)</td>
</tr>
<tr>
<td><strong>Long_PostManage_Seq</strong></td>
<td>-0.3508***</td>
<td>0.1282**</td>
</tr>
<tr>
<td></td>
<td>(0.0408)</td>
<td>(0.0498)</td>
</tr>
<tr>
<td><strong>Two_Worker_Loop</strong></td>
<td>-0.6105***</td>
<td>-0.0497</td>
</tr>
<tr>
<td></td>
<td>(0.0395)</td>
<td>(0.0479)</td>
</tr>
<tr>
<td><strong>Manager_Worker_Loop</strong></td>
<td>0.0828**</td>
<td>0.4314***</td>
</tr>
<tr>
<td></td>
<td>(0.0377)</td>
<td>(0.0447)</td>
</tr>
<tr>
<td><strong>Manager_MultiWorker_Loop</strong></td>
<td>0.1891***</td>
<td>0.3522***</td>
</tr>
<tr>
<td></td>
<td>(0.0493)</td>
<td>(0.0574)</td>
</tr>
<tr>
<td><strong>Concordance</strong></td>
<td>0.7010</td>
<td>0.7120</td>
</tr>
<tr>
<td></td>
<td>0.7200</td>
<td>0.7230</td>
</tr>
<tr>
<td></td>
<td>0.7340</td>
<td>0.7300</td>
</tr>
<tr>
<td><strong>Rsquare</strong></td>
<td>0.0300</td>
<td>0.0300</td>
</tr>
<tr>
<td></td>
<td>0.0340</td>
<td>0.0320</td>
</tr>
<tr>
<td></td>
<td>0.0380</td>
<td>0.0340</td>
</tr>
<tr>
<td><strong>Likelihood ratio test</strong></td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td><strong>Wald test</strong></td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td><strong>Score (logrank) test</strong></td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td></td>
<td>0.0000</td>
<td>0.0000</td>
</tr>
</tbody>
</table>
3.7.2 Qualitative Analysis

We conducted a case study (Yin, 2003) to investigate why patterns have different effects on process efficiency. The goal of the case study is to validate the findings of pattern efficiency and provide a rich and contextualized explanation of pattern efficiency in software issue resolution. In this study, we focus on three patterns: Long_Worker_Seq, Long_PreManage_Seq, and Short_PostManage_Seq. More case studies and interviews will be conducted in our future work to explore the efficiency of other patterns. We study these three patterns because (1) their effects are consistent in all models, (2) both positive and negative effects are included in the three patterns, and (3) their structures do not contain loops, which is easier for case analysis.

Cases were selected based on the following criteria: 1) They have priority level 1 (we used 1 as the priority level since it gives us the most cases). 2) A process only contains one type of pattern; 3) They have an assignee. Since whether a process has an assignee and its priority level are significantly correlated to their process time, we use the case select criteria to rule out possible differences that may be caused by having an assignee or priority level. We have found 28 cases from our dataset based on the three criteria and studied them all. Table 12 shows the summary of selected cases. From Table 12, we can confirm that processes that only include the Long_Worker_Seq pattern are less efficient than processes that include Short_PostManage_Seq (P2) and Long_PreManage_Seq (P12). It can be justified from three aspects: the whole process time, average time for each action, and average time for each change.
Table 11. Case Summary

<table>
<thead>
<tr>
<th>Patterns</th>
<th>No. of cases selected</th>
<th>Average No. of actions</th>
<th>Average No. of change items</th>
<th>Average duration of process (hours)</th>
<th>Average time for an action (hours)</th>
<th>Average time for a change item (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short_PostManage_Seq (P2)</td>
<td>10</td>
<td>5.2</td>
<td>5.7</td>
<td>195</td>
<td>37.5</td>
<td>34.2</td>
</tr>
<tr>
<td>Long_Worker_Seq (P10)</td>
<td>8</td>
<td>7.8</td>
<td>11.8</td>
<td>1129</td>
<td>144.7</td>
<td>95.7</td>
</tr>
<tr>
<td>Long_PreManage_Seq (P12)</td>
<td>10</td>
<td>6.6</td>
<td>7.1</td>
<td>488</td>
<td>73.9</td>
<td>68.7</td>
</tr>
</tbody>
</table>

Next we read the actor logs of all selected cases. We then summarize the reasons that may cause Long_PreManage_Seq and Short_PostManage_Seq to be more efficient than the Long_Worker_Seq pattern. The major difference between the Long_Worker_Seq pattern and the other two patterns is that managers do not perform any actions in the Long_Worker_Seq pattern. It is critical to use managerial control to enhance the effectiveness of virtual teams, due to their limited life span and cross-functional or cross-organizational membership (Kayworth and Leidner, 2001). Otherwise, virtual teams may suffer from many problems that lead to low team efficiency (Kirkman et al., 2002). We identified three reasons that may lead to differences in pattern efficiency.

First, did managers set clear goals for team members? Goal clarity and lack of goal conflict are found to be related to team effectiveness (Hertel et al., 2005). In virtual teams, setting up explicit goals is extremely important because autonomic teams have greater levels of team self-direction that may lead to increased confusion, decreased responsibility, and a lack of accountability. In the cases we observed, managers often assigned well-defined tasks to team members and gave detailed guidelines. For instance,
the following two examples show how managers set up goals and assign tasks, where participants’ names are replaced by “XXX” for privacy issues.

“Hi XXX, I'm assigning this to you, so you can determine if it should be moved to the Labor Module, or if instructions need to be clarified, etc. Let me know if you have any questions. Thanks, XXX”

“XXX, I broke some of the accounting period stuff with my recent code changes, so I'm assigning this to you directly, in hopes that you can jump on this as soon as you can. Thanks.”

In the processes that contain the Long_Worker_Seq pattern, members sometimes are confused about what to do and who is responsible for what. For example, we observed a comment from one worker in a case.

“sorry XXX - i know this violates what we discussed today, but given your involvement with the issue that lead to this, i think it's best this go to you.”

Second, did managers provide developmental feedback to workers? Developmental feedback has two unique characteristics: (1) it provides high-quality (i.e., helpful) information and (2) it is future oriented, as feedback recipients are directed toward making improvements on the job (Zhou, 2003). Therefore, developmental feedback is different from traditional performance feedback in that it focuses on improvement and is likely to enhance motivation. As a result, managers’ developmental feedback is positively related to performance (Li et al., 2011). The following case shows an example of developmental feedback for managers.
“This is XXX’s email about how to interpret this attachment, and this is very useful for the non-simple main docs. Basically you'll need to read the code and figure out which of a document's properties these attributes are referring to, and specify tho...”

An opposite situation in the Long_Worker_Seq pattern is shown in the following case where a worker says he cannot fix the problem but nobody has ever responded to him/her. In this case, the issue resolution was greatly delayed because of insufficient information.

“As XXX explained it to me, this change will require work in the KNS/the use of the KNS for document searches by XXX. We can't fix this one ourselves.”

Third, did managers express their appreciation explicitly and frequently? Physical disconnectedness in virtual teams can lead to various challenges to members' work motivation. One of the most important challenges is that self-efficacy is more difficult to maintain due to reduced feedback (Kirkman et al., 2002). Moreover, team members’ satisfaction is often very low because of reduced feedback. Managers’ confirmative attitude and thankful expressions may help maintain workers’ self-efficacy. In processes with patterns 2 and 12, we found 12 comments with “thanks” or “thank you” in 20 cases. But we did not find one “thank you” comments in 8 processes with the Long_Worker_Seq pattern.
These reasons explain why patterns have different effects on efficiency. In the future, we will study the reasons for all identified patterns through more case studies and interviews with developers.

3.8 Conclusion

3.8.1 Implications

This study makes three major contributions. First, we proposed an analytical framework to study the effect of interaction patterns. The pattern analysis framework contains two steps: pattern mining and pattern impact assessment. Second, we identified 16 interaction patterns in issue resolution processes, which can be used as primary blocks for process analysis. Third, most identified patterns have significant correlations with work efficiency, which may change from positive to negative. The rationale and principles behind the correlation between interaction patterns and software development efficiency is worthy of further investigation. As such, our research opens new opportunities in collaboration research.

A salient finding from our study is the significant impact of interaction patterns on process efficiency in software issue resolution. It helps answer the question of why interaction processes in software issue resolution vary in efficiency. It points to a new direction toward understanding the factors that affect interaction process efficiency: interaction process structures. Current software project management mainly focuses on how to enhance collaboration efficiency by improving social, cultural, organizational, and technical support provided to teams (Faraj and Sproull, 2000). We show that process
support is also an important perspective for collaboration management in software projects. For the same group of people with the same collaboration environment, different interaction process structures may lead to performance differences. Considering all these regression results, we have the following suggestions for software engineering collaboration:

1. Assign an assignee for each issue.
2. Encourage manager’s involvement in longer operations.
3. If the task is complicated and needs iterative operations, make the assignee one chain of the loop.
4. If the task is long, even if it is sequential, managers’ early planning and final evaluation is helpful.
5. Even if the task is simple, managers should conduct final evaluations.

From the process management perspective, traditional workflow management techniques mostly deal with structured processes that can be predefined (Stohr and Zhao, 2001). Dynamic and ad hoc interaction processes are usually treated as a black box, and they are not supported well by workflow management techniques (Thomson and Perry, 2006). Currently, interaction processes are normally managed in an ad hoc way because most collaboration activities are unpredictable and unstructured. Our study shows that although each interaction process instance is ad hoc and seldom repeated, they do contain recurrent process patterns. Our approach can be considered as a first step to open the
black box of ad hoc interaction processes because we identified the recurrent portions of interaction processes that can be supported by process management techniques.

From the perspective of identifying interaction patterns in software development, we adopt a quantitative approach based on data mining techniques and extract patterns from real-world working logs, while existing literature is mostly based on qualitative analysis such as case studies and interviews (Aranda and Venolia, 2009). The two approaches are complementary to each other because the quantitative approach is objective but may miss the contextual explanation while the qualitative approach provides a good understanding of the meaning of each pattern but tends to be subjective. An integrated approach that combines both qualitative and quantitative approaches in pattern mining may provide insightful understandings of interaction processes.

Our findings may also have practical implications for designing better collaboration management tools for software issue tracking. For the purpose of interaction process management, awareness is one of the most important functions that a collaboration system can provide to teams (Dourish and Bellotti, 1992). However, it is usually difficult to figure out how to provide the right level of awareness in virtual teams with interconnected relationships among team members (Cataldo et al., 2006). When managers or team members are aware of their interaction process and the current process pattern, they can try to adjust their interaction process and use efficient process patterns when possible. Therefore, tools that detect process patterns and allow team members to be aware of the interaction process structure should help interaction process management.
These kinds of tools can help expand groups’ “Intellectual Bandwidth” by enhancing their ability to create, sustain, and then change their patterns of collaboration (Nunamaker et al., 2000).

Our contributions extend beyond the issue resolution process in open source software development and have implications for virtual teams in other contexts. Our study indicates the existence of interaction patterns and the importance of managers in such a typical virtual team. Since interaction processes in virtual teams are often autonomic, identifying and encouraging “efficient process patterns” is likely to be a critical success factor for the performance of any virtual team. Our research points out how to identify “efficient process patterns”: pattern mining and pattern assessment. There is potential to extend this finding to virtual communities in other contexts by replicating the approach proposed in this chapter. Our findings regarding manager-worker interactions can also inform research on leadership style in virtual teams. The effect of leadership style has been considered a critical factor for the success of a team (Morgeson et al., 2010). The existing definitions of leadership styles are mainly based on qualitative descriptions such as democratic, autocratic, or free reign. Our research suggests a formal method of describing the process of managers’ activities to study the effect of leadership styles.

3.8.2 Limitations and Future Work

This study has two main limitations.
First, the interaction patterns are derived from data collected from an issue tracking system in an open source community. We do not have a record of collaboration activities that are outside of the issue tracking system. Therefore, the interaction patterns in this chapter do not consider the activities that are not recorded by Jira. However, the data was collected from a community that requires their members to use Jira as the platform to track issue resolution. Further, in a qualitative study of issue tracking systems as used by small, collocated software development teams, researchers found that even in collocated teams, issue trackers are a focal point for communication and coordination (Bertram et al., 2010). Moreover, collaboration systems are much more important in large virtual teams than in collocated teams. Thus, for the large, distributed teams that we are concerned with in this chapter, interaction activities outside Jira are only a minor threat.

Second, since our data came exclusively from one open source community, the extent to which our results are valid for other collaboration contexts is not clear without replication. Drawing general conclusions from empirical studies in collaboration is difficult because any process depends on a potentially large number of relevant context variables, such as group characteristics, organizational support, culture, and so on. For this reason, we cannot assume that the results of our study generalize beyond the specific environment in which it was conducted. The community we studied in this chapter has thousands of developers, managers, and users. The data we collected involved more than
20 projects and 400 software developers and managers. Nevertheless, for different collaboration contexts, replications of this study would help resolve this question.

This research can be extended in several directions. First, we will consider different types of actions in issue resolution and incorporate them into the procedure of interaction pattern mining. Second, we will apply the pattern analysis framework proposed in this chapter to other collaboration contexts and generalize our findings on interaction patterns. Third, we will design and implement an issue tracking system that can support interaction process management via interaction pattern monitoring and recommendations.
CHAPTER 4. A THEORY OF COLLABORATION VIRTUALIZATION

4.1 Introduction

Collaboration is broadly defined as a process in which more than two people work together to achieve a common goal. Recent years have witnessed a trend of collaboration virtualization: geographically or temporally dispersed people collaborate with each other via collaboration technologies or other virtual means. Group-based collaboration and technologies to support a broad range of interaction have proliferated and are increasingly a central component in organizations (Smith and McKeen, 2011). In a review of the literature on virtual collaboration, the authors conclude that “with rare exceptions all organizational teams are virtual to some extent” (Martins et al., 2004).

The move towards collaboration virtualization is faster than ever before. However, some tasks are more successful with virtual collaboration while others are more successful with physical collaboration. For example, with the support of advanced technology some virtual teams achieve improved collaboration performance (May and Carter, 2001). However, when facing great management challenges, physical collaboration is better than virtual collaboration (Kirkman et al., 2002). Similarly, when compared with physical group decision making, virtual group decision making could either be more or less effective (Dennis et al., 2001). Although virtual collaboration is and will continue to be an important and necessary type of work arrangement, it is not appropriate for all circumstances (Nemiro, 2002). It is not clear what precisely the
determinants of effective virtualization are. These observations lead to the following research questions: What factors of collaboration affect suitability of virtualization? And how do those factors affect the design of effective collaboration systems?

This chapter addresses these questions by proposing CVT. As a theory, it integrates and builds upon prior academic research to propose specific constructs, relationships among those constructs, and propositions. The theory contains three categories (e.g., task, technology, and team) of constructs that determine the suitability of collaboration virtualization. Moreover, it discusses how multi-task degree is related to virtual collaboration management, a novel yet salient factor that has not received much research attention from virtual team scholars.

The main contribution of this chapter is to extend previous studies on virtual collaboration by providing new theoretical insights on the suitability of virtual collaboration. The results of our investigation will help collaboration managers better understand the requirements of virtual collaboration management in different contexts. The rest of this chapter is organized as follows. Section 4.2 reviews the relevant literature. A conceptual model and propositions are then presented in the Section 4.3. Finally, Section 4.4 concludes with expected findings and implications of this study and future research directions.

4.2 Literature Review

Collaboration involves multiple individuals who combine their efforts to achieve mutually desired states or outcomes. Collaboration is defined as joint effort towards a
group goal (Briggs et al., 2003). There is no doubt that information and communication technologies are enabling different ways of working. IT is one of the key components for successful collaboration (Smith and McKeen, 2011). They further elaborate four fundamental building blocks of collaboration IT: communication, information access and management, security and risk, and technology integration. Literature has shown that IT is a significant factor in facilitating the success of collaboration in organizations (Zammuto et al., 2007).

IS literature suggests that collaboration efficiency is significantly affected by the media selected for collaboration. Selecting the right collaboration tool is essential for a high level of collaboration performance. Given the wide range of tool options from email to instant messenger, theories adopting this perspective try to establish a set of principles which would guide users to select the most appropriate tool for facilitating collaboration. Media richness theory (Daft and Lengel, 1986) and task-technology fit theory (Zigurs and Buckland, 1998) state that the medium used for team communication should fit the type of information needed for the task. Also, media synchronicity (Dennis and Valacich, 1999) establishes a connection between the task and the way the information is exchanged.

Process virtualization theory is designed to explain why some processes are more suitable to being conducted virtually than others. There are four constructs in process virtualization theory that describe process characteristics: sensory requirements, relationship requirements, synchronism requirements, and identification and control
requirements. According to process virtualization theory, if a process requires more human sensory experience, social context, time control, and identity control, it will be less amendable for virtualization. The moderating effect of representation, reach, and monitoring capability are also discussed in this theory.

Most existing theories (e.g., task-technology fit theory, media richness theory, and media synchronicity theory) can be used to explain the relationship between IT and collaboration performance. However, performance is not always the only concern for collaboration management. For example, when the output of a task is highly sensitive, physical collaboration is preferred even when the virtual collaboration performance is better (Hunsaker and Hunsaker, 2008). It is difficult to determine whether if a collaboration task is suitable to virtualization. Process virtualization theory can partially explain whether a process is suitable for virtualization. However, collaboration is a special type of process that includes communication, coordination, and cooperation. Group, task, and technology characteristics should be considered when making the decision of collaboration virtualization. Therefore, we need a new theory to explain the virtualizability of collaboration. This is an imperative due to the increasing pervasiveness of virtual collaboration teamwork in modern organizations today.

4.3 Collaboration Virtualization Theory

4.3.1 Definitions and Overall Conceptual Model

Collaboration is defined as a process where two or more people work together to achieve a common goal. First, we define some terms that are important for CVT.
• **Virtual collaboration** is a collaboration process where participants interact “virtually” (via IT-enabled channels) to achieve a goal. Most researchers define it in terms of dispersion on various dimensions, at a minimum across time or space.

• **Physical collaboration** is a collaboration process in which participants work face-to-face to achieve a goal.

• **Collaboration virtualizability** is the suitability for virtual collaboration.

• The transition from physical collaboration to virtual collaboration is defined as **collaboration virtualization**.

Using a meeting as an example, team members can either have a virtual meeting through a web meeting system or have a physical meeting by gathering all team members in a conference room. Table 13 compares the differences between fully virtual collaboration and fully physical collaboration.

<table>
<thead>
<tr>
<th>Table 12. Differences between Virtual and Physical Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fully Virtual Collaboration</strong></td>
</tr>
<tr>
<td>Team members are all located in different locations.</td>
</tr>
<tr>
<td>Team members communicate through virtual means.</td>
</tr>
<tr>
<td>Team members may communicate asynchronously.</td>
</tr>
<tr>
<td>Team members may devote part of their attention to collaboration.</td>
</tr>
</tbody>
</table>

Our literature review yielded surprisingly little in terms of a systematic, theoretical discussion of the factors of virtualizability of collaboration. For example,
employee training and new product development tasks are conducted physically in some companies but virtually in others. In virtual collaboration, teams employ certain technologies to collaborate on a project. This implies that traditional collaboration research, conducted in a physical environment, performing contrived tasks, will not be particularly applicable. We reviewed the virtual collaboration literature and developed three theoretical categories of the factors that may have important effects on collaboration virtualizability: team, task, and technology characteristics (Kirkman et al., 2002; Smith and McKeen, 2011; Straub and Karahanna, 1998). Then, we developed a model of the three categories of constructs that are likely to lead to lower or higher levels of virtualizability. Figure 22 describes the conceptual model of CVT.

Every collaboration process is virtual to a certain degree (Griffith et al., 2003; Martins et al., 2004). Thus, the overall collaboration process is neither purely physical nor purely virtual, but a hybrid of the two extremes. The distinction between physical and virtual collaboration is more continuous than discrete. Most collaboration may contain both physical and virtual activities. CVT can be applied to each activity of the collaboration process. For example, a new product development project may have both face-to-face and remote meetings. The face-to-face meeting is considered as physical collaboration and other forms of remote meetings are virtual collaboration. The suitability of a virtual meeting is determined by task, technology, and team.
4.3.2 Task Characteristics

Not all tasks are equally suitable for virtual collaboration. The nature of collaboration tasks is important for collaboration management. For example, researchers have found that physical collaboration (face-to-face) is preferred for equivocal tasks (Daft and Lengel, 1986). Moreover, security and physical constraints of tasks also need to be considered when collaborating virtually (Hunsaker and Hunsaker, 2008). Based on the literature, we derived three characteristics of tasks as important indicators of collaboration virtualizability.
Task Urgency is defined as the degree of temporal constraints posted on the task. The logical reasoning behind this proposition is straightforward. For urgent tasks, collaboration needs to be done within a short time period. More urgent tasks incline people toward real-time, synchronous response communications. All things being equal, urgent tasks are predicted to be correlated with synchronous media such as face-to-face meetings and telephone (Straub and Karahanna, 1998). Previous researchers have also proven that it takes longer for teams to collaborate via virtual means than it does face-to-face (Baltes et al., 2002). Physical collaboration participants can interact with one another with little delay because they are all located in the same place. By contrast, virtual collaboration participants are located away from one another, which may introduce delays. Thus, urgent tasks require more synchronous collaboration management. Based on process virtualization theory (Overby, 2008), more synchronism requirements lead to lower virtualizability.

Proposition 1. The more urgent the task, the less virtualizable the collaboration is.

Complexity is a measure of the amount of mental and physical effort needed for achieving the goal (Rana et al., 1997). It is related to three aspects: the structural certainty, the information processing requirement, and interdependency. As teams perform more complex tasks, they are expected to be more likely to use synchronous communication media (Bell and Kozlowski, 2002). Further, knowledge is not well-structured in complex tasks (Rana et al., 1997). As a result, it is hard to represent and transmit knowledge through virtual means for complex task collaboration. Complex tasks are likely to be
better done with technologies of higher informational value and synchronous member interactions (Kirkman and Mathieu, 2005). Also, complex tasks require more precise forms of coordinated effort because team members’ roles become highly interdependent and the need for reciprocal communication is essential. For example, previous studies compared the performance of face-to-face groups on three tasks of different complexity to that of computer-mediated groups (Straus and McGrath, 1994). They found that face-to-face groups did perform significantly better than computer-mediated groups on a more complex task. Research has shown that synchronous communication is superior to asynchronous communication for complex tasks that require a great deal of information sharing and collaborative decision making. Therefore, complex tasks require support for highly synchronous communication and coordination and lead to low virtualizability.

Proposition 2. The more complex the task, the less virtualizable the collaboration is.

Sensitivity is defined as the need to protect task-related information from being disclosed to others who might have low or unknown trustworthiness or undesirable intentions. Due to many potential security risks in virtual teams and internet-based technology, virtual collaboration is susceptible to unauthorized information leaks (Lee, 2009). Participants cannot physically inspect others to confirm their identity in the virtual environment. Security threats in virtual collaboration can be classified into social (e.g., phishing attacks and social engineering) and technical perspectives (e.g., worms and viruses). The security of virtual environments and the integrity of virtual objects may also
be targeted by malwares and bots that scan for weaknesses. Virtual collaboration relies heavily on IT and social media and is vulnerable to both social and technical security threats (Hunsaker and Hunsaker, 2008). Therefore, sensitive tasks require more identity control, encryption, and mentoring mechanisms; thus they are less virtualizable.

*Proposition 3. The more sensitive the task, the less virtualizable the collaboration is.*

4.3.3 Technology Characteristics

IT plays an important role in virtual teams (Smith and McKeen, 2011). Virtual teams use IT to communicate and coordinate. Whether IT can provide sufficient support for virtual collaboration is an important concern of collaboration management. So it is important to consider available IT support when collaborating virtually. We derived two technology characteristics that may affect collaboration virtualizability.

*Capacity* is related to the functional options, communication bandwidth, and information richness provided by information technology. Advanced collaboration tools have encouraged organizations to assign tasks to groups that are distributed rather than co-located (Sengupta and Zhao, 1998). Prior research pointed out that IT is one of the key components for successful collaboration (Smith and McKeen, 2011). Collaboration technologies usually can be classified into different levels based on their functionalities, richness, and communication bandwidth. It is a reciprocal process in which team members share knowledge with each other and achieve shared understanding. During the
process, team members need to participate in different kinds of communication and coordination processes. As such, virtual collaboration cannot be carried out smoothly without IT support at all levels mentioned above. Thus more IT capacity will make it easier for virtual collaboration management.

**Proposition 4. The more capacity information technology can provide, the more virtualizable the collaboration is.**

*Accessibility* is defined as the degree of ease to access information technologies. Accessibility of a technology is influenced by infrastructural factors (power supply, hardware support, network availability, etc.) or software service and support (software quality, stability, etc.). Accessibility to a collaboration technology is a fundamental requirement for technology usage. Difficulty accessing the information system will hinder a potential user (Kling and Elliott, 1994). Further, better accessibility leads to more usage of an information system (Graham, 1995). Digital library researchers found the positive effect of accessibility on perceived ease of use. Media accessibility will increase IT usage in virtual teams (Park et al., 2009; Thong et al., 2002).

**Proposition 5. The more accessibility information technology can provide, the more virtualizable the collaboration is.**

4.3.4 Team Characteristics

IT function alone cannot make collaboration happen, even if it provides robust collaboration technologies. The nature of groups will also influence the success of virtual collaboration (Handy, 1995). For example, some groups are good at virtual collaboration
while others tend to resist because of team conventions or norms. We derived two team characteristics as predictors of collaboration virtualizability.

*Team Relationship* is defined as the degree to which team members are familiar with each other. Due to the absence of social cues in electronic media, developing interpersonal relations is very difficult in virtual collaboration (Yoo and Alavi, 2004). However, trust between team members has significant impact on the success of collaboration. According to (O'Hara-Devereaux and Johansen, 1994), "Trust is the glue of the global workspace, and technology doesn't do much to create relationship.” Researchers (Handy, 1995) points out that, in virtual organizations, trust requires constant face-to-face interaction-the very activity the virtual collaboration tries to avoid or reduce. Further, familiarity among team member allows them to know each other’s expertise and reduce knowledge barriers during virtual collaboration. Otherwise, activities that get people to know each other in virtual teams are very important for effective virtual collaboration (Nunamaker et al., 2009). Therefore, lack of relationship and trust reduces virtualizability of collaboration (Aubert and Kelsey, 2003).

**Proposition 6. The stronger the relationship among team members, the more virtualizable the collaboration is.**

*Team Experience* is defined as the degree to which the team is familiar with the task and technology. Task experience will reduce uncertainty, increase self-efficacy, and lead to better collaboration (Littlepage et al., 1997; Staples et al., 1999). Researchers have shown that familiarity with the task is positively related to the level of group success
(Goodman and Leyden, 1991). In addition, team experience with technology will let the team appropriate the technology and achieve better outcomes (Majchrzak et al., 2000).

**Proposition 7.** The more experience the team has, the more virtualizable the collaboration is.

4.3.5 Moderating Effect of Multi-task Degree

*Multi-task degree* is defined as the number of tasks assigned to team members at the same time. One of the main differences between physical and virtual collaboration is that members of virtual teams do not belong to only one organization or team and cannot pay continuous attention to the project. In virtual collaboration, participants usually work on multiple tasks at the same time and they devote a portion of their attention to each task. As such, it is appropriate to use an attention-based view to analyze virtual collaboration (Ocasio, 1997). A major challenge for virtual collaboration managers is their inability to physically observe their employees' participation and manage their attention. As such, attention management is critical for successful outcomes of virtual collaboration (Davenport and Beck, 2002). Collaboration requiring a high degree of multi-tasking will require much coordination and communication efforts among team members. When the degree of multi-tasking is high, collaboration technologies should provide additional functionalities for attention management. Further, when tasks are complicated, collaboration management could be even more complex. The detailed moderating effects
of multi-task degree need more investigation in our future work. Therefore, we have the following proposition.

Proposition 8. The degree of multi-tasking moderates the relationship between task, team, and technology characteristics and collaboration virtualization.

4.4 Applying Collaboration Virtualization Theory to an Empirical Setting

Software development is a collaborative process that requires collaboration among users, analysts and programmers. In different stages of software development, different people collaborate on different tasks with different people. For example, in requirement analysis, analysts collaborate with users to extract user requirements and document them for system design and implementation. In the stage of coding and testing, programmers write software code to implement the software based on system design. Figure 24 shows a typical software development process. In this section, we mainly focus on analyzing two tasks (i.e., requirement analysis and coding and testing) in the software development process. This example shows how CVT can be applied for strategic decision making on collaboration virtualization in organizations.
From the perspective of task characteristics, for requirement analysis, analysts need to interact with users and compose documents that describe user requirements. This task usually needs to be done within a short time period after the project starts. For the coding and testing task, it usually takes much longer time than requirement analysis. The claim that requirement analysis is usually more urgent than coding and testing is also supported by previous research in software engineering. In a survey of software projects, researchers found that requirement analysis on average takes 25% of the whole software development time while the task of coding and testing takes about 56% of the whole development time (Zhang and Pham, 2000). “The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements” (Brooks, 1987). Thus, it is generally accepted by the software engineers that requirement analysis is more complex than other part of software development, including coding and testing. Sensitivity is a
construct that is related to the specific features of business domain. So we don’t explicitly compare sensitivity among the two tasks in this example.

From the perspective of technology characteristics, software development teams may use many kinds of technologies in both tasks. For example, general communication media such as emails, messengers, and video chat are widely used in software development processes. In addition, tools supporting requirement analysis (joint application development) and coding and testing (e.g. version control tools) are also widely used in software projects (Scacchi, 2002). Thus, we consider similar technology capacity and accessibility in both tasks.

From the perspective of team characteristics, users and analysts collaborate with each other to analyze the requirements for system design and implementation. In most cases, users are not very familiar with analysts because they tend to work in different organizations and have no experience of collaboration with each other before (Newman and Robey, 1992). This is similar for the comparison on experience. While analysts might be experienced in requirement analysis, users are generally not very experienced on this task.

Table 13. Comparison between Requirement Analysis and Coding and Testing

<table>
<thead>
<tr>
<th></th>
<th>Requirement Analysis</th>
<th>Coding &amp; Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Urgency</td>
<td>High</td>
<td>Medium/Low</td>
</tr>
<tr>
<td>Complexity</td>
<td>High</td>
<td>Medium/Low</td>
</tr>
<tr>
<td>Sensitivity</td>
<td>Not comparable in this case</td>
<td>Not comparable in this case</td>
</tr>
<tr>
<td>Technology</td>
<td>Capability</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Medium/High</td>
<td>Medium/High</td>
</tr>
<tr>
<td>Accessibility</td>
<td>Medium/High</td>
<td>Medium/High</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Team</td>
<td>Relationship</td>
<td>Low</td>
</tr>
<tr>
<td>Experience</td>
<td>Medium/Low</td>
<td>Medium/High</td>
</tr>
</tbody>
</table>

By following the framework in CVT, we can basically conclude that collaboration on the coding and testing task is more suitable to be virtualized than the collaboration on the requirement analysis task. When software project managers makes the decision on which part of software development should be virtualized, coding and testing is more preferable than requirement analysis. The historical data on software development also support this conclusion. Software coding and testing has moved away from traditional collocated model (physical collaboration), to the virtual collaboration model in the past two decades (Kaiser and Hawk, 2004). But requirement analysis is mostly done physically at the client’s location (Zowghi, 2002). Initial requirement analysis phase of software projects normally gets executed at client locations to leverage frequent and deep interaction between user and developer teams. Until recently, the software industry started to explore the possibility of virtual requirement analysis, but the likely negative impact of software quality hindered collaboration virtualization in this task (Nath et al., 2008).
4.6 Discussions and Conclusions

4.6.1 Contributions

In this chapter, we proposed CVT to explain the suitability of virtual collaboration in organizations. CVT extends existing theoretical work on virtual team and collaboration management by trying to incorporate three categories of constructs to predict collaboration virtualizability. Most of the exiting literature on virtual teams studies the factors that affect team performance. We contribute to the virtual team literature by showing that performance is not the only concern for virtual collaboration. Dimensions of adoption and cost of virtual collaboration are also important measurements of suitability of virtual collaboration. The unique feature of virtual collaboration adoption is that it is not restricted to any particular technologies.

The theory will also guide practitioners to consider how IT might help satisfy the requirements of virtual collaboration. Based on CVT, technology capabilities and accessibilities are very important for collaboration virtualizability. We point to the importance of increasing collaboration virtualizability by enhancing technology capabilities and accessibilities. When companies are under the pressure of virtualizing their collaboration processes, one of the important things that they need to do is to increase technology capabilities and accessibilities in their companies.

CVT also provides an analytical framework for managers who are planning to migrate their collaboration processes from physical to virtual. The framework will help managers prioritize the progress of virtualization according to the suitability of different
collaboration tasks, teams and technologies. For example, managers can use the theory to assess the suitability of collaboration processes by considering their task, technology and team characteristics. Tasks with higher virtualizability are easier to gain success through virtual collaboration.

4.6.2 Limitations

We identify three major limitations of this chapter.

First, although CVT provides insights for managers to choose virtual or physical ways to collaborate, CVT cannot be directly used to assess whether a virtual collaboration process is better or worse than a physical collaboration process. It only explains the factors that may affect suitability of collaboration virtualization. To illustrate, we can claim that coding and testing is more suitable for virtual collaboration than requirement analysis. This is different from saying that virtual requirement analysis is worse than physical requirement analysis or vice versa.

Second, it does not address the issue of relative impact of each of these constructs in this model. At this stage, we suspect that this is likely to be affected by other factors such as collaboration environment, personality of participants etc. For example, task sensitivity in a risky environment is arguably more important than that in a safe environment. Considerations of this sort are outside the scope of the theory. Future empirical tests can be used for a particular environment and estimate the weighted impact of all factors.
Third, the moderating effect of multi-task degree is not fully explored. One of the major differences between virtual and physical teams is that team members in virtual teams are more flexible to switch among tasks. But too many tasks will increase the coordination efforts and cognitive load. This will bring a critical effect to virtual teams. For example, when people collaborate on urgent tasks, higher degree of multi-tasking may lead to more delay. It would be of theoretical and practical interest to understand how multi-task degree will affect the relation between task, technology and team characteristics and collaboration virtualizability.

4.6.3 Future Work

Our model would benefit from empirical testing to refute or validate the propositions that have been stated. The theoretical model could be tested through experiments. Specifically, future research can investigate how different types of tasks (varying in complexity, urgency, and sensitivity), technology support (single technology with limited access vs. multiple technologies with good accessibility) and team composition (experienced team vs. non-experienced team) will affect the success of virtual collaboration (in terms of adoption, efforts and performance). Variations of adoption, efforts, and performances can be measured at the end of collaboration processes. Any empirical test of the model would require researchers to develop conceptual definitions and operational measures for each construct in the theory. Empirical studies that have been done on media usage, virtual teams and group decision making will
provide a good starting point for developing operational measures of CVT constructs and for testing the theory.

The problem with experiment design is that we may have to use artificial tasks that can be finished in a short time period. Further, it is very likely that we have to use students as subjects. Multiple methods evaluation can provide us more comprehensive understandings on the model than single method evaluation does. Filed studies such as survey can be employed to test the model partially. We can test the model by measuring the human perceptions and their impacts on decision making related to virtual collaboration. We may conduct surveys to ask managers’ perceptions of CVT constructs on multiple collaboration cases. Perceived task urgency, complexity, etc can be measured with survey questions and their relationship with managers’ decision on virtual collaboration can be tested statistically.
CHAPTER 5. CONCLUSIONS AND FUTURE WORK

5.1 Contributions

The research in the information systems discipline has been classified into behavioral science and design science paradigms (Hevner et al., 2004). Design science studies aim at producing technology-based IT artifacts, such as constructs, models, methods, or instantiations, to solve relevant practical problems in organizations. The studies conducted in this dissertation follow the design science paradigm and aim to create a set of new and innovative IT artifacts (e.g., constructs, models, and methods) for collaboration process management.

Based on the theoretical bases of team collaboration and business process management, I addressed the issues in collaboration process management at three levels: theoretical models, system constructs, and application cases. Theoretical models (e.g., three-layer workflow modeling framework, collaboration process analytical framework, and collaboration virtualization theory) provide genetic guidelines for practitioners to manage collaboration processes. System constructs (e.g., formal definitions, algorithms, and prototypes) can be built into systems to (semi)automate collaboration process management. Application cases can be used to illustrate the value of applying theoretical models and system constructs. Moreover, theoretical models and system constructs will be improved based on practical insights from application cases. Each of the three studies in this dissertation developed one or multiple IT artifacts to address the issue of collaboration process management.
In Chapter 2, we proposed a framework for process model design via model transformation. We first divided process modeling into three layers and then presented a formal approach for transformation from conceptual to logical process models. For workflow model transformation, we proposed a semi-automated procedure to add information to a conceptual model and transform a conceptual model to a logical model. Further, the consistency between a conceptual model and its corresponding logical model is checked via an ontology-based approach. Finally, we conducted a case study and implemented a prototype system to evaluate the approach. To the best of our knowledge, our study represents the first attempt to (a) formally define three layers of workflow models, (b) transform a conceptual model to a logical model, and (c) check consistency between conceptual and logical models. While this study has direct practical implications for workflow model designers, it may also have theoretical implications for design theories of process models.

In Chapter 3, we proposed a framework for analyzing the relationship between collaboration processes and collaboration efficiency. The main contributions of this chapter are as follows. First, we proposed an analytical framework to study the effect of interaction patterns. The pattern analysis framework contains two steps: pattern mining and pattern impact assessment. Second, by applying the pattern mining method to software issue resolution processes, we identified 16 interaction patterns in issue resolution processes, which can be used as primary blocks for process analysis. Third, through a time-dependent Cox regression model, we found that most identified patterns
have significant correlations with work efficiency, which may change from positive to negative. Fourth, we conducted a case study to confirm findings about process patterns and explore the rationale and principles behind the correlation between interaction patterns and software development efficiency.

In Chapter 4, we proposed the CVT to explain the suitability of virtual collaboration in organizations. CVT extends existing theoretical work on collaboration management by trying to incorporate three categories of constructs (e.g., task, technology, and team) to explain collaboration virtualizability. Moreover, the construct of multi-task degree is defined as a unique concept in virtual collaboration and its relationship with virtual collaboration is discussed. The theory will also guide practitioners to consider how IT might help satisfy the requirements of virtual collaboration management.

In summary, this dissertation fills a critical void in business collaboration and process management by bridging the gap between collaboration research and process management research. As such, our research will have important theoretical and practical impacts.

5.2 Future Directions

Although this dissertation has addressed several challenges in collaboration process management, there are several potential future directions that can broaden our understanding of business collaboration and process management.
Firstly, we plan to develop a collaboration process management system to facilitate teamwork. This will be based on the process modeling framework and process analysis framework in Chapters 2 and 3, respectively. We will improve the prototype system for process modeling, implement the interaction pattern mining algorithms, and integrate them with workflow technologies. The system will allow collaboration managers to track, monitor, analyze, and control processes in teams. Once the system is complete, we will be able to test the usability of collaboration process management systems and mechanisms. In particular, this will help us refine our model on interaction patterns and efficiency through controlled experiments. These studies must address a bevy of issues, including appropriate user interface design, methods for enhancing the perceived usefulness of the system, mechanisms for error alerting messages when interacting with users, etc.

Secondly, we will conduct more field studies to evaluate and validate the constructs and models proposed in this dissertation. Most of the models in this dissertation were only validated through logical proof, system prototyping, and statistical models. Methods such as field experiments, surveys, and case studies will be used to evaluate and refine the theoretical models. Further, we will explore different types of collaboration teams and try to apply our approaches in different contexts. We only dealt with interaction processes in software development in this dissertation. However, collaboration can be found in almost every organization, and collaboration technologies are widely adopted. Moreover, people collaborate in many different ways, such as virtual
vs. physical, common interests vs. conflicted interests, and intra-organization and interorganization. We believe we will gain many more insights when we are closer to different types of real-life collaboration teams in organizations.
REFERENCES


2011 conference on Computer supported cooperative work, Hangzhou, China, ACM.


