8-11-2015

An Event-based Analysis Framework for Open Source Software Development Projects

Tianjie Deng

An Event-based Analysis Framework for Open Source Software Development Projects

BY

Tianjie Deng

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

Of

Doctor of Philosophy

In the Robinson College of Business

Of

Georgia State University

GEORGIA STATE UNIVERSITY
ROBINSON COLLEGE OF BUSINESS
2015

ACCEPTANCE

This dissertation was prepared under the direction of the *Tianjie Deng* Dissertation Committee. It has been approved and accepted by all members of that committee, and it has been accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Business Administration in the J. Mack Robinson College of Business of Georgia State University.

Richard Phillips, Dean

DISSERTATION COMMITTEE

Dr. William Robinson
Dr. Dr. Balasubramaniam Ramesh
Dr. Mark Keil
Dr. Kalle Lyytinen (Case Western Reserve University)

ACKNOWLEDGMENT

I would especially like to express my deepest gratitude to my advisor, Dr. William Robinson, for his excellent guidance, caring, and patience. My dissertation could not have been completed without him. I enjoy his creativity and his passion to programming. I especially enjoy brainstorming with him. I am fortunate to have him as my advisor.

I would like to thank my committee members: Dr. Balasubramaniam Ramesh, whose practical questions challenged me to grow, Dr. Mark Keil, whose constructive criticism helped me to improve, and Dr. Kalle Lyytinen, whose open-mindedness encouraged me to reach higher.

I am very grateful to Dr. Dan Robey and Dr. Mike Gallivan, my unofficial committee members. Dan has helped me greatly during my doctoral study. He has been a mentor, a motivator, and a close friend. He often refers to me as his “academic family.” His work in process models has have given me much inspiration. Dr. Gallivan**’s** suggestions and comments added considerable value to my study.

I owe a great amount of gratitude to Youyou Tao and Langtao Chen, my colleagues and close friends. They have spent so many nights helping me with my defense presentation and my job talks. Those stressful nights have now become some of my happiest memories.

I would also like to thank my colleagues, Arash Akhlaghi, Zirun Qi, and Aron Lindberg who helped me with data collection and coding. I also want to thank Hyung Koo Lee, who gave me helpful advice on statistics.

I was very fortunate to have met Lynet Mortensen, Scott Mortensen, and their children, Jake, Aaron, Carrie, and Elena, through the AMIGO Program in 2009. They have become my family ever since. I am SO thankful to have them in my life. Special thanks to Lynet’s sister, Terri Germann, who volunteered to edit my dissertation. I thank her very much for her time and effort.

I want to thank my uncle, Li Deng, his wife, Xiuwen Luo, and their children, George (Eagle) and Alex. They are my family in Atlanta. My uncle and auntie treat me like their own daughter. Because of them, I always have a home in Atlanta.

Last but not least, I want to thank my whole family for their support and patience, and for their faith in me.

Table of Contents

List of Tables

List of Figures

Abstract

An Event-based Analysis Framework for Open Source Software Development Projects

By

Tianjie Deng

July 2015

Committee Chair: *Dr. William Robinson*

Major Academic Unit: *Computer Information Systems*

The increasing popularity and success of Open Source Software (OSS) development projects has drawn significant attention of academics and open source participants over the last two decades. As one of the key areas in OSS research, assessing and predicting OSS performance is of great value to both OSS communities and organizations who are interested in investing in OSS projects. Most existing research, however, has considered OSS project performance as the outcome of static cross-sectional factors such as number of developers, project activity level, and license choice. While variance studies can identify some predictors of project outcomes, they tend to neglect the actual process of development. Without a closer examination of how events occur, an understanding of OSS projects is incomplete. This dissertation aims to combine both process and variance strategy, to investigate how OSS projects change over time through their development processes; and to explore how these changes affect project performance. I design, instantiate, and evaluate a framework and an artifact, EventMiner, to analyze OSS projects' evolution through development activities. This framework integrates concepts from various theories such as distributed cognition (DCog) and complexity theory, applying data mining techniques such as decision trees, motif analysis, and hidden Markov modeling to automatically analyze and interpret the trace data of 103 OSS projects from an open source repository. The results support the construction of process theories on OSS development. The study contributes to literature in DCog, design routines, OSS development, and OSS performance. The resulting framework allows OSS researchers who are interested in OSS development processes to share and reuse data and data analysis processes in an open-source manner.

1. Introduction

An open-source software development team is working on its sixth release, openPhotoBooth.06. During the past two years, the team has gone through various changes: new developers joined and old members dropped out, hundreds of bug requests were reported, the project's vision went through several shifts, thousands of commits were made, and the download numbers of the previous releases went up and down. Though release 4.0 was a huge success, the last release did not garner much attention. Now before the sixth release, the team is wondering, "Are we doing well so far? What is the likelihood that this version will be a success?"

This scenario raises interesting questions about Open Source Software (OSS) development projects: how do we utilize the huge amount of digital trace data of OSS projects to investigate their evolutionary patterns? Can we do more than just taking a "snapshot" of an ongoing OSS project and analyzing its static data at a given moment? How can we examine and understand the evolving trajectory of the projects to predict their future prospects?

1.1 Motivation

The success of Open Source Software development projects has generated the interest of academics and practitioners over the last two decades. Known for their "chaotic" development style, OSS projects have produced software with exceptional quality (Mockus et al. 2002). Surveys show that today's open source software has a higher quality than the industry average (Coverity 2012). Most extant literature attributes this superior quality to static cross-sectional factors such as number of developers, project activity level and license choice. How the everyday “chaotic” development activities and their characteristics shape OSS project performance remains under-researched.

Researchers have been engaging in analyzing OSS development processes (Mockus et al. 2002; Scacchi et al. 2006) and project performance (Crowston et al. 2003; Crowston et al. 2004; Crowston et al. 2006; Grewal et al. 2006; Lee et al. 2009; Raja and Tretter 2006; Ravi Sen 2012). However, there are several limitations in the current studies. First, many studies on OSS success take a “snapshot” of a given project at a given point, but the ever-changing nature of OSS projects and the dynamic structure of OSS teams require an evolution-oriented perspective. Second, extant studies mostly used a cross-sectional method to analyze project attributes, while overlooking the change pattern of development activities and processes. Consequently, the understanding of the evolution of OSS projects through their development processes is limited. Software quality is believed to be heavily dependent on development processes (Humphrey 1989). IS researchers have been advocating for the empirical examination of projects and IS projects through sequences of events (Lewin and Minton 1986; Lucas 1981; Van De Ven 1992) with a process strategy (Hirschheim et al. 1991; Sabherwal and Robey 1993; Sabherwal and Robey 1995). Although several studies have investigated the development processes of OSS projects, many of the studies adopted a qualitative method and used narrative descriptions of OSS development processes (Mockus et al. 2002; Reis and De Mattos Fortes 2002; Scacchi 2002a; Scacchi 2002b; Scacchi 2004; Scacchi 2005; Scacchi et al. 2006). Another set of studies applied a simulation approach to model and reenact OSS development processes (Jensen and Scacchi 2005; Smith et al. 2004). Among all these studies, the timing and sequence of the events in development processes received little attention. Another gap in these studies is that most of them study the processes with a single-case approach, rather than in a multiple project setting.

Within the era of “Big Data”, data with high volume, velocity and variety are made available to researchers (Mcafee and Brynjolfsson 2012). Open source software repositories, such as SourceForge, GitHub.com, and Google Code, produce and archive a large volume of public event data of OSS development, providing researchers with the opportunity to investigate multiple OSS projects from a longitudinal and process-oriented perspective.

In summary, despite the attribution of success to identifiable factors associated with OSS projects, many successes may still be regarded as “chaotic” and seemingly disregard conventional wisdom regarding project success. While variance studies can identify some predictors of project outcomes, they tend to neglect the actual process of development, which may or may not occur “chaotically.” Without a closer examination of how events occur, an understanding of OSS projects is incomplete. A systematic and quantitative method of empirically investigating how OSS projects evolve through development processes, as well as how the evolution of development processes impacts project performance, is warranted.

In this set of studies, a longitudinal, mixed-method computational method that combines sequence mining techniques with quantitative analysis is conducted to analyze digital trace data of the development processes of 103 OSS projects. As a framework, EventMiner, is designed and instantiated to automatically collect, preprocess, classify, and analyze process digital trace data from open source repositories. Drawing from tenets of design routine literature (Gaskin et al. 2014; Gaskin et al. 2011; Gaskin et al. 2010) and the theory of distributed cognition (DCog) (Hutchins 1995; Hutchins and Klausen 1996), the framework classifies the obtained process data into sequences of theory-based constructs, for analysis and interpretation. This framework automatically detects potential changes in event streams. Using this framework, I am able to detect behavior transitions in patient event streams, and behavior transitions in OSS development processes. In addition, 103 OSS projects are clustered into five groups, based on their sequence of development events. Finally, building on a *design routine* perspective (Gaskin et al. 2014; Gaskin et al. 2011; Gaskin et al. 2010), a factor model of impacts of development routine diversity and routine change on project success is developed.

1.2 Research Objectives and Overview of Studies

This dissertation follows the multi-paper model and includes three studies. The objective of this set of studies is to investigate how OSS projects evolve through development events, and how the evolution of development processes affects project performance. Therefore, the general research questions are:

How and why do OSS projects evolve through development processes?

What are the impacts of development processes' evolution on project performance?

To answer these two questions, this study approaches OSS development activities as *design routines*, performed by developers through *distributed cognition processes* (Hutchins 1995; Hutchins and Klausen 1996).

1.2.1 Key Terms

Before introducing the overview of the dissertation, I will define key terms to be discussed. Gaskin et al. (2011) define a design routine as "a sequence of (design) tasks, which transform some representational inputs into a set of material and representational outputs, leading ultimately to a generation of design artifact." In accordance with this definition, I define OSS routines as "a sequence of OSS design activities, which are patterns of behavior executed by participants to perform a design or development task." Those routines vary and change over time (Feldman 2000; Gaskin et al. 2011), due to changing and ambiguous goals, shifting requirements, and internal learning (Gaskin et al. 2011). Given the dynamic nature of OSS, the design routines described in this study are more dynamic than ostensive routines with standardized steps (such as payroll). In particular, in this dissertation, I will focus on two types of routines: issue handling routines and pull-request handling routines. A typical *issue handling routine* begins with a member of an OSS project reporting an issue in the current code base. Following this initiating event, other activities are performed: contributors can comment on this issue, make suggestions, or reference it to other issues or solutions. Developers can also make commits to the code, and reference the commits as "solutions" to the issue. If a solution is made, or if the core team decides the issue is not worth pursuing, this issue will be closed, and thus the routine ends. Otherwise, an issue will remain open. A typical *pull request handling routine* starts when a developer posts a pull request, proposing to commit changes to the current code base. Members then comment on the pull request, make suggestions, and reference other pull requests and issues. The pull request is also reviewed by core developers, determining how well the proposed commits follow the repository's standards. The pull request is either merged by core developers into the main source code, or rejected. Finally, the pull request is closed.

I chose these two types of routines for two reasons: first, they are common patterns in OSS development (Dabbish et al. 2012). Second, through these two types of routines, uncertain requirements are transformed into artifacts (software) through unstandardized procedures.

I define routine diversity and routine change similarly to Feldman and Pentland (2003). *Routine diversity* is defined as the number of different configurations/patterns of the same routine type (Pentland et al. 2011), and the number of different routine types observed *within a given period*. *Routine change* is defined as the change of routine configurations/patterns over time.

A framework is a structure intended to serve as a support or guide for the building of something that expands the structure into something useful. In particular, a software framework is a reusable architecture for various application domains (Pree 1994). I use the term framework for both methodological framework and "small" software framework that consist of reusable components. Therefore, I define a framework as a set of methodological guidelines to support sequence data analysis and process theory building. It can also be considered as a software framework that provides general sequence mining functionalities and components, which can be reused and modified for different applications.

1.2.2 OSS Development Processes Comprised of Design Routines

Design routines are believed to have less clearly defined inputs and outputs as a result of changing requirements (Dorst et al. 1996; Gaskin et al. 2011). Unlike ostensive routines such as payroll, design routines are more fluid (Gaskin et al. 2011). Given the dynamic nature of OSS development, one can expect that design routines are prevailing in OSS development processes (Gaskin et al. 2011; Gaskin et al. 2010), and that changes in development behavior can be reflected in the changing routines. Therefore, I suggest that changes in OSS development processes can be analyzed though changes of routines over time.

In the OSS contexts, routines are performed in a distributed way. From a sociomaterial view, those routines are comprised of social and material elements; they are performed through the collaboration of social actors, who draw upon “informalisms” to finish design and development tasks. To investigate how OSS design routines change and evolve in a distributed way through social and material factors, I will introduce the theory of distributed cognition as the analytical lens.

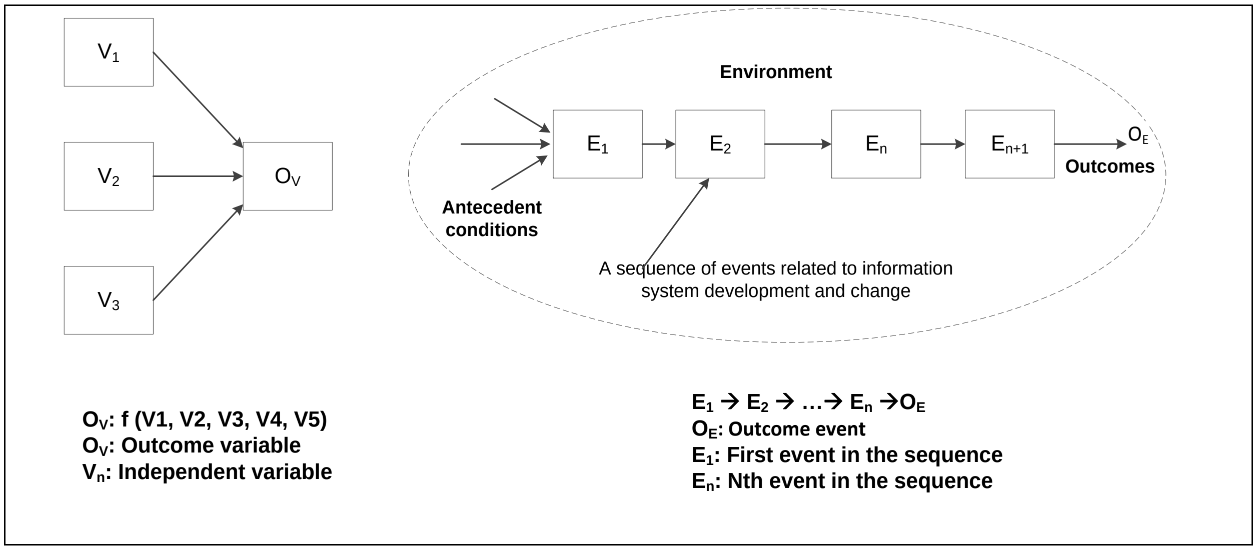
1.2.3 OSS Development as Distributed Cognitive Processes

The theory of distributed cognition (DCog) developed by Hutchins (Hutchins and Klausen 1996) extends the boundary of cognition processes from individual to socio-technical systems. It explains how cognition processes distribute socially, structurally, and temporally when a distributed team’s members collaborate on information processing tasks. *Social distribution* concerns the dispersion of activities across people and organizations. *Structural distribution* deals with the spread of knowledge across representational media and computational artifacts for knowledge validation and propagation. The perspective of DCog is well-suited to the OSS project development context; the social and structural contexts of OSS projects are rather unique. An open source project can have much more distributed social structure than a traditional one; an OSS team is normally self-organized, developers are often geographically distributed and have developed a unique mechanism for task assignment in which they self-assign their technical roles (Gacek and Arief 2004; Scacchi et al. 2006; Ye and Kishida 2003), the decision making is likewise decentralized (Gacek and Arief 2004). Scholars have observed an onion-like social structure in OSS projects (Gacek and Arief 2004; Mockus et al. 2002), which indicates that knowledge, decision making, and technical roles are distributed across different groups of actors in an

OSS team. From the structural distribution perspective, OSS communities rely heavily on various external artifacts and structural representations such as emails, websites, forums, and chats, for cognitive task processing. These structural representations and artifacts are referred by Scacchi as "software informalisms" (Scacchi 2002b). One of the reasons that OSS development is considered to outperform the commercial method is the implementation of distributed and concurrent design and testing (Kogut and Metiu 2001). Given the distributed characteristics of OSS development, DCog can serve as an appropriate lens to investigate how OSS development processes evolve socially and structurally. Therefore, I draw upon DCog to explore the social and structural factors involved in the evolution of OSS development behavior.

1.2.4 Towards a Process Perspective of OSS Development

One ultimate goal of this set of work is to analyze OSS project development processes toward supporting the construction of process theories on OSS development process change. Since Mohr advocated the process perspective for studying organizational changes in 1982 (Mohr 1982), and Markus and Robey introduced the perspective to IS in 1988 (Markus and Robey 1988), it has been commonly acknowledged that variance perspective and process perspective are two major alternatives for studying information systems. In a variance (factor) perspective, models are developed to predict outcomes from predictor static variables, whereas process theories are concerned with explaining how outcomes develop over time (Markus and Robey 1988). A comparison between variance models and process models is presented in Figure 1. More concepts and research in process theories is introduced in 2.4.



V_1
V_2
V_3
O_V
Environment
E_1
E_2
E_n
E_{n+1}
O_E
Outcomes
Antecedent conditions
A sequence of events related to information system development and change
O_V: f (V1, V2, V3, V4, V5)
O_V: Outcome variable
V_n: Independent variable
$E_1 \rightarrow E_2 \rightarrow \ldots \rightarrow E_n \rightarrow O_E$
O_E: Outcome event
E_1: First event in the sequence
E_n: Nth event in the sequence

Figure 1. Comparison between Variance Models and Process Models

Researchers who advocate for process models suggest that they complement variance models (Newman and Robey 1992), especially in studying organizational change. Mohr pointed out that the variance perspective was ill-suited to studying change, and process models are useful in explaining IS changes (Mohr 1982). Similarly, Lyytinen and Newman argued that "(change explanations using variance theories) close-box the change process and mask its dynamics and generative mechanisms" (Lyytinen and Newman 2008). Yet, over the past 20 years, only 20% of articles in leading IS journals used a process perspective (Paré et al. 2008). Although Mohr suggested that the process perspective and variance perspective should not be combined within a single research study (Mohr 1982), recent studies have been advocating for the joint use of the two strategies (Burton-Jones et al. 2014; Sabherwal and Robey 1995). To address my research questions regarding change in OSS development processes, a process perspective is an appropriate and fruitful strategy.

In this dissertation, I use both process strategy and variance strategy to examine OSS development processes. First, I develop a sequence-mining-enabled framework to automate the task of detecting and displaying patterns of events and patterns of change. I will demonstrate this framework in Study 1 of Chapter 3. Second, I draw upon distributed cognition (Hutchins and Klausen 1996), hidden Markov modeling and process theorizing to identify and interpret changes in OSS projects, for the purpose of formulating process theories on OSS development change. This work is described in study 2 of Chapter 4. Finally, in my last study presented in Chapter 5, I will make a preliminary effort to develop a variance model that captures the relationship between development process change and project performance. This model includes measures of project variability and change, which represent the process of OSS development. Thus, process and variance approaches are combined in the investigation of OSS development.

In summary, building on concepts from design routines, distributed cognition, process theories, and sequence mining, I designed and developed a framework and artifact, EventMiner, to acquire, classify, and analyze low-level sequential patterns and their aggregation as design routines from OSS development event data. By applying this framework to various data sources, I conducted three studies to investigate OSS development process evolution. In the following section, I will describe the topic of each study and main research questions addressed by it.

1.2.5 Study Overview

To model project evolution dynamics, the framework should first detect behavioral changes and identify specific behavioral patterns. Analyzing the sequence of events can reveal event patterns as a project evolves. Abnormal changes in the sequence of events indicate transitions. Detecting significant behavioral transitions would provide a base for further investigation on how and when the changes occur. It can also serve as a platform for future investigation on any outcome that might be associated with changes in development processes, such as project performance. This implies the following research questions:

RQ1: How can a framework be developed to facilitate the detection of behavioral transitions in OSS projects?

RQ2: What patterns of evolution are specific to OSS project development activities?

RQ3: How are sequential patterns related to OSS project performance?

To answer these three questions, I developed a framework and artifact, EventMiner, for behavioral monitoring and demonstrated its application to patient event data in study 1. As a framework, EventMiner serves as a methodological support and guide for building process theories from event stream data. It provides methods for analyzing event stream data in computational and mixed methods research. These methods transform incidents in event streams into theory-based entities (Van De Ven and Poole 1990). It is also a software framework that provides general, reusable sequence mining components. As an artifact, EventMiner provides functionalities for sequence mining event stream data and for detection transitions. In this study, I demonstrate how this framework and artifact can be applied to monitor software usage behavior. This study, as an initial attempt to develop tools for computational analysis of process data, builds the base for answering RQ1 and RQ2.

In the second study, I apply EventMiner to the OSS context. Using DCog as the theoretical lens, I detect behavioral transitions in OSS projects. Furthermore, I cluster 103 projects into five groups based on their sequential patterns, and find that projects in different groups have significantly different performances. This study addresses RQ1 – RQ3.

This study and the resulting artifact, automate the process of identifying (1) behavioral transitions, similar to the concepts of “encounters” refereed by Robey and Newman (1996), and (2) “recurring sequences of events that comprise the social process of ISD” (Markus and Robey 1988). As pointed out by Markus and Robey, “the resulting ‘pictures of the process’ can support a variety of theoretical interpretations” (Markus and Robey 1988), for building process models. These transition points and recurring patterns, can be interpreted by DCog, or other theories, as the base for process theory building. I will extend this study to building process theories on OSS development, in future research.

In the third study, I approach the OSS development processes from a design routine perspective. I develop a factor model to connect development behavior and behavioral changes, with project performance. This model includes measures of project variability and change, which represent the process of OSS development. In this way, I combine process and variance approaches, in examining the relationship between OSS development processes and project success. Research questions addressed by this model are:

RQ4: What factors drive routine diversity and routine change in OSS projects?

RQ5: What is the impact of routine diversity and routine change on OSS project performance?

Table 1 lists the topics presented and research questions addressed in each study.

Study	Topic	Research Objective	Research Questions
Study 1	Discovery and diagnosis of behavioral transitions in patient event streams	To build EventMiner and demonstrate its application to the detection of behavioral changes	R1
Study 2	(a) OSS development behavior transition discovery (b) Clustering of OSS projects based on sequential patterns	To detect behavioral transitions and evolutions of sequential patterns	R1-R3
Study 3	Investigating the temporal dynamics and variety of OSS development activities	To investigate the relationship between OSS design routines and project performance	R4-R5

Table 1. Study Overview

In addition, the study produces EventMiner, which is both an event-based automatic analysis framework and an implemented artifact. IS researchers have been advocating for the sharing of both data and analysis practices in the IS discipline (Lyytinen 2009). OSS researchers also suggested sharing and analyzing OSS process data in an open-source manner (Scacchi et al. 2006). Similarly to OSSmole developed by Howison and his colleagues (Howison et al. 2006), EventMiner can serve as both an open data repository and an open source toolkit for analyzing process data. It consists of five components: (1) a raw data extraction component to automatically extract raw data from public OSS repositories, (2) an extracting method that incorporates the theory of distributed cognition to determine data of interest, (3) a rule-based classifier that incorporates DCog theory to map extracted data into theory-based *constructs*, (4) event-based sequence mining techniques to analyze event data stream, and (5) an open data repository containing OSS process data (Figure 2). This framework can be used by researchers who are engaged in OSS development process research and OSS evolution research in particular, and in process research in general. Researchers for data analysis and theory building can reuse each component. For example, the rule-based classifier component can be reused for event classification with any analytical lens of the researcher's interest. However, a "rule engineer" needs to modify the rules, based on the theory or ontology he or she wants to use for event classification.

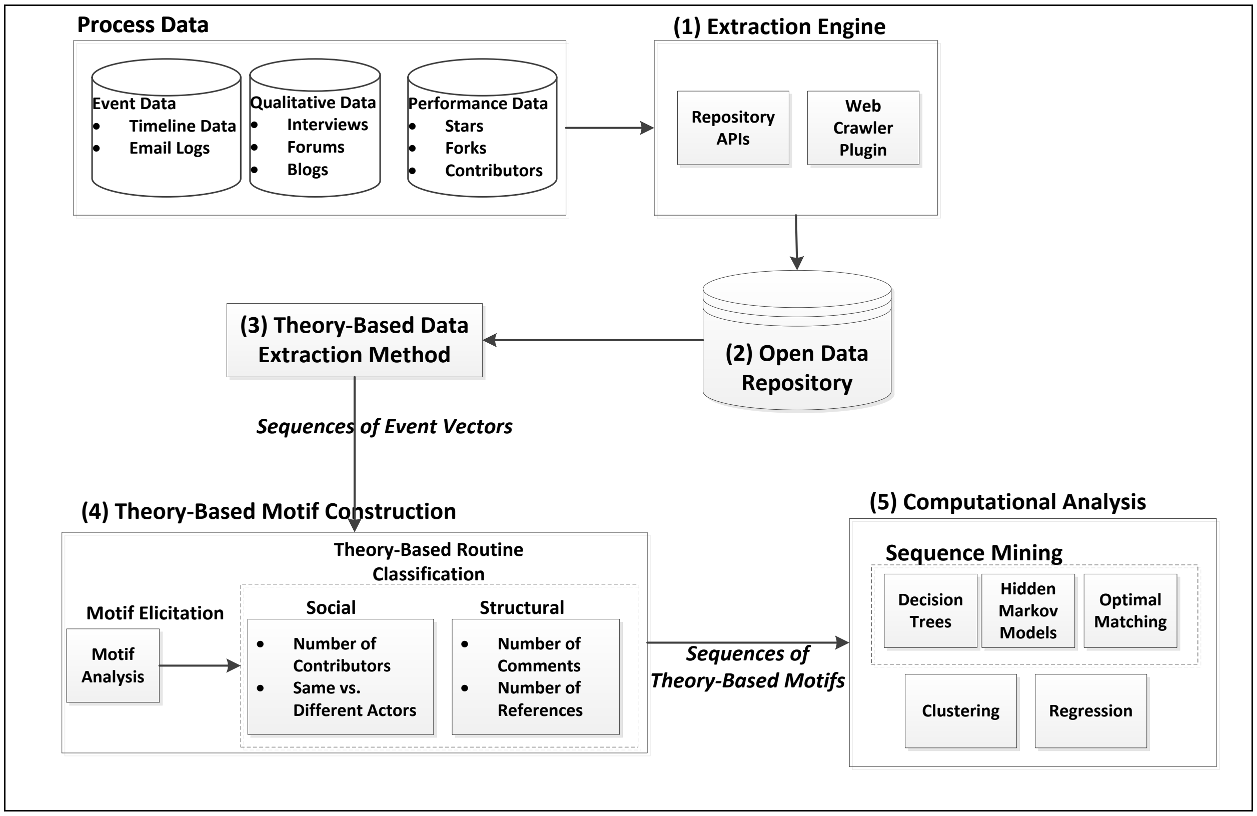


Figure 2. The EventMiner Framework

2. Literature Review

In this chapter, I summarize the state of the art in related research areas. I will introduce extant literature in OSS evolution and OSS development routines. In particular, I will discuss literature on routine diversity and routine change. I will also review the history and major works in process theory research. Then I will introduce the theory of distributed cognition, which will serve as the theoretical lens for development process analysis. After reviewing current studies in OSS performance, I will introduce the sequence-mining techniques that guided the design of EventMiner.

2.1 Evolution of OSS Development Process

While software evolution of commercial systems has been in focus of research since Lehman and Belady's work on IBM's OS/360 operating system (1976), there have been only a handful of studies looking into evolution and evolutionary patterns in open source software projects. Several of those studies reported a super-linear or exponential growth rate of individual OSS projects (Scacchi 2002a; Scacchi 2006; Smith et al. 2004), contradicting the well-established Lehman's law (Lehman 1980). Similar patterns were reported in Godfrey and his colleagues' series of studies on evolutionary patterns of open source software projects including Linux Operating System Kernel, VIM text editor, fetchmail utility and GCC compiler suite (Godfrey and Tu 2001; Godfrey and Lee 2000; Godfrey and Tu 2000; Tu and Godfrey 2001). They examined evolution both on the system level and subsystem level. Godfrey and Tu (2001) conducted a case study on the evolution of the Linux Operating System Kernel over its six year lifespan, and found the growth at system level to be super-linear. They suggested that researchers should take into account the nature of the subsystems when studying their evolutionary patterns. They also identified cloning as a useful practice for open source projects to evolve, as opposed to the common belief that it is an indicator of poor process (Godfrey and Tu 2001). Koch also looked at OSS project evolution from the perspective of growth rate, and found that a quadratic model outperformed a linear model when modeling evolutionary behavior (Koch 2005). In those studies, cloning, modularization, and self-selection for tasks are found to be beneficial to the evolution of OSS projects (Godfrey and Tu 2001; Koch 2005). Overall, those studies focused on the system and its growth behavior, not on the community or its development behaviors.

Nakakoji et al. (2002) expanded the focus of the investigation on OSS system evolution to the evolution of OSS communities, and further studied the relationship between the two types of evolution. They found that different collaboration models resulted in different evolutionary patterns of system and communities; for system evolution, GNU and Jun evolve as a single version tree; Linux allows multiple implementations for the same functionality, and postgreSQL starts with multiple patches but those patches which merge into one single core version. As for the evolution of communities, they concluded that it is determined by the existence of motivated members and social mechanism of the community. This paper proposed to classify OSS projects into three types based on those difference models: exploration-oriented, utility-oriented, and service-oriented. Besides evolution in project size and project communities, there are also studies looking into evolution on OSS developers' individual participation in projects. Qureshi and colleagues looked into the growth patterns of developers' socialization behavior and how that behavior relates to their status progression (Qureshi and Fang 2011). They discussed joiner's nonlinear growth trajectory and classified those trajectories into four types based on two dimensions: initial level of social resources of the developer (low, medium and high) and growth rate of his/her socialization (low, medium and high). Although there could be nine possible types of trajectories, the data only yielded four types with different combinations of initial level of social resources and growth rate. Researchers have identified five primary entities as suitable targets in the study of software evolution: software releases, systems, applications, development processes, and process models (Lehman 1980; Lehman and Ramil 2003; Scacchi 2006). Most of the above studies focused on the evolution patterns of the system, application or community, but lacked a process-oriented perspective (Langley 1999).

2.2 OSS Development Routines

Routines are important to organizations because they are performed to accomplish work in organizations (Cyert and March 1963; Feldman 2000; March and Simon 1958; Nelson and Winter 1982). Feldman defines routines as "repeated patterns of behavior that are bound by rules and customs and that do not change very much from one iteration to another" (2000, p.611). However, she pointed out a "discrepancy between the concept and the observation", with the observation that routines were "undergoing substantial changes" (Feldman 2000, p.611). Cohen et al. (1996) define a routine as "an executable capability for repeated performance in some context that has been learned by an organization in response to selective pressures''. Although slightly different from each other, these definitions all consider routines to be repeatable behavioral patterns that also have the inherent potential to change.

Routines and the diversity of routines, have been commonly studied in both organizational literature (Feldman 2000; Feldman and Pentland 2003) and IS literature

(Gaskin et al. 2014; Gaskin et al. 2011). Recently, Gaskin et al. (2011) argued the importance of studying the variations of design routines and defined a design routine as "a sequence of (design) tasks" (Gaskin et al. 2010).

2.3 Routine Diversity and Routine Change

2.3.1 Routine Diversity

Routine diversity has been referred to by different terms such as routine variation (Gaskin et al. 2014; Gaskin et al. 2011; Gaskin et al. 2010; Pentland et al. 2011), or routine heterogeneity (Lindberg et al. 2015a; Lindberg et al. 2015b). There are conflicting theories on the value of routine diversity. Pentland et al. (2011) believe that variation is "a prerequisite for change." Page argues that diversity can enhance the robustness of complex and adaptive systems (Page 2010). Diversity and variation are considered to be the foundation for learning in general (Campbell 1960; Weick and Kiesler 1979) and for learning in routines (Levitt and March 1988). Routine diversity allows actors to work in different ways, which provides the flexibility required by dynamic environments. In the context of open source software development, Lindberg conducted a case study on Rubinius, an open source project, to investigate the co-evolving relationship between open source software development coding practice and communities (Lindberg 2013). A positive relationship between practice diversity and inflow of new developers was reported.

Conversely, some studies advocate for the reduction of process variation to improve process performance. The six sigma framework argues in favor of minimizing process variation (Schroeder et al. 2008). Frei et al. investigated the relationship between service process variation and firm performance in the retail banking industry (Frei et al. 1999). They found that process variation is negatively related to firm performance.

Researchers have also attempted to identify drivers of the increases or the decreases in routine diversity. Environmental contexts, such as degree of digitalization (Gaskin et al. 2011), degree of centralization (Gaskin et al. 2011), degree of volatility (Gaskin et al. 2011; Sutton 2000), structural variation of the community (Crowston and Howison 2006), social and technical challenges (Lindberg et al. 2015b), and social discourse are explored (Lindberg et al. 2015b; Scacchi 2009; Winograd 1987). Sutton argued that start-up companies in a "fast-paced, reactive, and innovative" environment may not have the foundation to have repeatable and predictable processes, which are required by CMM. On the contrary, in an environment where flexibility is more important than structure, companies simply need more flexible and adaptable processes (Sutton 2000). Gaskin et al. (2011) found that design routines have less variation among more centralized organizations than among networked organizations.

2.3.2 Routine Change

Routines are a central element of organizations and have been the subject of discussions on organizational stability and change (Cyert and March 1963; Feldman 2000; March and Simon 1958; Nelson and Winter 1982). Pentland et al. suggest that routine changes are changes in "patterns of action" (2011,p.1371). Researchers hold different views on whether and how routines contribute to organizational stability and change. Some researchers conceptualize routines as stable, repetitive, standard operating procedures that do not change (March and Simon 1958). Others hold a "routine as change" (Feldman 2000; Feldman and Pentland 2003; Pentland et al. 2011), viewing routines as "continuously changing entities"(Geiger and Schröder 2014. page 171). Pentland et al. (2011) argue that every performance of a routine is different due to the different time, places, actors and other objects involved. Levitt and March (1988) attribute the routine change to direct organizational experience.

Changes of routines have been empirically studied. Different dimensions of routine change include: rhythm of change (Klarner and Raisch 2012), frequency of change (Klarner and Raisch 2012), magnitude of change (Pentland et al. 2011), and types of change (Feldman 2000). Feldman analyzed the changes of five routines in an organization for four years. She found that routines change as a response to problems. This view is consistent with Levitt and March's theory that routines change in response to evaluation of outcomes. Pentland et al. (2011) studied changing routines in invoice processing in four organizations. They found that some routines are stable, whereas the patterns of actions generated by those routines, which are also routines, do change over time. They noted that one cause of routine changes is inexperienced users; however, routines also changed due to the allowed variability in the system and its potential for user exploration (Pentland et al. 2011).

Adapting from the concepts of "requisite variety" (Ashby 1956), Beer developed a Viable System Model (VSM) (Beer 1975; Beer 1979; Beer 1981). VSM is concerned with the essential characteristics that systems need to survive in a forever-changing environment. A system increases its degrees of freedom to accommodate greater variety in requirements, by increasing it variety (Beer 1979). That is, any system has to be able to generate a variety equivalent to the variety of the system to be regulated (Beer 1979). It is important to note that both the environment and the organization should be complex systems to allow the behavior to emerge (Snowdon and Kawalek 2003).

Similarly, complexity theory suggests that diversity enhances the robustness of complex adaptive systems (Benbya and Mckelvey 2006; Page 2010). ISD projects are commonly considered as complexity adaptive systems (CAS) (Benbya and Mckelvey 2006; Van Aardt 2004). OSS projects, are considered by several researchers to be the best example of CAS

(Muffatto and Faldani 2003; Van Aardt 2004). Complexity is magnified in the OSS development context, by the continuous changes in requirements and thus the continuous changes in resulting artifacts. With such complexity and changes, it is essential for the development system to have the requisite variety in its processes. For example, Lindberg et al. (2015b) suggest routine variation as a "coping mechanism" to both social and technical problems.

2.4 Process Models

A process model "explains development in terms of the order in which things occur and the stage in the process at which they occur" (Van De Ven 2007). The process model perspective is advocated by researchers for its advantage in explaining organizational changes (Mohr 1982). Markus and Robey first introduced it to IS discipline in their study (Markus and Robey 1988). Newman and Robey (1992) stated the importance of treating the information system development (ISD) process as a "dynamic social process," and to conceive it as a "sequence of events that occurs over time."

A process theory can be derived from data by analyzing event sequences. An event can be viewed as the change in state of some variable values (Chandy and Schulte 2009). An event sequence can be characterized by common metrics, such as length, entropy, subsequences, pattern frequency, and similarity to other sequences. Through abduction, a researcher can infer common constructs and higher level concepts, and eventually relate these terms to a theory that can explain relationships among concepts within a set of boundary conditions (Van De Ven 2007).

The variance perspective and process perspective are often considered a dichotomy (Markus and Robey 1988; Mohr 1982; Seddon 1997). However, several researchers argued that although the two perspectives cannot be used in the same model, they can be combined together as a method. For example, in their 1995 paper, Sabherwal and Robey demonstrated how variance strategy and process strategy can be reconciled (1995). They applied variance strategy to the examination of levels of participation of key actors and process strategy with sequences of actions, to 50 ISD projects. The results showed that projects that are similar based on participation are also similar based on event sequences. More recently, Burton-Jones et al. suggested “a shift from the traditional process-variance dichotomy to a broader view,” by using different perspectives more flexibly (Burton-Jones et al. 2014). They illustrated how one can use different theoretical perspectives (variance, process and system perspectives) to critique and extend the IS success model.

2.5 Distributed Cognition

Proposed by Hutchins and colleagues in the late 1980s and early 1990s, distributed cognition theory (DCog) (Hollan et al. 2000; Hutchins and Lintern 1995; Johnson-Laird 1989; Newell 1980; Simon and Kaplan 1989; Wright et al. 2000) views cognition as a process of computation and expands the unit of analysis from the individual to a socio-technical system attending to a specific task. The fundamental concept of this theory is that in collaborative projects, information processing activities are not limited to individuals, but are distributed among participants, artifacts, and the environment. According to Hutchins, there are three important facets within distributed cognition. Cognitive processes can be distributed socially, structurally, and over time. *Social distribution* refers to the distribution of social actors among the projects: each member of the team plays a specific role when processing the information. *Structural distribution* refers to cognitive processes involving coordination between internal and external structure. Individuals and teams employ external structures such as artifacts in their information processing activities. Finally, cognitive processes distribute in a temporal manner in which the prior cognitive processes will influence the future ones.

2.5.1 Distributed Cognition in Software Development Projects and OSS

Researchers from different disciplines have studied DCog in several contexts such as airline and navigation systems (Hutchins and Klausen 1996; Hutchins and Lintern 1995), human computer interaction (Hollan et al. 2000; Wright et al. 2000), peer tutoring (King 1998), collaborative activities in different organizational settings (Rogers and Ellis 1994), and classroom practice (Hewitt and Scardamalia 1998). More recently, scholars have started to explore distributed cognition in the discipline of IS development practices. Lyytinen and colleagues have conducted a series of studies to build a distributed process model of RE practices, both in traditional software projects and OSS projects (Hansen and Lyytinen 2009; Hansen et al. 2012a; Thummadi et al. 2011). Hansen and Lyytinen applied a multi-case study approach to explore how requirements are distributed socially, structurally, and temporally in RE practices (Hansen and Lyytinen 2009). Later, they examined the nature of distributed cognition in RE practices in a more systematic manner by including both a case study and simulation experiment (Hansen et al. 2012a). In this study, the authors identified several DCog related goals, along with other goals in RE practice, and analyzed how different RE tasks satisfied those goals by running a simulation. They suggested DCog could serve as a lens to analyze RE processes and provide implications for RE process design and process efficiency evaluation. In their study on the quality of RE in open source projects, Thummadi et al. (2011) proposed to investigate how social, structural, and temporal dimensions of distributed cognition impact the quality of requirements in open source development. Table A1 in the Appendix summarizes the matrices used or proposed to measure the three components of DCog (social distribution, structural distribution, and temporal distribution) from existing studies focusing on software development activities.

2.6 OSS Performance

Studies in OSS success have been investigating both identification of determinants of OSS successes and definition of OSS success measurements. The identified factors leading to success mainly fall into two categories: time-invariant factors and time-dependent factors (Subramaniam et al. 2009). In the category of time-invariant factors, researchers have repeatedly investigated license choice (Comino et al. 2007; Lerner 2005; Sen et al. 2012; Stewart et al. 2006a; Stewart and Gosain 2006a; Stewart and Gosain 2006b; Subramaniam et al. 2009). Most of these studies found that restricted licenses do not provide developers with the freedom to modify the code, and thus the licenses have an adverse impact on OSS success. Comino et al. (2007) and Sen et al. (2012) have discussed how the intended user type might affect project success. For example, Comino et al. (2007) found applications for more sophisticated users are more likely to evolve successfully. Other time-invariant factors include whether the project accepts financial donations (Sen et al. 2012), as well as the choice of programming language (Sen et al. 2012; Subramaniam et al. 2009), operating system (Sen et al. 2012; Subramaniam et al. 2009), developer motivation and interest (Bonaccorsi and Rossi 2003), sponsorship (Stewart et al. 2006a), and modularity (Bonaccorsi and Rossi 2003; Giuri et al. 2010; Mockus et al. 2002).

Several time-dependent factors have been investigated in terms of their impact on OSS success. For example, project activity level is believed to be an important factor contributing to OSS success (Crowston and Scozzi 2002; Stewart et al. 2006b; Stewart and Gosain 2006b; Subramaniam et al. 2009). An OSS project's activity level can be indicated by the number of files released, number of bugs fixed, and number of commits. Some other examples of time-dependent factors include age and duration of the project (Beecher et al. 2009; Indyk et al. 2000; Sen et al. 2012), size of the project and community (Beecher et al. 2009; Comino et al. 2007; Mockus et al. 2002), developer interest over time (Bonaccorsi and Rossi 2003; Subramaniam et al. 2009), user interest (Subramaniam et al. 2009), project status (Subramaniam et al. 2009), and knowledge of developers and users (Mockus et al. 2002). A group of studies has also focused on the social aspect of OSS projects. For example, while an effective collaboration structure and process (Bonaccorsi and Rossi 2003; Méndez-Durón and García 2009) can contribute to OSS success, Bonaccorsi and Rossi (2003) also suggested that "a widely accepted leadership" is important. Grewal and associates (2006) examined the effect of network embeddedness on project success. They used network embeddedness to capture the architecture of network ties, and defined structural, junctional, and positional embeddedness as its subconstructs. They found heterogeneity existed in network embeddedness in OSS projects and that the effect is quite complex. Both positive and negative effects were observed. Singh and colleagues (2008) investigated network social capital and found that internal cohesion among the developers has a positive impact on success, while external cohesion (the cohesion among the external contracts of the project) does not always benefit the project. Knowledge flow direction (Méndez-Durón and García 2009) within a network and a developer's status in the network were also studied (Frank 2008).

Many different measurements of OSS success have been used in these studies mentioned above. To obtain a better understanding of OSS success and to help facilitate the empirical study of OSS success, several scholars attempted to systematically define OSS success and its measurements. Most of them found that the measures of OSS success are interrelated. A summary of those measures is in **Table A2**. Crowston and colleagues have conducted three studies to define OSS success and measures (Crowston et al. 2003; Crowston et al. 2004; Crowston et al. 2006). They reviewed existing IS success models developed by Delone and Mclean (1992) and Seddon (Seddon 1997) to propose potential measurements for the OSS context. They then reexamined the OSS development process to remove measurements, which are not applicable to OSS and proposed additional measurements. They categorized measurements of OSS success into three types: measurements concerning the process, measurements concerning project output, and measurements concerning outcomes for project members. Similarly, Lee and colleagues proposed five measures based on DeLone and McLean's model by incorporating OSS characteristics: software quality, use, user satisfaction, individual net benefits, and community service quality. They then tested the relationship among those measures and found that usage is determined by user satisfaction and software quality; and user satisfaction is determined by software quality and community service quality. Based on the categorization summarized by Crowston and colleagues (2006), some commonly-used success measures in previously discussed empirical studies regarding project outputs are: user and developer satisfaction (Crowston et al. 2006; Lee et al. 2009), project status (Comino et al. 2007; Crowston and Scozzi 2002), and community service quality (Lee et al. 2009). User interest has been used often, being operationalized as the number of downloads (Crowston and Scozzi 2002; Grewal et al. 2006; Méndez-Durón and García 2009), page reviews (Crowston and Scozzi 2002), and number of subscribers (Sen et al. 2012; Subramaniam et al. 2009). Because user interest can change over time, some studies have measured the change of number of subscribers over time (Stewart et al. 2006b). Another indication of success for project outcome suggested by Crowston, Howison and colleagues (2006) is project completion. This often measures the technical achievements of the project, such as movement from alpha to beta to stable and the achievements of the identified goals (Crowston et al. 2006). Size achieved is also a common measure, in the form of lines of code (Beecher et al. 2009), or number of commits (Giuri et al. 2010; Grewal et al. 2006) . Other measurements to evaluate the achievements of the project include number of releases (Giuri et al. 2010; Grewal et al. 2006) and defect density (Mockus et al. 2002). Another set of OSS success measurements focuses on the development process. For example, during the development process, a more successful OSS project would be able to attract inputs from developers (Stewart and Gosain 2006a). Therefore, the number of developers of an OSS project has been used repeatedly to measure success (Beecher et al. 2009; Crowston et al. 2006; Sen et al. 2012; Stewart and Gosain 2006a; Subramaniam et al. 2009). Another important indicator of success is activity level, which can be measured by the number of commits within a time period (Beecher et al. 2009; Grewal et al. 2006), or the number of releases (Crowston et al. 2003; Crowston et al. 2006; Crowston and Scozzi 2002). Others have suggested or used cycle time such as the time required to fix bugs or implement features (Crowston et al. 2006; Mockus et al. 2002), or the time between releases (Crowston et al. 2003; Crowston et al. 2004; Crowston et al. 2006). A third group of OSS success measurements are concerned with outcome for members, such as individual jobs, opportunities and salaries, knowledge creation, and individual reputations (Crowston et al. 2006). Lee and his colleagues used individual net benefit as one construct and found that it is influenced by use and user satisfaction (Lee et al. 2009).

Different methods are deployed to explore the success of open source projects. There have been several case studies which attempted to explore the nature of open source projects and their success (Mockus et al. 2002; Stamelos et al. 2002). Later, more researchers started to study OSS success quantitatively, trying to identify the factors contributing to it and quantify the effects of those factors. There are also studies applying data mining methods to predicting OSS success (Raja and Tretter 2006; Wang 2007). Raja and Tretter (2006) used three data mining techniques to mine OSS data to predict success: logistic regression, decision trees, and neural networks. The factors identified as contributors to success are: number of downloads, number of bugs reported, team size, and use of a project manager or not. They also used text mining to cluster the projects based on their description. Wang (2007) used k-means clustering to predict OSS project success, with a performance accuracy of over 94% (i.e., just 2 prediction failures among 42 projects).

However, most of those studies looked at static attributes of projects, without investigating the dynamic changes of projects and teams while those projects evolve. Subramaniam et al. (2009) pointed out that since most open source software projects are continual, the dynamics should be analyzed. Aksulu and Wade (2010) also suggested that "multi-dimensional frameworks covering all procedural stages would likely lead to the emergence of better performance."

2.6.1 Project Attractiveness

The OSS literature has emphasized the importance of attracting users and developers to keep a project active and successful (Arakji and Lang 2007; Koch 2004; Krishnamurthy 2002; Von Krogh et al. 2003). Developer motivation and participant interest has been suggested as one important factor for OSS success (Bonaccorsi and Rossi 2003). Some researchers also attribute OSS success to user interest (Subramaniam et al. 2009). Users, often serving as the observing "eye balls" to bugs (Raymond 1999), contribute to a project's success. Hence, it is important for an OSS project to attract both developers and users to be successful. Several studies attempted to identify what makes an OSS project favored by developers and users. Drivers of attractiveness include contributors' intrinsic and extrinsic motivations for joining OSS projects (Crowston and Scozzi 2002; Fang and Neufeld 2009; Hertel et al. 2003; Krishnamurthy 2006; Roberts et al. 2006), contextual factors of the project (Santos et al. 2013), visibility of the project, and the work activities performed toward software maintenance and improvement (Santos et al. 2013).

2.7 Sequence Stream Mining

Sequence mining aims to find statistically relevant patterns between data examples where the values are delivered in a sequence (Mabroukeh and Ezeife 2010). Common problems include pattern discovery, prediction, classification, clustering, the efficient building of an index for sequence data, and the comparison of sequence for similarity. Agrawal and Srikant (1995) made the first attempt to discover sequential patterns in transactions, proposing two algorithms AprioriSome and AprioriAll. Some important applications in sequential patterns include choice of patterns (closed vs. regular) (Chang et al. 2008), and efficient algorithms which include exact methods and approximate methods. Several algorithms were developed in order to efficiently mine frequent sequential patterns, such as SPADE (Zaki 2001), GSP (Srikant and Agrawal 1996), SPAM (Ayres et al. 2002), and PreFixSpan (Han et al. 2001). Some common algorithms to mine closed sequential patterns include CloSpan (Yan et al. 2003) and BIDE (Wang and Han 2004).

A Markov model is a stochastic model that can be used to model a random system that changes states according to a transition rule that depends solely on the current state. In particular, a hidden Markov model (HMM) is a type of Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. As a commonly-used stream mining technique, HMM has been used to detect changes in the sequence patterns (Rabiner and Juang 1986). Common applications of the hidden Markov model include speech recognition, artificial intelligence, pattern recognition, and bioinformatics. More recently, HMM has been used to detect anomalies in behavior. For example, Joshi and Phoha (2005) applied the HMM method to anomaly detection in network traffic. Cho and Park (2003) used HMM to build an intrusion detection system, while Ourston et al. (2003) applied HMM to detect multi-stage network attacks. Hoang et al. (2003) applied HMM to process sequences of system calls for anomaly detection. Additionally, human behaviors are modeled by HMM in studies. Lane (1999) presents a method for human behavior modeling in the computer security domain. HMM models of normal human behaviors from genuine users are built and then any deviation from the model will be considered a potential attack. Later on, Srivastava, Kundu et al. (2008) applied HMM to detect credit card fraud. Customers' previous transaction history was obtained as a training set to construct an HMM model on the customer's spending profile. Any new transactions are added to the observation sequence of the old transaction behavior sequence to calculate the probability using the constructed model. If the probability is lower than the old one, the new transaction might be fraudulent.

This dissertation aims to investigate the change in the development behavior, which is captured by the sequence of development events. Therefore, I reviewed studies that have analyzed the change of sequence patterns in time series data. There have been a limited number of studies looking into variations on a single pattern over a data stream. Silberschatz and Tuzhilin (1996) proposed a method to measure the "interestingness" of patterns to monitor variation. Both Agrawal and Psaila (1995) and Chakrabarti et al. (1998) measure variation of a pattern. However, these studies only analyze change over a single pattern. Ganti et al. (2002a) proposed a framework FOCUS, in which they generalize a method to compute the deviation among a series of same data mining models (decision tree models, frequent item-set models) to detect changes among datasets.

3 Discovery and Diagnosis of Behavioral Transitions in Patient Event Streams

3.1 Introduction

In this study, I demonstrate how various data mining techniques, as components of EventMiner, are applied to discover and diagnose behavioral transitions in patient event streams.

More than one million adults in the U.S. are diagnosed each year with cognitive impairments (CI) due to neurological disease or trauma (e.g., traumatic brain injury, stroke, tumor, epilepsy, infectious disease). Currently, there are between 13.3 to 16.1 million Americans living with chronic brain disorders and associated CI (Alliance 2001). In addition, approximately 4 million Americans have developmental disabilities that impact cognitive functioning (Services 2002). Cognitive impairments prevent this large and growing segment of our society from fully integrating into society; they are unable to participate in mainstream computer-based activities (Mccoll et al. 1998).

Clinics provide assistive technology (AT) to help with cognitive rehabilitation. However, studies have found that AT systems are abandoned by CI users at shockingly high rates (De Joode et al. 2010; Lopresti et al. 2004; Wilson et al. 2001; Wright et al. 2001). One major cause of abandonment is an eventual misalignment with: (1) user goals and abilities, and (2) the functionality delivered by the system. To support the monitoring of this relationship between user goals and their satisfaction to the system, I developed EventMiner. It is a data-mining enabled, event analysis framework, for change detection in user behavior. Users are given an email system to aid in their activities of daily living (ADLs). At appropriate times, the system is adapted to meet the changing needs of the user. By monitoring a user's event stream, EventMiner can detect changes in user behavior that indicate that the system should be adapted.

Over the past 10 years, a multi-disciplinary group of cognitive psychologists, computer scientists, and clinical workers have been successfully delivering AT to CI clients (Fickas et al. 2005; Sohlberg et al. 2003a; Sohlberg et al. 2002; Sohlberg et al. 2003b; Sohlberg et al. 2005a; Sohlberg et al. 2005b; Sohlberg and Mateer 1989; Sohlberg and Mateer 2001; Sutcliffe et al. 2006; Sutcliffe et al. 2003; Todis et al. 2005). As part of the *Think and Link* (TAL) project, which developed an email AT for cognitive rehabilitation, EventMiner serves as the tool to provide real-time data analysis. By combining and extending stream mining techniques, I develop EventMiner as an AT-monitoring software system with ever-increasing functionality. This article summarizes unique data mining aspects of EventMiner as a monitoring system.

This research provides two significant contributions to real-time data analytics:

(1) Automated recognition of changes in user behavior, where a user's behavior is defined by the stream of events that they initiate with the AT. The tool can automatically detect change by differencing models such as decision tree models and Hidden Markov models.

(2) Automated diagnosis of a user's behavioral change, as characterized by the most influential behavioral differences around a moment of change.

This analytic technique that discovers transitions from routine behavior will be helpful in identifying potential problems in planned behavior. This study is an initial attempt to develop a framework that integrates and extends event stream mining techniques to achieve personalized user monitoring; it can generalize to monitoring voluminous streams of event data. Consider, for example, business processing in support of order fulfillment. The sequence of events generated with each business process represents the planned behavior of the organization. Similarly, a computer hacker generates a sequence of events, such as improper Logins, in an effort to fulfill the goal of a system break-in. In both cases, EventMiner helps detecting transitions from routine behavior and thus identifying potential problems. Although this framework is currently applied to AT monitoring, it is scalable to other contexts such as business processes monitoring, transaction monitoring to detect potential fraud, and software development processes monitoring (Robinson and Deng 2015). In Chapter 4 and 5, I will show its application to OSS event streams.

3.1.1 A Cognitive Rehabilitation Scenario

Assume that Don is learning to email his friend. His cognitive impairment impedes his progress. His TAL caregiver, Andrew, uses the CORE methodology to obtain two important items (Sohlberg et al. 2002): (1) Don's personal goals for using email, and (2) Don's existing skills for using email independently. Using this information, Andrew produces a user profile and a training plan that fit with both Don's current skills and his personal goals.

As a user operates TAL, events are logged and then analyzed in support of decision-making about deferred goals. Don's daily usage of the email system produces raw data. This data includes that which is generated from the email system itself, along with Andrew's input on training progress. Working backwards, Don has goals that are not satisfied currently. In the AT context, goal failure is often associated with a poor fit between the client's goals and the goals supported by the AT. By monitoring user goals TAL can responsively react to changes in goal satisfaction. Such changes are attributed to TAL's success.

This study thus focuses on raising alerts when patients like Don need help and thus when the system needs to adapt. Rather than looking for specific events, the monitor looks for significant variations from historically normal behavior.

3.1.2 Goal Attainment Scales

The cognitive rehabilitation field uses a goal attainment scale to specify the individual goals and desires of a person. Each goal is broken into a set of attainment levels to provide a measure of attainment. Using this style, each user is asked to first list a goal and then five levels of attainment, ranging from not-attained to fully-attained. For example, Don's goals is to be socially involved through online communications. One of the subgoals was to learn to email with no help. He divided this goal into five levels: (1) level 1 (not attained): will not be able to learn how to use email; (2) level 2: can email, but only with lots of prompting and help; (3) can email, with some prompting and help; (4) level 4: can email with no prompting and help, and (5) level 5 (fully attained): can teach others how to email. Figure 3 illustrates Don's formalized emailing goals. Don wants to use email to *engage in online social communication*. This need is shown as the root goal in Figure 3. Supporting emailing subgoals are shown below the root.

Clinicians want to see goal satisfaction, and in particular: (1) a good success-to-failure ratio over sessions, and (2) a constant or improving trend of this ratio. In the case of Don, who is just acquiring simple email skills, clinicians want to see Don succeed with: (1) *read email* and (2) *compose and send email*. In support of clinicians, the monitoring system needs to: (1) recognize changes in user behavior in using the email AT, and (2) diagnose each significant change by characterizing by the most influential attributes of in AT usage, These two monitoring goals are demonstrated for Don's two email skills in the case study in section 3.4.

3.1.3 Approach

The approach to the monitoring of behavior is summarized as follows:

(1) A patient uses target software in the context of learning higher-level goals.

(2) The monitor builds models of the user's behaviors and changes in the user's behaviors. These models are structured according to the events generated by the software, such as composing an email message.

(3) A human analyst interprets the generated models to determine the changes in user learning and goal attainment.

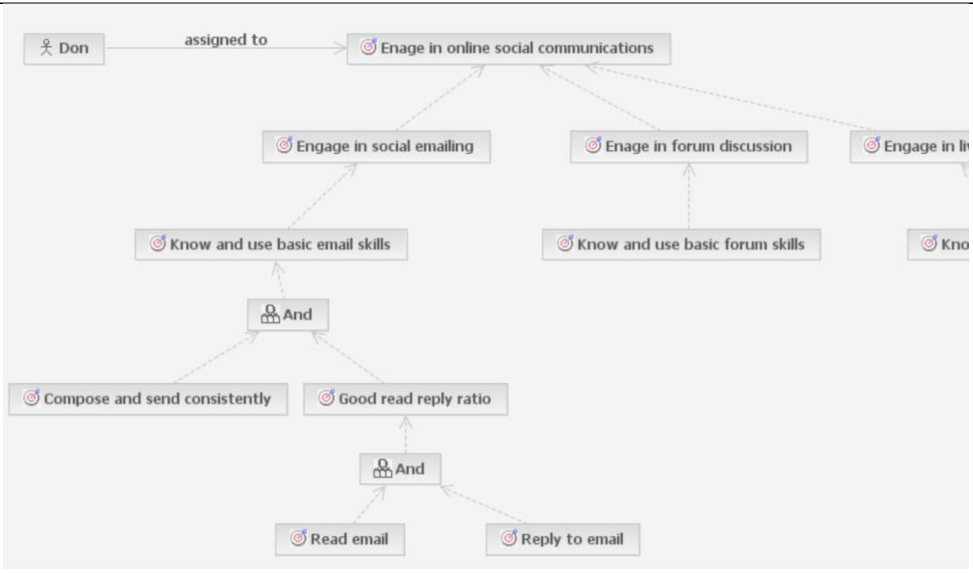

Don
assigned to
Enage in online social communications
Engage in social emailing
Enage in forum discussion
Engage in li
Know and use basic email skills
Know and use basic forum skills
Kno
And
Compose and send consistently
Good read reply ratio
And
Read email
Reply to email

Figure 3. Some of Don's Emailing Goals

The automation of the first two steps, achieved by EventMiner, dramatically reduces the monitoring effort on the post-clinical team. The monitor provides notification and characterization of transitions, vastly simplifying the work of the caregivers. The contribution of this research is the automated characterization of behavioral sequences as potential behavioral transitions, which is a prerequisite to the interpretation task.

Figure 4 presents the kind of analysis automated by EventMiner. Consider the line graphs as a representation of consistent behavior. The sharp dips in the graph are the significant points of interest. They suggest that the user substantially changed his or her behavior. My automated analysis reveals these potential goal transitions, and the automated diagnostic technique presents the behavioral differences. For example, there is a potential transition around week 10, and the change in behavior is a 30 percent reduction in sending email.

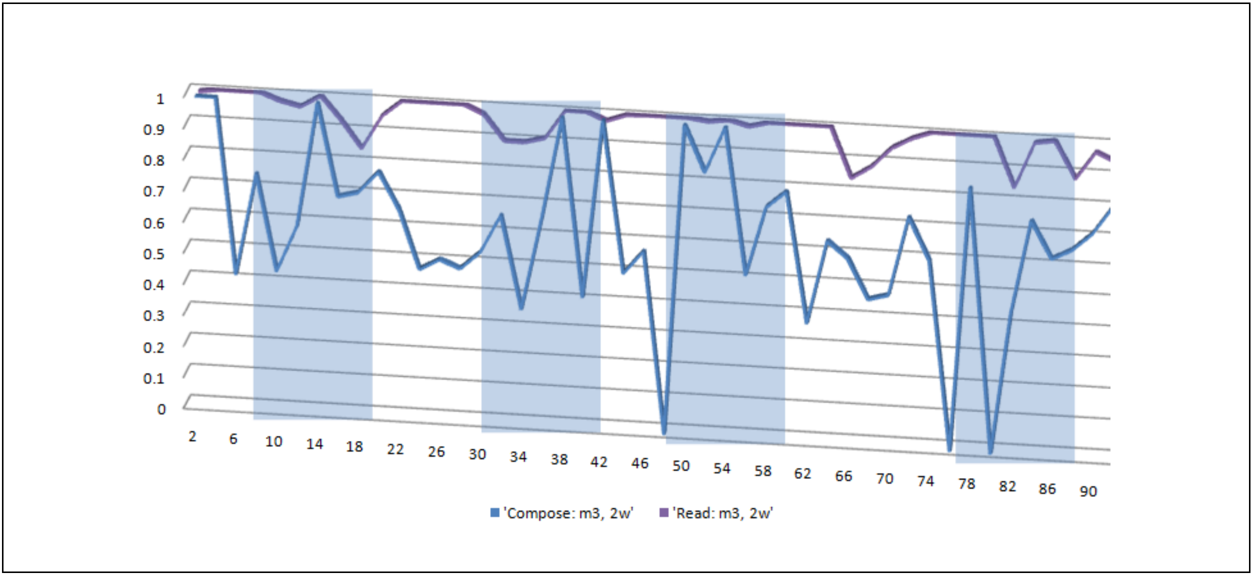


Figure 4. Quality of Stream-mined Models for Read and Compose Email, over 2 Years of Data Using 2-week Windows; Potential Goal Transitions Shaded

3.1.4 Essay Overview

This essay introduces the approach to the problem of monitoring individuals with cognitive impairments. Related research are presented in 3.2 followed by the design of the monitor (3.3) which is part of the EventMiner framework. A case study (3.4) precedes the conclusion (3.6).

3.2 Related Research

This study applies process mining techniques to AT monitoring. Research in process mining is introduced in 2.7. Here, I will focus on literature in data stream mining in particular, and model-based monitoring.

3.2.1 Data Stream Mining

Data stream systems analyze voluminous, continuous data streams where it is not practical to store all the data. Instead, sequential data subsets, called windows, are analyzed as they arrive. Consequently, there is an inherit tradeoff between accuracy (which requires all data) and timeliness (which dictates continuous updates). Stream mining aims to find interesting relationships over a sequence of data segments (Gaber et al. 2005; Gama 2010; Gama et al. 2009). A variety of techniques can be applied to stream data (Aggarwal et al. 2004; Ferrer-Troyano et al. 2004; Hulten et al. 2001; Last 2002)–much of the work is focused on the efficiency of incrementally updating the model (Domingos and Hulten 2000). Phua et al. (2007) address the issue of recognizing *spikes* in the data stream.

Detecting changes in data-streams is important for monitoring, in particular for AT monitoring systems. Two types of algorithms are common: (1) distribution detection, which watches for changes in the data distributions, and (2) burst detection, which watches for sudden large and unusual changes in a data-stream. Distribution detection algorithms have two common forms: (1) data from two windows (current and reference) are compared using some distance measure, (2) a predictive model is created from a prior window and then its prediction is compared with the current window—high prediction error indicates a significant change.

Many data stream techniques address change detection of item sets (Aggarwal et al. 2003; Agrawal and Psaila 1995; Chakrabarti et al. 1998; Ganti et al. 2002a; Ganti et al. 2002b; Kifer et al. 2004; Silberschatz and Tuzhilin 1996). For example, Agrawal and Psaila (1995) and Chakrabarti et al. (1998) monitor the change on the support of an item set over temporally ordered transactions. Ganti et al. (2002a) propose a framework, FOCUS, in which changes are detected by quantifying the difference between the two models induced by the datasets. Two trees are first extended so that they become identical, and then they are compared to derive a numeric difference value. The differencing approach in this study is similar to FOCUS. However, I efficiently derive the first *n* attributes ordered by their contribution to the difference (Robinson et al. 2011a). This approach allows us to not only detect changes, but also discover the nature of these changes such as what attributes (decision rules) impact the change and how the degree of attribute impact changes over time. Such information can give an analyst unique insight into the causes of changes.

3.2.2 Model-Based Monitoring in AT

Work on model-based monitoring in the context of assisted living includes (1) the use biometric and sensor data from home activity to identify trends and drifts from those trends (Jain et al. 2006), and (2) the use of positional data to detect behavioral deviations from routine patterns (Virone et al. 2008). This work differs in that I am concerned with the attainment or denial of goals—the long-term trend is less important than identifying departures from recent trends. Clinicians need to be notified when a client appears to transition to satisfying (or failing) an AT-based goal.

3.3 Monitor Design and Development

The monitor is designed to provide feedback to clinicians about AT usage by clients with CI. This monitor needs to: (1) identify transitions in user behavior, where a user's behavior is defined by the stream of events that they initiate with the AT; and (2) diagnose transitions of a user's behavioral change, as characterized by the most influential behavioral differences around a moment of change. Table 2 presents design rationale for the monitor. Two strategic decisions are: how to recognize transitions, and how to diagnosis transitions. These two choices are intertwined. I chose to use error rate to identify transitions and model differencing for diagnoses.

3.3.1 TAL Data Stream

The TAL email client provides an automated custom logger. To obtain real-time data access, a log file can be monitored. Here is a simplified entry from the log.

```
09:48:41 NewMailEvent [id=765406159;in-reply-to=311149530;chars=770;words=179;sentences=16]
```

This logged event specifies the time, the program event, and its associated arguments. The example logs the arrival of a new email that is in reply to previous e-mail; the identity of the sender and receiver and characteristics of the e-mail message, such as its length, are also included. The significant event types are: read email, compose email, delete email, and new (arriving) email. A database view provides a continuously updated stream. The dataset for one client, Don, included 3,695,086 records occupying 737 MB in Microsoft SQL Server 2005.

	How to diagnosis transitions	**Difference two models to find the rank-ordered changed attributes.**	**Ordered trees, like decision trees, simplify computation.**
Tactical	Which mining algorithm(s) to apply	Decision tree	Differencing, attribute ordering, and explanation is simpler using decision tree than most other models.
	How to select data window size	Sufficient data for good accuracy while being timely led us to 2-weeks.	Larger windows can increase accuracy but miss transitions.
	Which data attributes to mine	Predicting event type is the focus. Attributes that improved that prediction were included.	Some logged data was dropped because had no predictive value, which may be common for DI data streams.

Table 2. Design Rationale for the Design of the Monitor

3.3.2 Identify Transitions

The approach to the analysis of user behavior has been to extend classic data mining techniques. Two extensions are important. First, logged events are processed with stream-mining methods. Second, changes in mined-model qualities are considered an indicator of changes in user behavior. Overall, the approach automatically models user behavior based on events and recognizes significant changes in those events.

This approach depends on a few assumptions:

(1) User behavior is characterized by the stream of events that user initiates with the user interface.

(2) User behavior can be characterized as planned event sequences for goal fulfillment.

(3) User behavior is less consistent when involved in less familiar tasks, and more consistent when involved in more familiar, routinized tasks.

Based on these assumptions, the monitor analyzes patterns in event sequences to find inconsistent behaviors, which are interpreted as *goal transitions*, i.e., moving from a state of low goal achievement to higher goal achievement (or vice versa). Consider a user goal set, $\mathbf{G}_i$, which specifies the currently attained goals. I want to be notified when a user transitions to new goal set, $\mathbf{G}_j$, where the difference between the user goal attainment is $\mathbf{G}_j - \mathbf{G}_i = g$. EventMiner applies a data-stream mining approach to identify unusual behavioral patterns and specific metrics to select those most likely to be associated with goal transitions.

3.3.2.1 Model Changes as Behavioral Changes

EventMiner applies a predictive model approach to distribution detection to identify changes in the data stream. Stream mining produces a sequence of models, m_1 .. m_n that predict the behavior observed in windows w_2 .. w_{n+1}. After a predicted window is observed, w_i, the prediction quality of its model, $q(m_{i-1},w_i) = [0,1]$, can be calculated. For example, the classic metrics of *accuracy* and *precision* can be used individually or in combination.

Accuracy measures how error-free the model's predictions are, according to this equation: *accuracy* = (true negative cases + true positive cases) / all cases where all cases = true negative + true positive + false negative + false positive cases

Precision measures fidelity, according to this equation:

precision = true positive cases / (true positive + false positive cases)

Predictive quality can be automatically evaluated with each new window during stream mining. Given that model m_i is trained over window w_i, one can evaluate the predicted values of m_i against the known data in w_{i+1}.

Consider the case where the predictive quality is nearly a constant 0.9. This suggests that the models are good and that the behavior from one window to the next is nearly constant. Now, consider a sequence of some n predictions, with $q_1 .. q_n$, where each $q_i \approx 0.9$ with the exception of q_k ($1 < k < n$) which is 0.1. This suggests that the models are good with the exception of m_{k-1}, which was trained on window w_{k-1} to predict window w_k. I infer that something interesting happened during window w_k. That is, the events in window w_k are so different from the events of window w_{k-1} that the model trained on w_{k-1} cannot reasonably classify the new window w_k behavior.

The predictive quality change is dq/dt. Thus, $|q'|>\varepsilon$ implies behavioral changes from window w_{k-1} to window w_k. We can also consider how quickly the predictive quality changes, which is q''. An analysis of typical domain values for q' and q'' can provide guidelines that distinguish normal behavioral variations from significant behavioral changes (Robinson and Akhlaghi 2010).

3.3.2.2 Good Model Quality is Sufficient

A good model is sufficient for finding significant differences in a model sequence. Great or nearly perfect predictive models are wonderful, but unnecessary for identifying transitions. The model differences are more important. Low quality models will have substantial variance from model to model in stream mining. Good quality models have less variance and the differences will be striking. The key is that the differences are more prominent than the random variance in the models. In the TAL domain, the goal is better than 70% for both accuracy and precision.

There are many ways to improve model quality, such as selecting the best mining technique, apply multiple techniques simultaneously, and apply multiple window sizes simultaneously.

Selecting the appropriate window size, w_s, is important. If w_s is small, then insufficient data will be available when the mining model is derived. Conversely, if w_s is large, then the analyst must wait, perhaps a long time, before the model is derived—moreover, model construction itself can take a long time. Finally, large data windows have a *regression to the mean problem*—short variations in behavior will be discounted in favor of more common behavior, and thus short variations may not be represented in the mined model. Thus, widow size affects precision, accuracy, and availability of the mined model.

In the case study of section 3.4, I applied: (1) multiple concurrent windows, (2) multiple models, and (3) decision trees because: (1) their pre-testing accuracy was better than Bayesian networks, neural networks, and association rules; and (2) decision trees simplify transition diagnosis, which I describe next.

3.3.3 Diagnosing Transitions

A discovered transition change begs the question, "what exactly has changed?" Diagnosing a transition provides an answer, which is used by clinicians to decide if a goal transition has occurred. In the TAL diagnosis context, it can reveal that the user "now sends more emails on Wednesday than in the past." (Failing to take medications on Wednesday was an underlying cause for this real example.)

Model differencing reveals the most significant changes between two models. A data mined model, m_i, provides a model of the data observed in window w_i. When a model quality falls substantially from w_{k-1} to window w_k, it means that the data varies substantially from w_{k-1} to window w_k. Thus, given a window w_k with $|q'|>\varepsilon$, three data windows are of interest: w_{k-1} (before change data), w_k (change data), and w_{k+1} (after change data). To diagnose persistent change, I compare the models associated with windows w_{k-1} with w_{k+1}. I consider w_k (change data) of lesser importance because it represents the transition from between two models of more consistent behavior. Next, I illustrate (decision tree) model comparisons.

3.3.3.1 Decision Tree Differencing

Recall the TAL dataset from section 3.3.1. Figure 5 illustrates a portion of a decision tree that models user email events. For a data window, the model summarizes the kind of email event (Event: read, receive, delete, compose), the day of the week (weekday: 1-7), and a 2-hour period in which the event occurred (DaySegment: 0 - 11). The leaves of the tree represent the email events. A path from the root to a leaf can be considered a query (or proposition) that characterizes the leaf data. For example, the path DaySegment = 4, weekday = 1 leads to the events satisfying those two attribute values. In Figure 5, the data distribution is represented by the colored bar chart on the leaves. Thus, the bottom left leaf shows that the path DaySegment = 4, weekday = 1 has only read email events. On the other hand, other leaves have a mix of colors, indicating a mixed distribution of email events. (The node is labeled with the dominant event number—read = 1, compose = 5.)

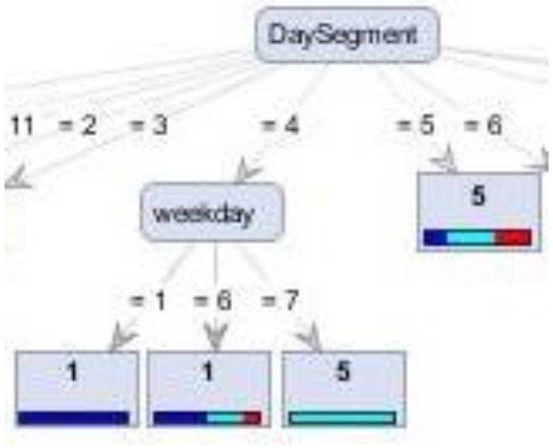


Figure 5. A Decision Tree Classifying Compose Events

To illustrate model differencing, consider two hypothetical decision trees. The first has only read events for day segment 1, while the second has only read events for day segment 2. The difference of the two models (m_2 - m_1) reveals that read has shifted from DaySegment 1 to

DaySegment 2 (i.e. DaySegment 1 is dropped and DaySegment 2 is added). This observation is the basis for THE diagnosis. By differencing two decision trees, one can characterize the change. The algorithm is sufficiently efficient because it stops on each branch comparison when it finds the first difference (or leaves) (Robinson et al. 2011b).

3.3.3.2 Significant Attributes in Decision Trees

In general, completely comparing unordered trees is computationally expensive (NP-complete) (Bille 2005). Even if we assume ordered trees, which is common for decision trees, the algorithmic complexity of complete comparison is high. The complexity arises because a node in tree A may move to anywhere within the next tree B (however unlikely in practice). In practice, I have found that comparing sequential decision trees reveal small changes in the tree, such as the attribute or attribute value changes. Thus, the entire tree must be exhaustively searched. Even so, complete comparison is practical for small decision trees. Yet, I do not need a complete comparison for diagnostic problem. Instead, an algorithm that returns the most influential changes is sufficient.

Decision trees use an attribute selection measure as a criterion for splitting at a node. Popular choices include information gain, Gini impurity measure, and gain ratio. The attribute with the highest metric determines the splitting rule. The improvement at each node is calculated using the selection measure and fraction of data split. For each attribute, the importance measure is derived using a weighted sum of the improvements due to that attribute. Such importance values are used to rank the decision tree attributes.

More influential attributes are found at the top of the tree, while less influential attributes are found closer to the leaves (Wu et al. 2008). Thus, when comparing two trees, the most significant differences are found by first comparing the roots and working towards the leaves. I apply this method, stopping along each branch when either a difference or a leaf-node is found (Robinson et al. 2011b).

By comparing structures of two trees based on the order in which attributes appear near the top, I can determine the differences in attribute influence between two models. This ordering of attributes by significance simplifies comparison and is one reason why we use decision trees to model behavior.

Given two decision trees, there are two kinds of differences to consider: (1) value differences at the nodes, and (2) attribute difference. Additionally, it is possible that the two models have nothing in common. These issues are considered next.

3.3.3.3 Model Differences in Attribute Values

Given two decision tree models, it is possible that they have an attribute value difference. Consider the models in Figure 6 and Figure 7. The two trees branch on day segment. The absence of a branch for DaySegment =2 in Figure 7 is the only structural difference.

Dropping a tree branch, from m_1 to m_2, occurs when the latter model is constructed from a dataset lacking data for the branch. In the case of Figure 7 compared to the prior model of Figure 6, the user quit reading email during DaySegment =2.

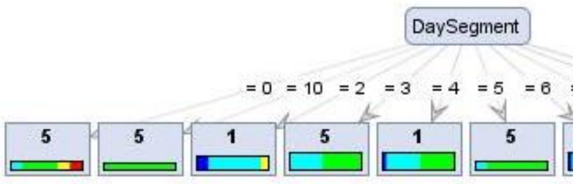


Figure 6. A Decision Tree that Classifies DaySegment =2

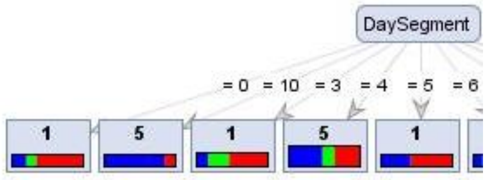


Figure 7. A Decision Tree that Lacks DaySegment =2

3.3.3.4 Model Differences in Attributes

Given two decision tree models, it is possible that they have an attribute difference. As an illustration, compare Figure 6 with the previous Figure 5. The attribute difference occurs in the path DaySegment =4. In Figure 6, there is no subsequent branching, whereas in Figure 5 DaySegment =4 is segmented into three weekday branches (1, 6, and 7).

Adding an attribute, from m_1 to m_2, occurs when the latter model is constructed from a dataset that includes data for the branches (e.g., weekdays 1, 6, and 7), and that data is non-uniformly distributed over the attribute (weekday). Thus, we can infer from Figure 5 that the user, during DaySegment =4, mostly had email events on weekdays 1, 6, and 7.

We cannot infer specific quantity changes from attribute differences alone — the addition of weekday in Figure 5 does not suggest that more events occurred during weekdays 1, 6, and 7. Instead, it shows that the distribution of events changed from Figure 6 to Figure 5. It is likely, however, that fewer events occurred for the weekdays other than 1, 6, and 7, thereby creating a non-uniform distribution.

It is worth noting here the pathological case of two identical models, m_1 to m_2, for which the events occur with the same distribution, but different quantities. For example, consider Figure 6. There is no indication of the quantity of DaySegment =10; the figure simply indicates that at least one event occurred at DaySegment =10. It is possible, but not likely, that another model, identical to that of Figure 6 would have twice as many total events with the same distribution. Therefore, the decision-tree model differencing is a diagnostic aid, which can be improved by computing the classified quantities in each leaf. Once a tree difference is identified, our algorithm does compute the difference in data counts. However, analysis is not applied to identical trees, because the pathological case is exceedingly unlikely in practice.

3.3.3.5 Models with No Common Attributes

Given two decision tree models, it is possible that they have no attributes in common. For example, model m_1 branches only on DaySegment 1 and model m_2 branches only on weekday 1. From this example, we can deduce that the concentration of email events was first on DaySegment 1, uniformly distributed for all weekdays, while the second model concentration shifted to weekday 1 uniformly distributed on all hours. Such a dramatic change has not occurred in our data, but may occur in other domains. In such cases, my algorithm reports the entire two trees as the differences.

3.3.3.6 Summary of Approach

I summarize the transition and diagnosis approach as follows:

(1) The mined model provide statistical summary of the distribution of events.

(2) Model differencing reveals the top-most influential attributes changes over a period.

(3) The method is automated and as accurate as the mined-models, which are dependent on the quality of the data.

3.3.4 Development

EventMiner was developed in KNIME, an open source, Java-based platform. Multiple workflows were developed in KNIME to automate the monitoring process. KNIME allows me to integrate my own algorithms as nodes, as well as use specialized tools, like the sequence-mining R package, TraMineR. Figure 8 shows an example KNIME workflow we used for model differencing. A sample pseudo codes for tree differencing is presented in **Table A3** in the appendix.

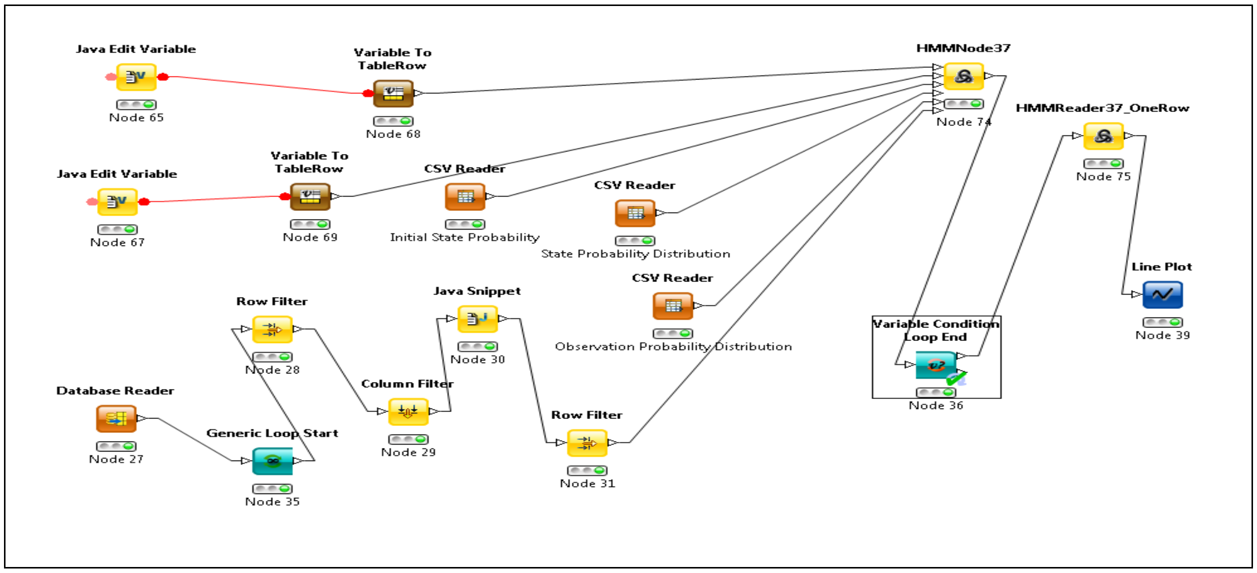


Figure 8. A KNIME Workflow Example

3.4 A Case Demonstration

In this study, I have applied a design science approach to the analysis of analytic techniques (Hevner et al. 2004). Having described the use of decision tree models in modeling event streams and identifying transitions, I will illustrate in a case study, on how I apply this approach to analyze the behavior of a randomly selected client from the TAL project. The data includes 3,695,086 email events collected from one single patient over two years.

3.4.1 Data Mining Models

Transition detection requires a good quality model. In pre-testing, a variety of classification models were considered, including decision trees, Bayesian networks, neural networks, and association rules. I chose three decision tree algorithms, which were the best in quality: (1) M1: Gain Ratio (C4.5) is a successor of ID3; (2) M2: Information Gain (ID3) minimizes the information needed to classify the data represented as tuples, the resulting partitions reflect the least randomness or impurity in these partitions; and (3) M3: Gini Index (CART) uses a formula based on probability to branch on nodes.

In pre-testing, I considered a variety of data window sizes. I decided to run three window sizes: two-, four-, and eight-week windows. These allowed me to consider short-term and long-term behavioral patterns. Shorter than two weeks produced poor quality models due to insufficient data, while longer than eight weeks was not valued for diagnoses.

3.4.2 Identifying Transitions

The decision tree models predict the event type (read email, or compose email) for a time-slot within a day-of-week. Table 3 summarizes accuracy and precision for read and compose event types. The values for the different window sizes are very similar. The read models are good — the models predict well the number of emails that will be read during a time-segment on a day-of-week. The compose model is not as good. It weakly predicts the number of emails that will be composed during a time-segment on a day-of-week. Because read models have the best quality and least variability, I rely on them to predict behavioral changes. The compose graph mostly supports the same transition points, but with less accuracy. (See Figure 4, which compares the two model qualities.)

		Read		**Compose**	
		Accuracy	**Precision**	**Accuracy**	**Precision**
2-week window	Average	0.967	0.970	0.631	0.484
	StdD.	0.049	0.018	0.241	0.144
4-week window	Average	0.984	0.971	0.657	0.478
	StdD.	0.027	0.011	0.202	0.119
8-week window	Average	0.983	0.971	0.657	0.478
	StdD.	0.027	0.011	0.202	0.119

Table 3. Qualities of Model M1 for Read and Compose Email

Consider the read model. The predictive qualities of the three data mined models (M1, M2, M3) for three windows are graphed in Figure 9. The graph shows that the models roughly track the same events. Windows of four and eight weeks are nearly the same. They both hide or diminish events considered interesting by the two-week window (e.g., weeks 66 – 68). In general, the two-week window analysis has greater variability and foreshadows the larger window analysis. Notice that week 34 has poor predictive quality (q) as identified in both the two- and four-week window analysis. Moreover, the quality in the immediately surrounding weeks is good; thus, the rate in change of the quality (q') around week 34 is also high. Together the dramatic decrease in q and zeroing of q' suggest that week 34 may be a behavioral inflection point worthy of further analysis.

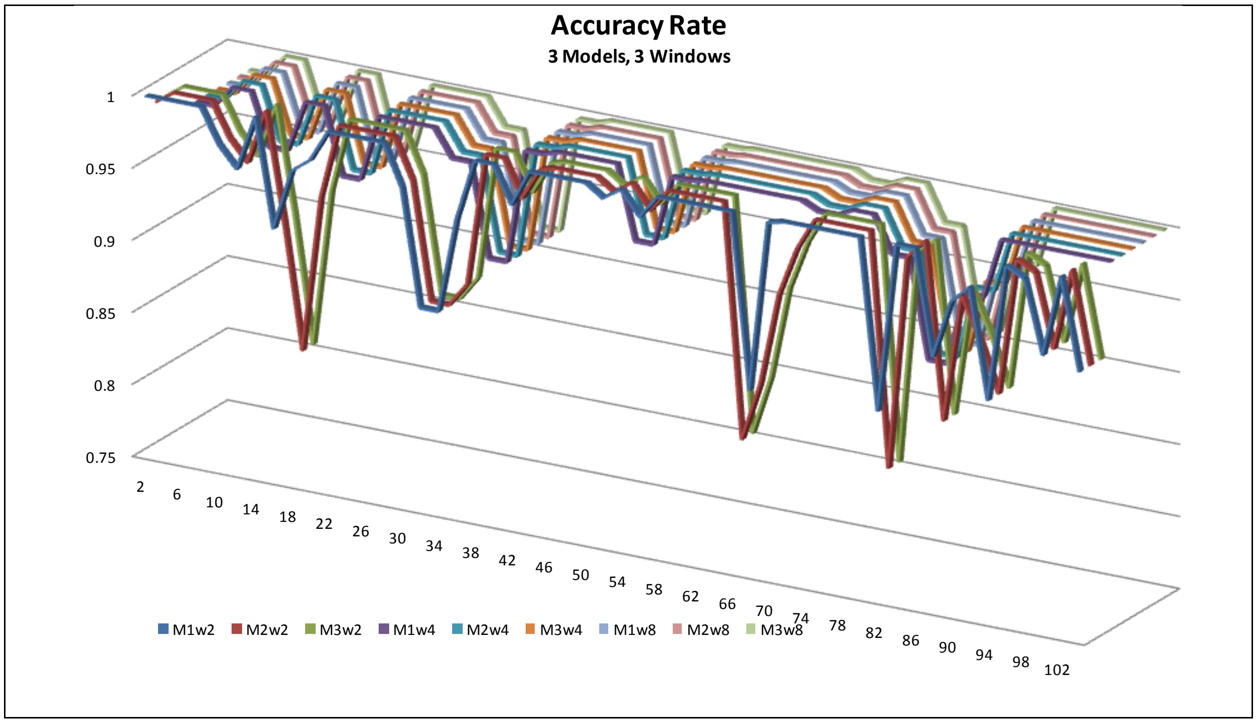


Figure 9. Predictive Quality for the Read Models

Table 4 presents *q*' for two- and four-week windows for the three models; it illustrates the significance of the transition identification technique. The table shows where *q*' turns negative between 0 and 36 weeks. The analyst can combine these values to automate the recognition of interesting events. The rule is that two consecutive *q*' windows means that the transition began on the first window. Table 4 shows that week 12 and 32 are potential transitions.

	2-week window			4-week window		
Week	M1	M2	M3	M1	M2	M3
10	-0.012	-0.012	-0.012	0.000	0.000	0.000
12	-0.007	-0.007	-0.007	-0.018	-0.018	-0.018
16	-0.037	-0.037	-0.037	0.018	0.018	0.018
18	0.021	-0.043	-0.043	0.000	0.000	0.000
20	0.005	0.054	0.054	-0.023	-0.023	-0.023
…						
30	-0.014	-0.014	-0.014	0.000	0.000	0.000
32	-0.040	-0.040	-0.040	-0.010	-0.010	-0.010
36	0.033	0.009	0.009	-0.032	-0.032	-0.032

Table 4. First Derivative of Accuracy, *q*'.

3.4.3 Diagnosing Transitions

As transitions are identified, diagnosis is automatically applied. For example, if a transition is identified in the read model, then diagnosis will be applied to the read model as well as other models such as the *compose email* and *delete email* models. This is based on the assumption that a significant change in any usage of the AT may signal a transition (however slight) for any of the monitored goals. Thus, identified goal transitions are considered signs of behavioral changes that may be reflected across the AT system usage.

Table 5 and Table 6 summarize the diagnostic analysis identified in the *read email* and *compose and send email* data streams underling Figure 9. Here I present the results of comparing pairs of M3 decision trees over 2-week windows. The results are similar for the other models and windows.

Table 5 presents the results of model differencing the inflection points found in the *compose and send email* data stream. One potential goal transition occurs during the two weeks of 32 and 33 of the dataset. The three data windows for this inflection point begin at 32, 34, and 36 weeks.

The model differencing technique compares the models beginning at 32 and 36—it does not consider the intervening transitional model. For the transition at week 32, there was a 15% increase is email composition during the periods from 6 to 12 A.M. and 2 to 8 P.M. There were 48% and 9% increases for the inflection points of beginning at weeks 50 and 80, respectively. These changes indicate an increased attainment for the compose email goal of Figure 3.

Weeks	Compose in Selected Hours	Total Compose for Model	% Activity	Δ Activity
10 PM				
10 - 12	7	111	6%	
14 - 16	0	115	0%	-6%
Tuesday (6-12 AM and 2-8 PM)				
32-34	3	46	6%	
36-38	14	67	21%	15%
Saturday				
50-52	0	55	0%	
54-56	39	81	48%	48%
Saturday, Sunday, Monday and Friday (6-12 AM)				
80-82	5	31	16%	
84-86	23	91	25%	9%

Table 5. Significant Changes in Composing Email

Weeks	Read in Selected Hours	Total Read for Model	% Activity	Δ Activity
10 PM				
10 - 12	6	290	2%	
14 - 16	0	365	0%	-2%
12 AM - 2 AM				
32-34	0	146	0%	
36-38	2	162	1.23%	1.23%
Sunday				
50-52	0	203	0%	
54-56	57	272	21%	21%
10 PM – 2 AM				
80-82	0	121	0%	
84-86	13	442	3%	3%

Table 6. Significant Changes in Reading Email

Figure 10 presents a stack graph showing the differences in read count, with the top-5 buddies, for weeks 50 – 56. The graph shows that some buddy increases and decreases occurred in the transition week 52, followed by general increases in week 54. The graph reveals the increase in the number of reads, which was identified in the metrics of Table 6. This illustrates the value of the metrics — rather than generating and reviewing numerous graphs, the metrics point to windows where the data and their graphs have the most change. These changes indicate an increased attainment for the read email goal of Figure 3. Table 6 presents the results of the model differencing for inflection points found in reading email. For the transitions beginning at weeks 32, 50, and 80 the increases were 1.2%, 21%, and 3%, respectively.

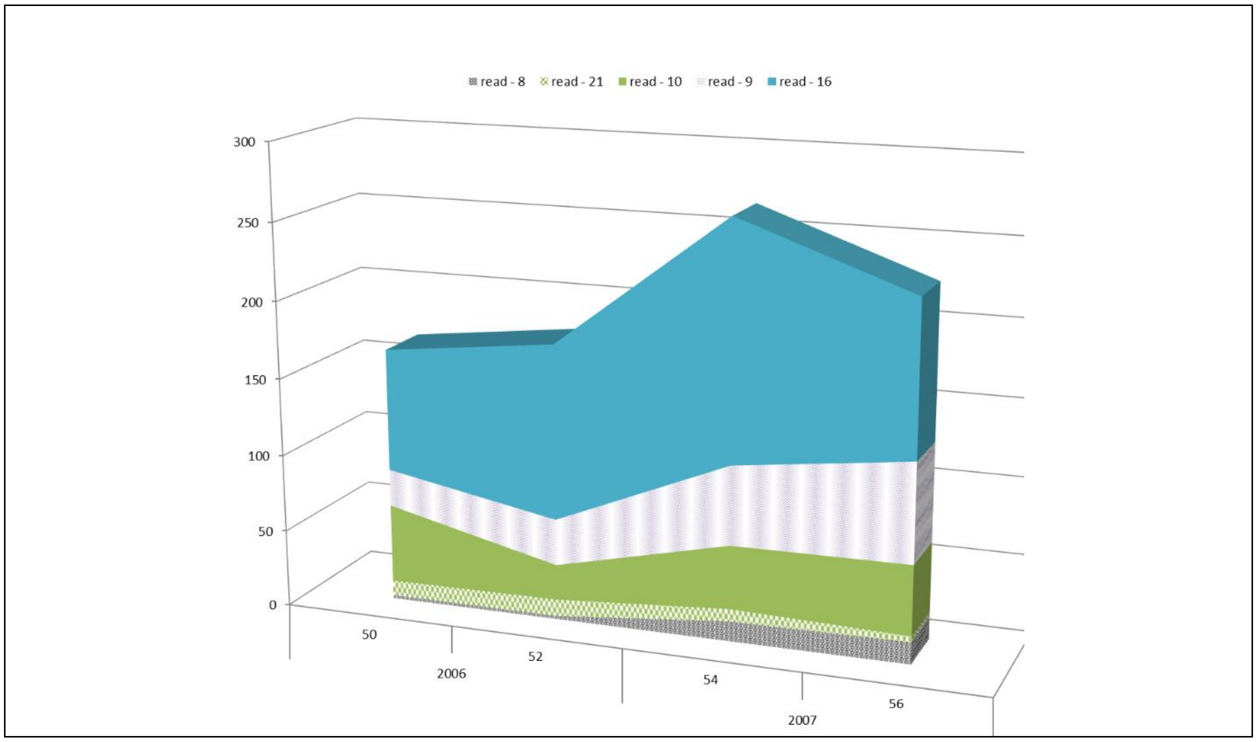


Figure 10. Stack Graph of the Read Counts, by Buddy, Weeks 50 – 56

In summary, the preceding tables summarize how model differencing applies. They demonstrate how transitions can be characterized by their underlying behavioral changes. It should be noted that, for this presentation, consecutive time segments were collapsed. The actual mined model considered 2-hour day segments. Thus, for example, the Saturday statistics of Table 5 is an aggregation of the 12 time-periods. The other statistics are automatically calculated (e.g., total and percent). Finally, it should be noted, that the results are calculated in real-time, as each window closes, immediately after the calculation of goal transitions.

3.4.4 Evaluation

The research evaluation considers two concerns, introduced at the beginning of this article:

(1) Can the EventMiner framework provide automated recognition of changes in user behavior, where a user's behavior is defined by the stream of events?

(2) Can the framework provide automated diagnosis of a user's behavioral change, as characterized by the most prominent behavioral differences around a moment of change?

Through software construction, testing, experimentation, and case study, I affirm both propositions. Next, I will discuss two more issues: (1) model quality and (2) validation by real-world events.

3.4.4.1 Accuracy and Precision

Accuracy and precision are standard metrics for determining predictive model quality. As Table 5 and Table 6 show, the read model is very good and while the compose model is not as good. This may be an anticipated because the read model depends on received emails and free time of the client; these elements are mostly routinized for the user population. On the other hand, email composition depends on the client's skills and interest in communicating. In a prior study, we showed that client interest and email composition increased with the addition of new *email buddies*, while decreasing slowly thereafter (Fickas et al. 2005). Therefore, given the limited information in the data and the real-world behavioral variations of the users, the availability of at least one very good model appears to provide adequate information to identify some significant behavioral changes.

3.4.4.2 Real-World Events

Changes in model quality reveal real changes in user behavior. This case study illustrates this with the inflection points that Table 5 and Table 6 summarize. There is a causality chain from the user's manipulation of the AT to the diagnosis of goal transitions:

(1) A user exercises the AT interface, such as reading and composing email.

(2) Data mined models are generalized from and accordance with the distribution of the events.

(3) Model differences are calculated, revealing changes in the models, which reflect changes in the event distribution, and thus changes in the user behavior.

(4) Model differences, at goal transitions, are characterized according to the events observed.

Thus, assuming the algorithms and software are correct, there is a direct causal chain from changes in usage of the AT and the diagnosed changes presented to clinicians.

Discussion on the monitored analysis with other TAL researchers reveals that the goal transitions do seem to reflect persistent changes in behavior. For example, week 34 of the data corresponds to 8/20/2006 - 8/26/2006, while week 82 of the data corresponds to 7/29/2007 - 8/4/2007 (week 31 of 2007). I hypothesize that something interesting happens to the client in the August summer holiday, such as a family member visit. Client anonymity prevented us from directly correlating such real-world events, but it has been intimated that such events have occurred.

Don's event data shows a general trend of increasing email usage. He consistently composes email and replies to emails, which is in support of his *know and use basic email skills* (Figure 3). Don also has periodic behavior of increasing emails after the introduction of a new buddy

(Fickas et al. 2005). A recent analysis of Don's sequences (e.g., read followed by compose), reveals that Don is becoming (1) more consistent over time because of the increasing length of non-variable sequences, and (2) becoming more conversational with email because of the increased usage of reply (Robinson et al. 2012). This latter point is significant, because it suggests that Don may be transitioning from using email for simple notifications or requests to dialogs.

3.5 Monitoring with Hidden Markov Models

One disadvantage of the decision tree differencing technique is that it cannot directly recognize changes in sequence distributions. Decision tree differencing monitoring is good for strict compliance checking; however, it ignores unmatched sequences, and thus is of limited use where there is a great variety of sequential patterns. Therefore, we extend EventMiner by adding another component: a hidden-Markov-model (HMM) based monitor. This component applies a HMM approach to complement the existing decision tree differencing component. A hidden Markov model is a stochastic signal model (Rabiner 1989) that commonly used for pattern recognition and anomaly detection. As introduced in 2.7, several studies have applied HMM for anomaly detection.

This new component is similar to the decision tree differencing monitor: windowed models are differenced to identify significant transitions in both methods. The difference is that, hidden Markov models are used to characterize user behavior. This HMM-based method is also applied to the usage data of a randomly selected client (Robinson et al. 2013b). In this work, we detected behavioral changes in the usage behavior. Furthermore, building on a Transtheortical theory perspective (Prochaska et al. 1997), we explained the identified transitions as normal learning and transitional learning states.

3.6 Conclusion

In this study, I describe and demonstrate how decision trees and hidden Markov models can be used to characterize software usage, look for unusual behaviors, and guide diagnosis of significant behavioral changes. By differencing the resulting sequence of generated models, this approach can identify transitions in software usage. The transition identification and diagnosis is automated by EventMiner.

This study contributes to both practitioners and researchers. It is important to the field of AT-based clinical therapy. Dynamically interpreting and adapting therapy plans for individuals currently requires substantial effort of clinicians. With the CI user population increasing, the need for some automation in therapy analysis is critical. For caregivers, when transitions are detected, they can provide assistance to ensure the client is not relapsing and encouragement to aid a progressing client. The tool and the analysis method provided will be a critical factor in addressing the needs of the millions of people with cognitive disabilities.

For researchers, the identification of behavioral transition points, similar to the concepts of "encounters" refereed by Robey and Newman (1996), can serve as a starting point for process theory building. Those identified transition points and recurring sequences of events can support different theoretical interpretations, by the choice of the researchers (Robey and Newman 1996, page 31). Process models, known for their faithful account of actual experiences, can become cumbersome and analytically complex (Kling 1987; Markus and Robey 1988). For researchers who are interested in building process models, this study provides the tool and method to automatically analyze stream data, for transition detection and pattern recognition. , greatly reducing the effort for data analysis. Researchers can then theoretically interpret the transition points, as a start pointing for theory building. In the following chapters, I will show how this framework can be applied to the OSS context.

4. OSS Development Behavior Transition Discovery

4.1 Introduction

Following the previous study in Chapter 3, this study applies EventMiner to an open Source Software (OSS) context. I have two major goals with this study: (1) to provide a project dashboard that can raise alerts when the OSS team appears to be losing its effectiveness, and (2) to develop a methodological and software framework to analyze OSS development process data. This framework can reduce process researchers' analysis effort and provide a "pictures of the process" (Markus and Robey 1988) by identifying transition points and behavioral patterns in the given data. This picture of process can be theoretically interpreted for building process models. Having already developed the data-mining framework EventMiner, I apply it to the team's repository event log to observe their activities and look for interesting transition points in the development processes. The approach of the monitoring framework EventMiner is summarized as follows:

(1) Developers use their standard team tools (e.g., Eclipse, GitHub, *etc*.) to develop software.

(2) EventMiner continually monitors the event log provided by the team's source code repository (e.g., GitHub).

(3) EventMiner builds models of the developer's behaviors and their changes by identifying transition points and recurring patterns. These models are structured according to the events generated by the repository events.

(4) Researchers can theoretically interpret those models of an individual project or multiple projects, generated in step (3), to build process theories on OSS development.

(5) Researchers can also develop models on the relationship between behavioral sequential patterns and project performance.

(6) Based on insights drawn from step (4) and step (5), developers can review the data analysis results for their own project's development history, and make inferences as to how successful the project will be.

In this study, I will describe the automation for steps 2 – 3 (model building). I will demonstrate how this process is applied to 103 projects from GitHub.com, an open source repository. Much research remains to extend our models. I also made an initial attempt to address step 5 in my third study in Chapter 5. I will address automation of steps 4 – 6 (model interpretation and forecasting) in the future, which will be discussed in 6.3. In the following, I introduce two major goals for this study in details.

4.1.1 Recognizing Behaviors and Change

This study aims to identify and analyze common sequences of actions by OSS developers. Several actions can be performed by developers in an OSS project, such as making a commit, raise an issue, and comment to an issue. I would like to know, for example, if the pattern sequence (Issue x, Comment x, Commit x) occurs frequently in the stream of events generated by the developer tools. I suspect that successful projects will apply this pattern more frequently than simply (Commit x)—that is, having no previously specified issue or comment for a code commit.

To uncover sequential patterns, sequential data mining techniques are commonly applied. When such techniques are applied directly to event streams generated by OSS developers, I found that it generates long lists of varied, low-level action sequences, which are not easily interpreted. To address this issue, I first specify work constructs, which are recognized by a rule-base system that integrate concepts from theory of distributed cognition. More generally, I am working towards a theory of OSS development by incrementally improving partial process models, which recognize theoretical constructs in OSS event streams.

The analysis proceeds as follows:

(1) Specify recognition rules for our theoretical constructs, such as an issue-based work unit.

(2) Apply the rules to the OSS event-stream data, to recognize theoretical constructs.

(3) Data mine the theoretical constructs, using sequence-mining techniques.

(4) Apply model-differencing techniques, to recognize changes in behaviors over time.

This approach allows me to analyze OSS developer behavior (as action sequences) and their changes over time.

4.1.2 Building a Process Theory

This work on analyzing OSS projects represents a study in the efforts toward supporting the construction of process theories. A process model "explains development in terms of the order in which things occur and the stage in the process at which they occur" (Van De Ven 2007). Abbott, for example, illustrates how time ordered events affect the lifecycle of individuals, which supports theorizing about process steps, and cause-effect relationships (Abbott 1990). The OSS study presented herein illustrates this approach to abducting theory elements from sequence data. I start with simple sequence-based constructs, such as higher-level development workflows that are comprised of lower-level development activities (raising an issue, making a commit, etc.). I have in mind some general theories which appear applicable to the analysis of OSS development, such as theory of distributed cognition (Hutchins and Klausen

1996). To bridge the gap between the hypothesized theoretical elements and the data, sequence data mining was applied. Process models can become cumbersome and analytically complex (Kling 1987; Markus and Robey 1988). The minor contribution of the study to building process theories is demonstrating how much of the data mining can be automatically applied to reduce the analytical effort.

4.2 Theoretical Background

4.2.1 Distributed Cognition

Several information system theories provide a conceptual lens to frame my analysis. The theory of distributed cognition provides a theoretical lens to understand software development. As developed by Hutchins, it posits the perspective that the boundary of cognition processes go beyond individuals to socio-technical systems (Hutchins and Klausen 1996). It presents how cognition processes distribute socially, structurally, and temporally when a distributed team collaborate on information processing tasks.

By conceptualizing cognition as "the propagation of representational state across representational media" (Hutchins 1995, p.118), distributed cognition expands the unit of cognitive analysis from that of the individual to that of the entire team attending to a specific task. With this shift in perspective on cognition, the theory asserts (Hutchins and Klausen 1996): (1) thought processes are distributed among members of social groups, (2) cognition employs both internal and external structures, and (3) cognitive processes are distributed over time.

Nearly all software design efforts are executed through a team structure (Guinan et al. 1998). While addressing complex design challenges, teams must bring together individuals from a variety of technical and functional domains. For example, the cognitive task of arriving at a stable requirements set, referred to as the *computation of requirements*, cannot be localized to any one participant, such as a designer (as is often assumed) (Hansen et al. 2012b; Jarke et al. 2011). Rather, it resides in the holistic process of cognitive computation that enables requirements to emerge as a quality of the social system.

In practical terms, this means that you can interpret the team member activities in distributed cognition terms. For example, team members may rely on the source-code commit log to share progress information about the project. When members fail to submit comments, then this form of distributed information sharing breaks down. Distributed cognition indicates, generally, the kinds of communications and breakdowns they commonly occur.

Other theories provide a background for the development of variables, constructs, and concepts needed to understand information processes in OSS development. For example, the recent theory of *collaboration through open superposition* suggests specific ways in which members collaborate in OSS projects (Howison and Crowston). Classic theories may not be as specific, but establish useful concepts. For example, Galbraith's information processing view suggests the need to examine structural mechanisms, such as information buffers (e.g., a repository), to reduce information uncertainty (Galbraith 1977).

In general, in this study I demonstrate, adapt, and extend software development theories by encoding them in process models and using them to conduct exploratory analysis.

4.2.2 Sequence Stream-Mining

OSS Developers have many interactions, directly or indirectly, through their tools. Some co-located OSS developers will go to their computers to meet, thereby ensuring a record on their meeting (as well as providing access to development records).

Many OSS interactions are logged as event histories. For example, the history of source code changes is maintained by source control systems (e.g., CVS, subversion, Git). As each change is committed to the source repository, the new code and comments are recorded as a change event. Similarly, edits within a code editor (e.g., Eclipse), messages within a chat session, forum comments, feature requests, FAQ edits, *etc*. are all event sequences. Such sequences can be mined for patterns.

Sequence data mining concerns analysis of events in sequence. The event data are often nominal-valued or symbolic and the goal is to discover variables and their correlations (Laxman and Sastry 2006; Zhao and Bhowmick 2003). This contrasts to the well-studied domain of time series analysis, which considers real or complex-valued time series of known parameters using methods such as autoregressive integrated moving average (ARIMA) modeling. Sequence mining techniques address: (1) prediction, (2) classification, (3) clustering, (4) search and retrieval, and (5) pattern discovery.

I apply sequence mining in the context of stream mining. Concepts, techniques and applications of stream mining were reviewed in 3.2.1. Stream mining can detect changes in the data-stream. Two types of algorithms are common: (1) *distribution detection*, which watches for changes in the data distributions, and (2) *burst detection*, which watches for sudden large and unusual changes in a data-stream. Distribution detection algorithms have two common forms: (a) data from two windows (current and reference) are compared using some distance measure, (b) a predictive model is created from a prior window and then its prediction is compared with the current window—high prediction error indicates a significant change. In this study, I apply both distribution detection techniques to discover changes, as well as model differencing, which was introduced in 3.3.3.1, and will be reviewed briefly next.

4.2.3 Model Differencing

Model differencing provides a mean to recognize important changes occurring within an event stream. Consider a stream of development repository events divided into data windows, (w_1, w_2, …, w_n). Transition identification marks each data window as either being normal or transitional. For example, (normal, normal, transitional, normal, normal …). Transitional behavior is historically unusual behavior, according to some measure such as statistical variance. I use the term transitional because the behavior is unusual and transient, and thus interesting from a theoretical perspective, such as cognition or learning theory.

In this study, a repository stream is divided into data windows. Each window is characterized by a model, ($w_i \Rightarrow \lambda_i$). Consider two models in sequence, λ_1 and λ_2. The software finds the difference of the models to characterize the change: $d\lambda/dt = (\lambda_2 - \lambda_1) / (t_2 - t_1)$. If the difference $\Delta\lambda$ is significant, by some measure, then we have found a transition point (Robinson et al. 2013a). The models (λ) vary, but include hidden Markov models (HMMs) for example. Now, because of this automated differencing technique, a monitoring system can quickly identify changes in the models. Thus, some intervention may be applied. In OSS development, this may be changing the project lead, increasing testing, or releasing the software.

4.2.4 HMM Probabilities

Given data containing sequences, a common task is find transition probabilities. That is, given an observed event A, what is the probably that the next event observed with be B or C? A hidden Markov model (HMM) can solve this problem by building a probability model from observed event sequences.

A hidden Markov model (HMM) is a stochastic signal model (Rabiner 1989). In our application to OSS repository analysis, the signals are sequences of discrete typed events (e.g., code commit). A HMM provides algorithms to solve three important problems:

1. Compute the probability that an observed sequence, O, is represented by a HMM, λ (using the Forward-Backward Procedure (Baum and Eagon 1967)).
2. Adjust the parameters of a HMM, λ, to maximize the fit to an observed sequence, O (using the Baum-Welch algorithm (Baum et al. 1970)).
3. Compute the optimal HMM state sequence that best explains an observed sequence, O (using the Viterbi Algorithm (Forney Jr 1973)).

Similar to the work presented in Chapter 3, I use HMMs to model patterns of sequential events within the stream of OSS repository events.

HMM transition identification detects significant changes in modeled events between consecutive windows of event data. HMMs can be used to identify transitions by: (1) *comparing consecutive HMMs* generated from the observation sequences, or (2) *comparing consecutive acceptance probabilities* (Robinson et al. 2013a).

Technique 1 compares consecutive HMMs. This a model differencing technique is generally characterized as follows:

$$\Delta\lambda = \lambda_2 - \lambda_1$$

Here, λ denotes a HMM. To find the distance between two HMMs, the widely applied Kullback-Leibler algorithm is used (Kullback 1997).

Technique 1 directly compares two HMMs, each generated from observation sequences. Technique 2 compares the acceptance probabilities of the observation sequences using the first HMM. Because the two techniques produce similar results, this analysis applies Technique 1.

4.2.5 Volatility Models

A variety of models can be applied to the sequential events found within the data windows of repositories. I consider two that measure variance in sequences: turbulence and optimal matching.

Given a sequence, *turbulence* calculates a metric based on the number distinct subsequences within a data window (Elzinga and Liefbroer 2007). Turbulence increases with the number distinct subsequences.

Optimal matching (OM) generates edit distances that are the minimal cost, in terms of insertions, deletions and substitutions, for transforming one sequence into another. OM can be applied to a repository data window to derive a measure of variance. Consider a data window with two observed sequences: O_i and O_{i+1}. Optimal matching, OM (O^i , O^{i+1}), is 0 if the two sequences are identical; OM increases with the differences between sequences. Like HMM models, turbulence and OM can be calculated for each data window, as well as differenced between two consecutive windows. Thus, I can measure ΔHMM, ΔTurbulence, and ΔOM between consecutive windows of repository data.

4.3 Approach

I obtained development data of 103 OSS projects from GitHub, the most popular open-source code repository site. Founded in 2008, GitHub had over 3 million users and over 5 million repositories as of January 2013. I applied the approach presented in the introduction. In particular, after selecting projects, a KNIME workflow generated sequence models and then clustered the projects. I then characterized the clusters and regressed them with summary project measures. For single project analysis, HMM differencing was applied and transitions were identified. All the steps are automated, from the retrieval of data from the repositories, up to the regression analysis. Finally, I validated some results by comparing the automated analysis with qualitative information. The results suggest that the sequence mining is helpful in clustering projects and detecting interesting transitions within a project.

4.3.1 Data Selection

The search function of the GitHub web site was used to enumerate the projects. The queries are listed in Table 7. To ensure the diversity of the projects, I searched for projects at different level of popularity. I used number of stars and number of forks as proxy for level of popularity. Furthermore, I included a control group which only includes java projects (see query #4 in Table 7), to investigate if language plays a role in projects' development patterns. Note that for the first three queries, the result of each query is a subset of the result of the next query. For example, the 43 projects returned for query #1, are included in the 148 projects returned for query #2. Therefore, to select 30 projects from each query result without any overlap, I sorted each query result differently. The top 30 projects from each query result were selected, for an initial set of 120 projects. Subsequent processing reduced the final set to 103 projects. Projects were dropped because they lacked data. Many Java projects, for example, maintain their issue database outside of GitHub. The analysis was limited to projects contained within GitHub.com. **Table A5** in the appendix enumerates the projects.

#	Query	Sort	Count
1	stars:>10000 forks:>1000	most forks	43
2	stars:>5000 forks:>750	fewest forks	148
3	stars:>1000 forks:>500	fewest forks	651
4	stars:>1000 forks:>250 java	most forks	78

Table 7. GitHub Queries for Data Selection

4.3.2 Data Preparation

A workflow automated the data acquisition and preparation. GitHub.com data was obtained from two sources:

1. GHTorrent provides access to a GitHub database (Gousios and Spinellis 2012). That MongoDB database is the result of GHTorrent monitoring the GitHub public event timeline.
2. GitHub API provides direct access to the project data by sending JSON over HTTPS.

GHTorrent provided the basis for the data. The GitHub API was used to validate the data, and in some cases provide missing data.

The GitHub data is comprised of a 16 collections, which we combined, through filtering and joining, into a single table. The data was derived mainly from these collections: issues, issue events, issues comments, pull requests, and pull request comments. The resulting table consists of these fields: Issue number, body, diff hunk, path, position, original position, commit id, original comment id, create at, updated at, event comments, milestone, title, assignee, closed at, state, merged at, head, base, and actor. Each record in the table provides a vector for input into our data mining process.

The table represents a sequence Git events. Of the 18 Git events, I focused on six, which most closely associated with teamwork:

1. IssuesEvent: An issue is created, closed, or reopened.
2. PushEvent: Code is committed (pushed) to the repository.
3. PullRequestEvent: A user requests that new code be pushed to the repository.
4. IssueCommentEvent: A comment is associated with an issue.
5. CommitCommentEvent: A comment is associated with a commit (PushEvent).
6. PullRequestReviewCommentEvent: A comment is associated with a PullRequest.

Other events, such as watch events in which users subscribe to a repository to get updates, do not concern teamwork. They are thus excluded from the study. A list of Git events are summarized in **Table A4**. The prepared table of sequential Git events is further process to represent elements of teamwork.

4.3.3 Work Constructs

Git events, such as push and commit, represent work; however, the context of the work is missing. For example, it seems that 10 code commits for the same issue is different than 10 code commits, each for a single issue. A rule-based system is applied to the prepared event data to derive a table of abstracted work events.

Work in most GitHub projects begins with an IssueEvent or a PullRequestEvent. Both represent a typical unit of development work, which may be scheduled, opened, closed, reopened, etc. An IssueEvent typically represents a bug or enhancement. It follows a common lifecycle of being opened, followed by code changes represented by commits, and then an issue close. For example:

IssuesEvent.open, PushEvent, PushEvent, IssuesEvent.close

Of course, other events may intervene (e.g., comment events), as well as the issue may be reopened or never closed.

The PullRequestEvent is similar to the IssueEvent, but the subsequent work events are related to integrating the new code into the project's code repository. A rule-based system is applied to recognize event sequences beginning with IssueEvent or a PullRequestEvent. I think about them as min-workflows, which are initiated in response to a work request (e.g., issue or pull request). However, here I use the more neutral term, motif, to indicate recognition of these common sequence patterns.

The rule-based system recognizes two kinds of work motifs in the prepared table of sequential Git events. The basic form is as follows:

1. (IssueEvent | PullRequestEvent) .*
2. (Reopen (of #1)) .*

As indicated above, a work motif begins with either an IssueEvent or PullRequestEvent, followed by any other Git event that references the initiating event (by number). The motif records the initial event, and all subsequent events (and their attributes). When either an IssueEvent or PullRequestEvent is reopened, it is consider a new instance of the second motif pattern (above). Thus, open and reopen are each considered the beginning of a work motif. The subsequent analysis shows common event sequences; however, those are Git event sequences within the context of these work motifs.

These work motifs are derived from the prepared data in support of our theoretical background. Thus, I call these derived, abstracted elements work *constructs*, to be consistent with theorizing process theories (Van De Ven 2007).

4.3.4 Sequence Feature Construction

Before the work motifs can be sequence mined, they are encoded. Most sequence data-mining algorithms process event sequences, where events are identified as members of a fixed alphabet. An event sequence, for example, could be A-B-A-B-B-C. Few algorithms directly address object sequences, where each sequence member is comprised of an information object. An object sequence, for example, could be

[Issue.ID=1,Issue.Author=a1,Issue.State=open]-

[Commit.ID=10,Issue.Author=a1,Commit.IID=1]-

[Issue.ID=1,Issue.Author=a1,Issue.State=close].

Object sequences can be processed by dropping information. For example, just processing object type information: Issue-Commit-Issue. More information can be processed by first encoding the object information into another alphabet; for example, [Issue.ID=1,Issue.Author=a1] becomes I1A1. We applied this transformation concept to the sequence of work motifs.

Given the work-motif sequences for a GitHub project, we using k-means clustering to transform each work motif into one of 50 clusters. The work motif is represented by a vector of these attributes:

IssueEvent | PullRequestEvent (open|reopen), ID, openDate, events, actors, state(open|closed), merged(true|false), duration, comments, turbulence.

The rationale for selecting those attributes is as follows: different instantiations of the same work motif might present different levels of collaboration. For example, in an IssueEvent→CommentEvent→PushEvent work motif, the level of collaboration would be different between an instantiation in which the same actor performed these three activities, and an instantiation in which three different actors performed these three activities. As discussed before, DCog seems an appropriate lens to interpret the development activities. Therefore, I used concepts from this theory, to extract relevant attributes of a work motif, to differentiate instantiations of work motif. I reviewed the indicators of distributed cognition based on existing literatures (Hansen and Lyytinen 2009; Hansen et al. 2012a; Thummadi et al. 2011) and summarized them in **Table A1**. From these indicators, relevant attributes in the work motifs were identified. This input vector is then encoded as cluster number [0, 49]. The sequence miner then processes sequences of these work-motif clusters numbers.

4.3.5 Sequence Modeling

Give sequences, EventMiner constructs models of (a) sequence pattern probabilities, (b) entropy within sequences, and (c) changes in the patterns and entropy over data windows. Sequence pattern frequencies are determined by simply counting pattern occurrence. For example, Figure 11 illustrates the frequency of the top 20 sequential patterns from the bootstrap project. Additionally, a hidden Markov model (HMM) calculates the sequence pattern probabilities. Optimal matching (OM) is applied to pairs of sequences, to determine a matching distance.

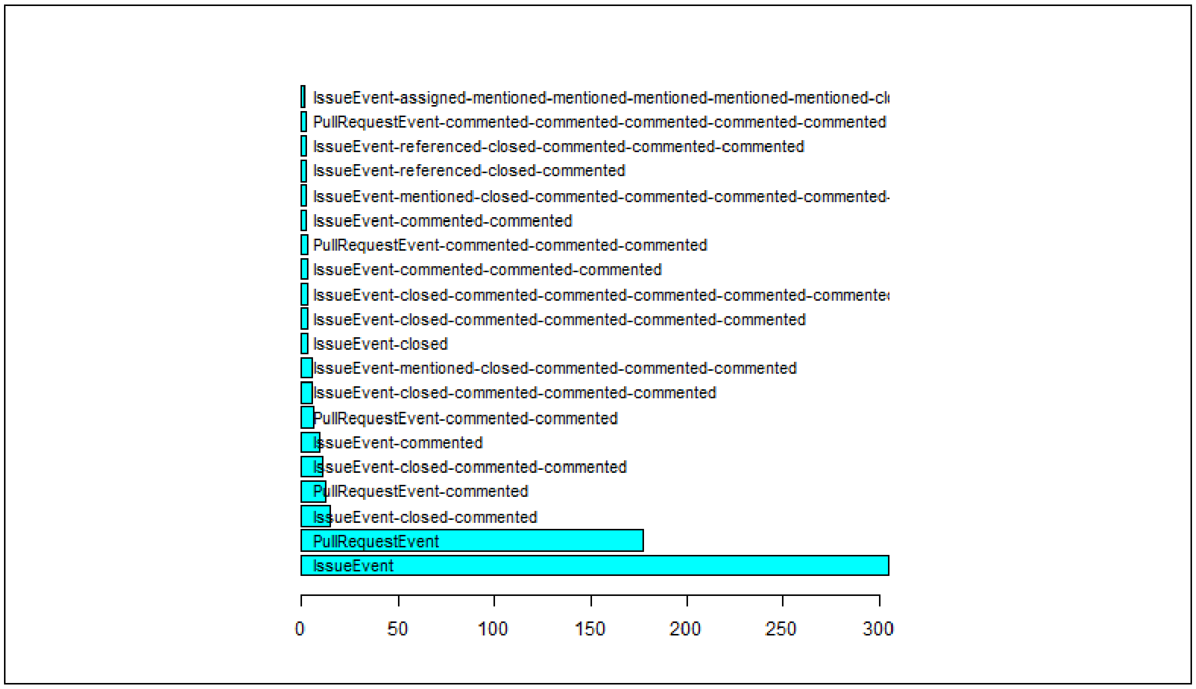


Figure 11. Sequence Frequencies for Bootstrap

Turbulence and OM are applied to sequential data windows of work motifs (themselves sequences). A series of data windows results in a series of values ($Turb_t$, OM_t … $Turb_{t+n}$, OM_{t+n}) indicating kinds of sequence entropy over time. Change in sequences over time can be calculated for turbulence, OM, and HMMs. The general equation is simply $\Delta\lambda = \lambda_2 - \lambda_1$, where λ represents either turbulence, OM, and HMM models. I calculate the models and their differences for each data window for each project. After some preliminary analysis, I choose four weeks for the data window size—it contains sufficient data and represents a common unit of work for open source development methodologies. The subsequent Figure 12 illustrates ΔHMM for a project from each cluster derived.

4.3.5.1 Project Clustering

Projects can be clustered by their change of sequencing, as represented by models of the prior section. Each project gets sequences of change in sequence turbulence, OM, and

HMMs; that is, Δλi where the model λ is one of sequence turbulence, OM, and HMM. Given the projects, each represented by a sequence of values, I pairwise compare them using optimal matching to generate a distance matrix. I apply hierarchical clustering to the distance matrix to deriver the clusters. (The appendix **Table A5** includes the cluster number for each project.)

Table 8 summarizes the cluster characteristics. Clusters 3 and 4 have relatively more stars, forks, open issues, and have relative short cycles[1]. I reviewed one project from each cluster. From clusters 3 and 4, ember.js and brackets, respectively, are relatively active. Ember.js has a 15 day release cycle, and brackets has a 17 day release cycle—relatively small comparing to the average of 101 days among all projects. Ember.js also has a big community of 360 contributors and has very frequent group meetings through google hangout. They used various communication channels including discussion forum, blogs, GitHub site and stackoverflow.com. Project information from such sources provides detailed projects information.

Cluster	**# of Projects**	**Forks**	**Stars**	**Daily Forks**	**Daily Stars**	**Release Time (Days)**	**Age (Days)**
0	29	1363.55	5952.55	1.37	5.96	141.72	1099.31
1	38	1094.00	5453.66	0.72	3.53	137.21	1581.16
2	21	885.10	5239.43	1.81	10.82	40.62	579.19
3	7	2432.71	14103.29	2.34	14.23	28.00	1081.00
4	8	2025.88	10131.25	3.64	19.89	8.25	576.00
project mean		**1290.66**	**6501.59**	**1.46**	**7.70**	**101.35**	**1129.15**

Table 8. Cluster Characteristics

The other three clusters (cluster 0, cluster 1 and cluster 2) are similar in terms of popularity—they have relatively similar number of forks, stars and watchers. However,

[1] Space limitations prevents us from showing all the cluster attributes.

among these three clusters, the daily stars of cluster 2 suggest that projects in this cluster were able to attract stars faster than the projects in the other two clusters. Clusters 2 also have a shorter release cycle time: 40 days compared to 142 days in cluster 0 and 137 days in cluster 1. Projects in clusters 0 - 1 are the least popular—they have the lowest forks rate, and star rate. They also grow relatively slower—both of them have smaller open issues rate than others, and have longer release cycles.

4.4 Analysis

The goal of the study is to monitor an OSS project by observing and analyzing its activities, I focused on two analysis tasks here: (1) detecting behavioral transitions within a project, to achieve step 3 in the introduction, and (2) investigating the relationship between sequential behaviors and project measures. Understanding on this relationship can help us achieve step 5 and step 6, in which inference between behavioral patterns and success needs to be made explicit. The descriptive characteristics of the clusters suggest that the KNIME workflow for sequence mining is useful for the classification of GitHub.com projects by their sequential behaviors. Because I want to relate the sequential behavior with project outcomes, I performed a linear regression of the clusters with summary project features. Finally, as an added level of validation, I sampled the clusters and correlated their mined characteristics with independent information. In particular, I showed how we used sequence mining to detect behavioral transitions in a project, and how the transitions were evaluated.

4.4.1 Regression

For exploration, I applied linear regression to determine if the clusters were related to project summary characteristics, some of which can be interpreted as project performance measurements.

I created dummy variables for the clusters and then ran the linear regression between those dummy variables and project summary variables. I chose cluster 4 as the reference group because it has the most daily stars and daily forks. I included project age and number of collaborators as control variables. The results in Table 9 show that the clusters are related to daily stars (adjusted R2 of 0.576) and daily forks (adjusted R2 of 0.477). The control variable collaborator is not significant, and therefore it is not included in the result table. The coefficients show that projects in cluster 0, 1, and 2 have significantly fewer daily forks and daily stars than projects in cluster 4. To better compare the clusters, I collapsed cluster 3 and cluster 4. This procedure resulted in four new clusters. Their characteristics are summarized in Table 10.

Variables	Daily Forks		Daily Stars	
	Model 1	Model 2	Model 3	Model 4
Constant	2.943***	4.158***	16.705***	23.017***
Project Age	-.001***	-0.001***	-0.008***	-0.005***
Cluster 0		-1.800***		-11.903***
Cluster 1		-2.025***		-10.906***
Cluster 2		-1.833***		-9.052***
Cluster 3		-0.845		-2.923
R^2	0.319	0.503	0.386	0.596
ΔR^2	0.319***	0.184***	0.386***	0.211***
Adjusted R^2	0.313	0.477	0.380	0.576
Observations	103	103	103	103

***: $P \leq 0.001$

Cluster 0, Cluster 1, and Cluster 2 and Cluster 3 are dummy variables created for the 5 clusters.

Table 9. Regression Results (Five Clusters)

Cluster	# of Projects	Forks	Stars	Daily Forks	Daily Stars	Release Time (Days)	Age (Days)
0	29	1363.55	5952.55	1.37	5.96	141.72	1099.31
1	38	1094.00	5453.66	0.72	3.53	137.21	1581.16
2	21	885.10	5239.43	1.81	10.82	40.62	579.19
3	15	2215.73	11984.87	3.04	17.25	17.47	811.67
project mean		**1290.66**	**6501.59**	**1.46**	**7.70**	**101.35**	**1129.15**

Table 10. Cluster Characteristics (Four Clusters)

Dummy variables were again created for the new clusters, with cluster 3 as the reference group. I applied linear regression again to determine if the clusters were related to project performance. The results in Table 11 show that the clusters are related to daily stars (adjusted R^2 of 0.573) and daily forks (adjusted R^2 of 0.466). The results also show that projects in cluster 0 - 2 have significantly fewer daily stars than projects in cluster 3. Likewise, projects in cluster 0-2 have a significantly slower fork rate. Given the great variability of the projects, as well as the internal variability of the sequential behaviors within projects, I am encouraged by the R^2. It suggests that, with improved models, we may be able to correlate project behaviors with summary project features.

As discussed before, I am interested on the relationship between usage of language and development behaviors. Especially, I wanted to investigate if projects that development with java would present different development behavioral patterns. Therefore, I compared the usage of java among the five groups. I found that projects in cluster 0 use java more frequently than projects in other clusters. Interestingly, projects in this cluster are the least popular, with slow growth rate. It would be interesting to look into the behavioral patterns of projects in this group and compare them with that of other projects.

Variables	**Daily Forks**		**Daily Stars**	
	Model 1	**Model 2**	**Model 3**	**Model 4**
Constant	2.943***	3.866***	16.705***	22.006***
Project Age	-.001***	-0.001***	-0.008***	-0.006***
Group 0		-1.369***		-9.603***
Group 1		-1.534***		-9.207***
Group 2		-1.467***		-7.789***
R2	0.319	0.487	0.386	0.590
∆ R2	0.319***	0.167***	0.386***	0.204***
Adjusted R2	0.313	0.466	0.380	0.573
Observations	103	103	103	103

***: $P \leq 0.001$

Group0, Group1, and Group2 are dummy variables created for the 4 groups after consolidation

Table 11. Regression Results (Four Clusters)

4.4.2 Behavioral Transitions

Transition identification detects significant changes in modeled events between consecutive windows of event data. Space limitations prevent us from showing all projects, or even all of a single project. However, Figure 12 shows five projects, one from each cluster. Two projects discuss herein, ember.js and bootstrap, are the first and second projects.

The *x*-axis represents the Kullback-Leibler comparison of HMMs generated from the data windows. Each point represents the comparison between two HMMs, each representing a month of data. The trend values are more important than the specific HMM comparison values. Notice that all projects have periods of transition, where their behavior models change significantly, as shown by the spikes. This illustrate how well HMM differencing discriminates unusual periods of sequential behaviors from the more common background.

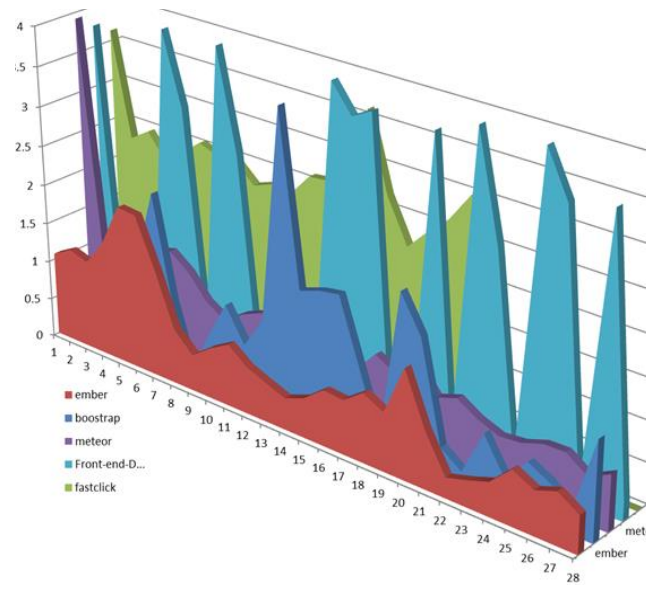


Figure 12. HMM Differences of Cluster Samples

The transitions (spikes) displayed in Figure 12 represent real changes in developer behavior—the developers have changed their patterns of their work motifs. We have correlated those changes with web data to validate that interesting team behaviors are being monitored.

Table 12 summarizes bootstrap blog entries that correlate to the transitions presented in Figure 12. (The x-axis is window count, not date.) These entries provide corroborating evidence that the transitions capture meaningful team behavior.

Week	**Events**
34	On 14th, there was an announcement about a future release of 2.0.3 on the blog. On 15th, the team asked for help on testing 2.0.3. Version 2.0.3 was released on 24th.
41	Version 2.0.4 was released on June 1st, 2012
57	The Bootstrap team announced on the blog on Sept 29th, 2012 that they were leaving twitter.
63	On Nov 9th, 2012, the team informed on their blog that version 2.2.2 will not include glyphicons.
67	The team asked for help on their blog for testing version 2.2.2 on Dec 2nd. 2012. They released version 2.2.2 on Dec 8th. Later on the 12th, they posted plans for Bootstrap 3.
68	The same as above.
70	Two pull requests were posted on GitHub on Dec 20th, 2012: (1) a pull request for Bootstrap 3, which would be the next major release with lots of changes, and (2) a pull request for version 2.3.0.

Table 12. Blog Entries for Bootstrap Transitions

For example, there is a spike at window 41. In the week, the bootstrap team posted a blog entry introducing a new plan of Bootstrap 3. Therefore, an explanation for the spike is that the team began work for Bootstrap 3 after the announcement, bringing a change on their development behavior. Another example is the spike of window 68. Several big events occurred during this period: on the Sunday of the first week, the team asked their contributors for help on the blog, for testing the coming new version 2.2.2. The version came out on Saturday of the same work. Therefore, we can expect that the first week would be a busy week for the team, compared to the following week, after version 2.2.2 was released. A similar example occurred at window 34, when the team went through another release announcement. Other spikes can be similarly explained.

4.5 Discussion

This study is aimed at understanding the sequential behaviors of developers in OSS projects. The experiment produced meaningful clusters from 103 GitHub.com projects. Our regression of clusters with daily stars and daily forks is encouraging, but limited. It demonstrates how automatically mined constructs may be linked to higher-level theoretical concepts. This experiment is an instance of our overall methodology for theory exploration.

Automation was implicit in our discussion; however, all the steps, from project data retrieval up to the regression analysis are automated. Retrieval of the project records is, by far, the slowest part of the analysis. On an ongoing basis, a dashboard can update many projects every minute.

Based on these results, the dashboard would simply cluster projects into those that have more daily downloads and stars. Note that such characterization was obtained exclusively from the sequential behaviors of the developers. This suggests that in the future, with more complete models, we may produce more refined behavioral analysis on what is most effective for OSS team success.

For future research, I plan to further investigate transitions within sample projects. I will examine quantitative and qualitative measurements of the project at transition points. For example, it would be interesting to examine if the team size and team structure changes around those transition points. I will apply theory of distribute cognition to explore the nature and reasons of those changes, for the purpose of constructing a process theory on OSS project development behaviors. This model will contribute to achieving step 4-6.

Limitations of this study include: (1) sampling, in that the projects may not be representative of OSS projects in general; and (2) modeling, in that the measurement of and constructs for sequential behaviors are incomplete. Future work will seek to diminish these limitations.

4.6 Conclusions

OSS developers generate many events, through their tools, which can be used to monitor their progress and predict their results. A carefully constructed data mining workflow can automat the acquisition and analysis of repository events to present a dashboard of clustered projects, highlight when significant changes in developer behaviors have occurred. Now, such automation is of great help to researchers who seek to demonstrate, adapt, and extend software development theories by encoding them in operational process models and using them to conduct exploratory analysis. When such research results become practical, then the future dashboards will produce more refined behavioral analysis on what is most effective for OSS team success.

5. Investigating the Temporal Dynamics and Variety of OSS Development Activities

5.1 Introduction

The previous study of Chapter 4 reveals the relationship between sequential patterns of development processes and project performance. To extend my understanding on this relationship, I examine the role of design routines in shaping the OSS project performance. I develop a factors model that includes measures of variability and change in routines, which represent the process of OSS development. A relationship between routine diversity and change and project performance is found and discussed.

OSS projects are known for their chaotic development style (Mockus et al. 2002).Several significant characteristics of OSS development are the following:

- Work is self-assigned: contributors choose what they want to undertake (Crowston et al. 2007; Crowston and Scozzi 2008).
- There is a lack of coordination mechanisms, which are observed in traditional development settings—there are few formal "plans, system-level design, schedules, and defined processes" (Crowston et al. 2007; Herbsleb and Grinter 1999; Mockus et al. 2002).
- Multiple different processes are performed by contributors simultaneously (Christley and Madey 2007).

However, the "chaotic" development processes produce high quality software. Most extant literature attributes this superior quality to static cross-sectional factors such as number of developers, project activity level and license choice. Despite the attribution of success to identifiable factors associated with OSS projects, many successes may still be regarded as "chaotic" and seemingly disregard conventional wisdom regarding project success. While variance studies can identify some predictors of project outcomes, they tend to neglect the actual process of development, which may or may not occur "chaotically." Without a closer examination of how events occur, an understanding of OSS projects is incomplete. This motivates me to look at the impact of characteristics of the development processes, such as process variation and change, on OSS performance. Although a variance model is developed, the characteristics of the development processes is considered. Thus, this study combines process and variance approaches.

In this study, OSS development is conceived as a sequence of design routines. Gaskin et al. (2010) defines a design routine as "a sequence of (design) tasks, which transform some representational inputs into a set of material and representational outputs, leading ultimately to a generation of design artifact." Design routines are believed to have less clearly defined inputs and outputs resulting from changing requirements (Dorst et al. 1996; Gaskin et al. 2011). In the context of open source software development, design routines are the prevailing activities (Gaskin et al. 2014; Gaskin et al. 2011; Gaskin et al. 2010). However, there is a lack of research that empirically investigates routine diversity and routine changes in the OSS context, and the underlying theoretical relationships, via analysis of big, digital trace data. To fill this gap, and to explore the relationship between the changing dynamics of development processes and project performance, I propose to address the following research question:

What are the impacts of routine diversity and routine changes on OSS project performance?

I base the analysis on a view that OSS projects are comprised of sequences of design routines, which take a diversified set of forms and change over time (Pentland et al. 2011). Drawing upon literatures in routine, I develop and empirically test a model of design routine diversity and change, and their impact on project performance. Extracting digital trace data from GitHub.com, I examine this model with a computational, mixed-method approach. This study contributes to both OSS and routine literature, and provides implication for OSS practitioners.

5.2 Research Model and Hypotheses

5.2.1 Research Model

The research model in Figure 13 includes routine diversity and routine change, and the effect of these constructs on OSS project performance. This model incorporates theoretical concepts including routine diversity, routine change, and project attractiveness. In this model, routine diversity and routine change contribute to project attractiveness. The unit of analysis is at the project level.

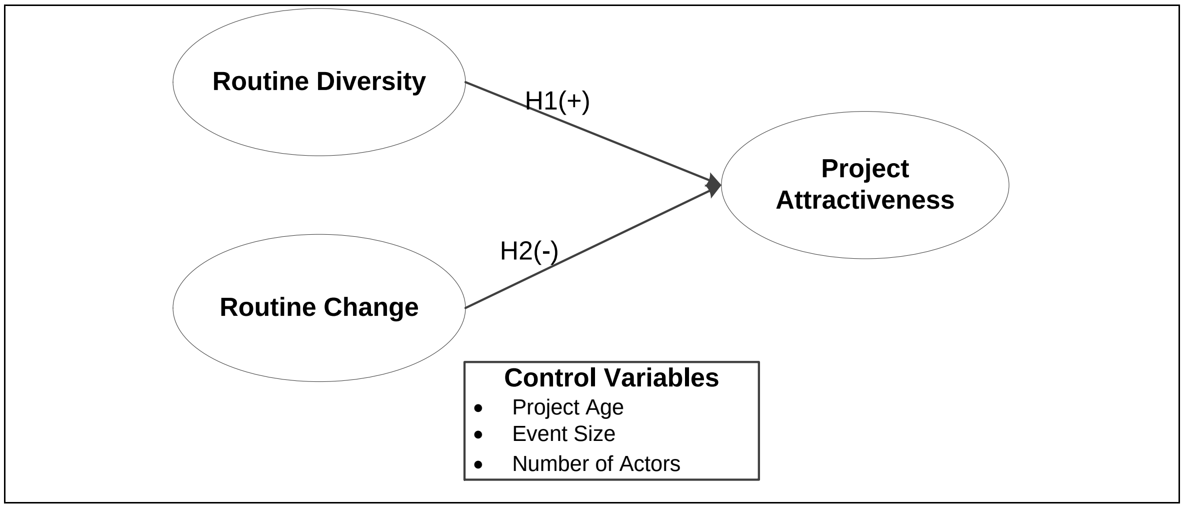


Figure 13. Research Model

5.2.2 Research Hypotheses

5.2.2.1 Relationship between Routine Diversity and Project Attractiveness

There are four explanations why routine diversity in OSS development processes can attract users and developers. Firstly, a complexity theory perspective suggests that diversity enhances robustness of complex adaptive systems (Benbya and Mckelvey 2006; Page 2010). ISD projects are commonly considered as complexity adaptive systems (CAS) (Benbya and Mckelvey 2006; Van Aardt 2004). OSS projects, are considered by researchers as the best example of CAS (Muffatto and Faldani 2003; Van Aardt 2004). Complexity is magnified in the OSS development context, by the continuous changes in the requirements. As a complex system, an OSS project with more diversified processes maintains more robustness, which in turn attracts potential developers and users. Secondly, a diversified set of routines allow developers to contribute to the project in multiple ways, thereby encouraging developers' contribution (Lindberg 2013). Pentland (1995) and Pentland and Rueter (1994) conceptualized routines as grammar to explain how variation in routines allow participants to produce a variety of performances. As they explained, "in the same way that English grammar allows speakers to produce a variety of sentences, an organizational routine allows members to produce a variety of performances" (Pentland and Rueter 1994, p.490). An OSS project with a higher level of routine diversity will allow developers to use more routine configurations, thus attracting more developers to participate. Thirdly, routine diversity is believed to be an indicator of innovation (Nelson and Winter 1982; Pentland and Rueter 1994). A project with higher routine variety provides more innovation opportunities, which can attract more developers. A project with low routine variety suggests simpler, repetitive routines, and thus failing to attract developers and users. Fourthly, routine diversity can facilitate learning. Variation has been considered as foundation for learning in general (Campbell 1960). A diversified set of routines, provide participants with more opportunities to learn the “lesson of history” as (Levitt and March 1988) suggests. As a result, developers will be more willing to participate in a project in which they can learn from various routines that codify experience. Therefore, I expect that:

H1: Routine diversity is positively associated with project attractiveness.

5.2.2.2 Relationship between Routine Change and Project Attractiveness

A project with unpredictability, such as a high level of temporal change, requires a substantial effort for participants to learn and adapt (Conboy 2009). Dramatic changes in design routines result in information overload (Dierickx and Cool 1989; Hambrick et al. 2005) and increased design difficulty, and thus hinders participation (Cant et al. 1995; Robbins and Redmiles 1996; Subramanyam and Krishnan 2003). Additionally, a high level of temporal change indicates a lack of control. Such projects will lose the capability of attracting new users and developers. This line of reasoning leads to the following:

H2: Routine change is negatively associated with project attractiveness.

5.2.2.3 Control Variables

A variety of characteristics can affect the popularity of an OSS project. For example, the longer a project has existed, the more likely that it will be widely known and consequently obtain forks and stars. Number of contributors may also increase forks and stars. Therefore, I control for project age, average number of actors per routine, and average number of events per routine.

5.3 Method

To test the model, I use a computational, mixed-method approach to analyze digital trace data collected from an open source repository.

5.3.1 Data Collection

A web crawler collected digital trace data from GitHub.com, one of the most popular open-source code repository sites. A stratified sample strategy was applied to ensure variation along the performance variables. Average number of stars a project gets per day (daily stars) and average number of forks a project gets per day (daily forks), are proxies for the level of attractiveness to users and developers. Random samples of projects are selected from the following three groups: (1) projects with more than 10,000 stars and more than 1,000 forks, (2) projects whose number of stars is between 5,000 and 10,000 and number of forks is between 750 and 1,000, and (3) projects whose number of stars is between 1,000 and 5,000 and number of forks is between 500 and 750. In total, the development activity data across seven years (January 2008 to April 2015) from 150 OSS projects were obtained.

5.3.2 Routine Elicitation

Each of the 150 projects is represented as a sequence of development activities. Design routines are extracted from the activity sequences. As discussed previously, I only focus on two kinds of routines: issue handling routines, and pull-request handling routines. A design routine begins with either an IssueEvent or PullRequestEvent, followed by events that reference the initiating event. A typical handling routine starts when a developer posts a pull request, proposing to commit changes to the current code base. Members then comment on the pull request, making suggestions, and referencing other pull requests and issues. Core developers, determining how well the proposed commits follow the repository’s standards, also review the pull request. The pull request is either merged by core developers into the main source code, or rejected. Finally, the pull request is closed. Using a rule-based system, I extracted 92988 routines from the 150 projects; with an average of 46 routines per month per project (summary statistics of those routines is provided in **Table A6** in the Appendix).

In accordance with the definition of routine diversity as "different configures of the same routine type" (Pentland et al. 2011), I clustered the extracted routines into 50 clusters based on eight configuration elements: type of open event (issue event or pull request event), open actor and close actor (same or different), duration, final state, outcome, number of unique actors, number of comments, and number of activities in the routine. Thus, each routine instance is coded into one of 50 routine types.

5.3.3 Constructs and Measurements

After extracting routines from activity sequences, I constructed a dataset to test the proposed model. Table 13 presents the constructs in the proposed model along with their measures. Because the unit of analysis is at the project level, I collected and constructed the measures for each project.

Constructs and Measurements			
Construct	**Measurement Item**	**Definition**	**Reference**
Routine Diversity	Activity entropy	Sum of probability of activity types at a given period	(Lindberg 2013; Robinson and Deng 2015)
Routine Change	HMM difference	Magnitude of changes in varied sequences of routines between time windows	(Rabiner 1989; Robinson et al. 2013b; Robinson and Deng 2015)
Project Attractiveness	Daily fork and daily star	Average number of forks (and stars) a project gets per day	(Dabbish et al. 2012)
Project Age	Project Age	Number of days since the project was created on GitHub	
Event Size	Event Size	Average number of events per motif of the project	(Robinson and Deng 2015)
Number of Actors	Number of Actors	Average number of actors per motif of the project	(Robinson and Deng 2015)

Table 13. Constructs and Measurements

5.3.3.1 Routine Diversity and Routine Change

I use the Shannon-Wiener index (Shannon 2001), to measure the average *routine diversity* in a project. Shannon's index has been used to calculate entropy, which has been defined as a transversal distribution of activities (Gabadinho et al. 2011). In our context, entropy captures the distribution of different design routine variations in a given time.

I use average Hidden Markov Model (HMM) difference, to measures the degree of design *routine change* in each project. A Hidden Markov model (HMM) is a commonly-used probability model for anomaly detection (Rabiner 1989). Each project is divided into data windows. For each window, an HMM (λ_i) is created representing the sequence of activities in the design routines. Given two HMMs in sequence, λ_1 and λ_2, model differencing characterizes the change: $d\lambda/dt = (\lambda_2 - \lambda_1) / (t_2 - t_1)$. This gives the magnitude of change in the transition probabilities, for a sequence of data windows — their average is used to measure the magnitude of design routine change.

5.3.3.2 Project Attractiveness

I use *daily fork (daily numbers of new forks)* and *daily star (daily number of new stars),* to measure the project attraction of users and developers. Forks show the popularity of a project with users or developers, or how useful this project is perceived by developers (Dabbish et al. 2012). Creating a star for a project allows the user to create a bookmark for easier access [to the project] and show appreciation to the repository maintainers (Github.Com). Crowston and colleagues categorized OSS success measures into three types: measurements concerning with the process, measurements concerning with project output, and measurements concerning with outcomes for project members (Crowston et al. 2003; Crowston et al. 2004; Crowston and Howison 2006). A summary of OSS success measures is listed **Table A2**. Mcdonald and Goggins (2013) suggested that “process measures may be more salient than product quality measures in distributed source code management systems like GitHub” (Mcdonald and Goggins 2013, p141). They argued that although studies have attempted to provide success metrics of success in OSS projects, the role of a code hosting workspace plays in how performance is viewed and measured has not been examined. By conducting interviews with members of projects hosted on GitHub, they found that developers used GitHub’s visible metrics of contribution (commits, pull requests, forks, etc.) and metrics of activity (commits, forks and stars) to measure success. Release quality and bug fixing, which are measures on product quality, were rarely mentioned as measures of success. Because measurements of commits and pull requests are already included in the development routines, I choose daily fork and daily star as the measures of project performance.

5.4 Data Analysis

At the current stage of the research, I have obtained the following data for 88 projects (some projects were excluded due to missing performance data): activity entropy, HMM difference, fork and star rate, event size, and average number of actors per routine. The descriptive statistics is presented in Table 14.

	Mean	SD	1	2	3	4	5	6	7	8
1. Daily Star	8.22	6.99	—							
2. Daily Fork	1.54	1.25	.77**	—						
3. Motif Event Size	4.42	1.92	0.19	0.16	—					
4. Motif Actor Number	2.22	0.58	.29**	.26*	.87**	—				
5. Project Age (day)	1058.15	516.50	-.61**	-.55**	0.15	0.13	—			
6. Cycle Time	91.40	175.54	-.28*	-0.16	-.28*	-.25*	0.01	—		
7. Δ HMM	1.61	0.79	-0.18	-.31**	-0.10	-0.04	0.06	0.17	—	
8. Entropy	1.83	0.87	.48**	.47**	.61**	.63**	-0.08	-.46**	-0.19	—

n = 88

*: $P \leq 0.05$. **: $P \leq 0.01$.

Table 14. Descriptive Statistics

I applied ordinary least-squares (OLS) regression to estimate the two dependent variables: fork rate and star rate. Before completing our analysis, assumptions of the multiple linear regression model for the ordinary least squares method were checked (i.e., normality and multicollinearity among independent variables). Diagnostic checks on residuals were conducted to ensure the assumptions of normal distribution of residuals are not violated. I also examined the multicollinearity among explanatory variables.

5.5 Results and Discussion

Table 15 presents the regression results. Daily forks, is positively and significantly associated at the 0.001 significance level with entropy. It is negatively and significantly associated at the 0.01 significance level with ΔHMM. A similar pattern arises from the other dependent variable, daily star; it is positively and significantly associated at the 0.001 significance level with entropy. However, it is not significantly associated with ΔHMM. Thus, there is strong statistical support for design routine diversity and change with outcomes of project attractiveness. Additionally, variables that significantly affect daily forks and daily stars include workflow size, number of actors, and project age. Thus, H1 is supported and H2 is partially supported.

Variables	**Daily Forks**		**Daily Stars**	
	Model 1	**Model 2**	**Model 3**	**Model 4**
Constant	1.178*	1.598***	6.294*	6.89**
Event Size	-.117	-0.228*	-0.615	-1.071*
Number of Actors	1.068**	0.875*	6.295***	4.89**
Project Age	-0.001***	-0.001***	-0.009***	-0.008***
Entropy		0.497***		2.706***
Δ HMM		-0.378**		-0.844
R^2	0.417	0.561	0.518	0.598
ΔR^2	0.417***	0.144***	0.518***	0.08***
Adjusted R^2	0.396	0.534	0.500	0.574
Observations	88	88	88	88

*: $P \leq 0.05$. **: $P \leq 0.01$. ***: $P \leq 0.001$

Table 15. Regression Results

In the context of OSS development routines, entropy measures routine diversity, while ΔHMM measures the magnitude of change. The results reveal that a project attracts more developers and users, with a more diversified set of routines. On the other hand, when a project experiences dramatic routines changes, it becomes less appealing to developers. Furthermore, it would seem that active developers who usually fork projects to contribute, are more sensitive to both measures, while more passive people (e.g., users and occasional developers who usually ‘star” projects just to get easy access) are less sensitive. People that only occasionally interact with a project may not even notice the change. Taken to the extreme, an uncontrolled, chaotic project with wild swings in routine diversity may be recognized by active developers as a failure, which then causes a drop in daily forks; yet, those less aware or affected by the apparent pending doom do not significantly alter their daily stars. This line of reasoning rationalizes the findings of

Table 15. These findings will contribute to both researchers and practitioners. With the insights provided by this study, OSS participants can better steer their projects to attract more developer participation. The study also contributes to literature in both OSS development and routines.

To extend the discussion, I illustrate a speculation about projects types according to the two dimensions of entropy and ΔHMM in Figure 14. Chaotic, floundering projects, commonly termed thrashing (or death march (Yourdon 2003)), are illustrated in the upper right of Figure 14. A maintenance project, with its simple, occasional updates, is the opposite. Regular moderate change indicates an innovative, successful project (Klarner and Raisch 2012). While frequent changes that have little lasting effect on routine diversity may indicate ineffective management interventions (Salvato 2009). Future research is necessary to understand how these dimensions affect project type.

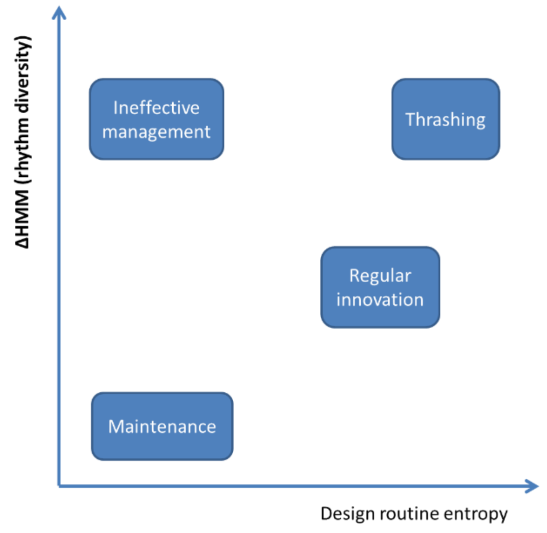


Figure 14. Hypothesized Projects Types

5.6 Contribution

Building on literatures on routine, this study proposes and tests a model that captures the underlying theoretical relationship between routine diversity and change, and OSS project performance. The contributions of this paper are epistemological and methodological. The study also provides insights to practitioners for project management.

The study contributes to literature both in OSS development and in design routines. It extends the current understanding of why and how OSS design routines change over time, and what is the effect of such diversity and changes on project performance. It also provides a novel perspective to predict OSS project performance by routine characteristics. The study makes a further methodological contribution by identifying and demonstrating appropriate data analysis methods for digital process data. For practitioners, a better understanding on drivers and effects of diversity and change of routines can help them to better manage and steer their projects to attract and sustain developer participation.

6. Conclusion

6.1 Research Objective Revisited

Motivated by the importance of OSS development process evolution and gaps in extant literature, this dissertation aims to address the following research questions:

How and why do OSS projects evolve through development processes?

What are the impacts of development processes' evolution on project performance?

To answer these questions, I developed EventMiner, an event-based analysis framework that integrates several stream mining techniques. I applied it to two different data sources: stream data from software usage and the event stream data of 103 OSS projects on GitHub.com, an open source repository. In the first study, I tested and demonstrated the application of the framework in the context of software usage behaviors. In the second study, I detected behavioral transitions of projects and clustered projects based on their sequential patterns. By doing this, I addressed the "how" in the first research question. A future research direction of this study will be to interpret the change patterns of OSS development, to explore "why" those changes occur, and to build process models on OSS development. In study 3, I developed a factor model to relate development process changes with project performance, thus addressing the second research question.

6.2 Limitations

Like any other research, this dissertation is not without limitations. The findings are based on the sample projects, which may not be representative of OSS projects in general. One advantage of EventMiner is that the automatic, workflow-based feature makes it easy to scale up the analysis to different contexts and much bigger datasets. One future research direction will be to apply EventMiner to other OSS projects from other open source repositories, such as Jazz.net and SourceForge.

In addition, the findings are limited to the data I can obtained from GitHub. A lot of communication among GitHub developers is stored in emails, forums, and bug track systems that are independent from the GitHub site. Obtaining and analyzing such data for the projects I studied will be my next step.

6.3 The Continuing Research Stream

I plan to extend my current study in three main directions.

Firstly, I want to build process models of OSS development. I have demonstrated in study 1 and study 2 that the EventMiner framework can assist in detecting behavioral transitions and sequential patterns in process data. This framework can greatly reduce the analytical effort required for process theory building. As I discussed before, study 2 of Chapter 4 only addresses the question of how OSS development processes change over time. It is important to theoretically interpret those changes and patterns (Newman and Robey 1992; Robey and Newman 1996). In the future, I plan to perform “zoom in” and “zoom out” analysis. I will “zoom in” at transition points, to examine specific patterns before and after transitions. I will also zoom out to look at the bigger picture, by identifying general patterns in development events. Building on these analyses, I will theoretically interpret why those patterns occur. I will use the theory of distributed cognition as the theoretical lens with which to interpret the results. Robey and Newman suggested that the “form of the process model allows researchers who operate from different perspectives to enrich their understanding of the process of system development” (Robey and Newman 1996). They used five different theoretical perspectives to interpret the sequential patterns identified by their model in this 1996 study. Similarly, the transition patterns identified in my dissertation can be explored with and explained by different theoretical lenses.

I have interviewed several developers who have worked in at least one of the sample projects studied in this dissertation. I am planning to conduct a second round of interviews and collect more qualitative data from GitHub, such as issue contents and discussion contents. This data can provide richer details of the picture of development processes.

Secondly, I want to theoretically interpret the project clusters identified in study 2. I clustered OSS projects into five different groups based on their sequential patterns. A future research direction can be to provide project taxonomy based on development patterns. Furthermore, this study shows that projects in different clusters perform significantly differently. Therefore, it would be interesting to explore the relationship between certain sequential patterns and project performance.

Thirdly, I am currently obtaining more data for the empirical tests needed for the complete model in study 3 presented in Chapter 5. Currently, this model only captures the outcomes of development process routine characteristics (diversity and change). The new round of data analysis will help prepare the testing for the remaining relationships.

6.4 Contribution

6.4.1 Contribution to Practice

For OSS participants, a better understanding of their team's evolving trajectory, as well as what patterns of development activity sequence might indicate DCog effectiveness transitions, will help them better steer and control their development activities. OSS teams can use the tool EventMiner to automatically monitor OSS projects in order to detect transitions, and thus improve the performance. OSS project investors can apply the toolset to evaluate a project's previous progress and its ability to manage distributed cognitive tasks, and can even use the toolset to predict future success.

6.4.2 Contribution to Research

The findings of this study contribute to literature in OSS development, theory of distributed cognition, and design routines. Firstly, the study applies and expands the concept of distributed cognition to the context of OSS and investigates the phenomenon in a quantitative manner. Future studies can build on this concept and further investigate an OSS team's ability to manage distributed cognition processes. This concept can be also applied in other areas where work, knowledge, or artifacts are distributed. Secondly, it will be the first attempt to investigate OSS success from a process-based view. Previous studies have focused mostly on static attributes as determinants of success and have not looked at how previous development processes can indicate an OSS project's performance. This study will be the first study to investigate how OSS projects evolve through sequences of development events and explore evolving patterns of events associated with project performance. It will also be the first study to model dynamics of team development behavior in an OSS context. The theoretic findings on OSS project development events can contribute to general software development literature. Future research can extend the findings and develop theories on software development processes, such as how the sequence of the processes and their evolving patterns might relate to project success. Finally, it extends the current understanding of why and how OSS design routines change over time, and explores the effect of such diversity and changes on project performance. It also provides a novel perspective to understanding OSS development processes and predicts project performance by routine characteristics.

Methodically, this set of studies identifies, demonstrates, and validates appropriate data analysis methods for digital process data.

Finally, the resulting event-based analysis framework EventMiner can serve as both an open data repository and an open source toolkit for analyzing process data. This framework can facilitate research in OSS development process research and OSS evolution research in particular, and in process research in general.

Reference

Abbott, A. 1990. "A Primer on Sequence Methods," *Organization Science* (1:4), pp. 375-392.

Aggarwal, C., Han, J., Wang, J., and Yu, P. 2004. "On Demand Classification of Data Streams," ACM New York, NY, USA, pp. 503-508.

Aggarwal, C.C., Han, J., Wang, J., and Yu, P.S. 2003. "A Framework for Clustering Evolving Data Streams," VLDB Endowment, pp. 81-92.

Agrawal, R., and Psaila, G. 1995. "Active Data Mining."

Agrawal, R., and Srikant, R. 1995. "Mining Sequential Patterns," Published by the IEEE Computer Society, p. 3.

Aksulu, A., and Wade, M. 2010. "A Comprehensive Review and Synthesis of Open Source Research," *Journal of the Association for Information Systems* (11:11), pp. 576-656.

Alliance, F.C. 2001. "Incidence and Prevalence of the Major Causes of Brain Impairment." from http://www.caregiver.org/caregiver/jsp/content_node.jsp?nodeid=438

Arakji, R.Y., and Lang, K.R. 2007. "Digital Consumer Networks and Producer-Consumer Collaboration: Innovation and Product Development in the Digital Entertainment Industry," *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*: IEEE, pp. 211c-211c.

Ashby, W.R. 1956. "An Introduction to Cybernetics," *An introduction to cybernetics*.

Ayres, J., Flannick, J., Gehrke, J., and Yiu, T. 2002. "Sequential Pattern Mining Using a Bitmap Representation," ACM, pp. 429-435.

Baum, L.E., and Eagon, J. 1967. "An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology," *Bull. Amer. Math. Soc* (73:3), pp. 360-363.

Baum, L.E., Petrie, T., Soules, G., and Weiss, N. 1970. "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains," *The annals of mathematical statistics* (41:1), pp. 164-171.

Beecher, K., Capiluppi, A., and Boldyreff, C. 2009. "Identifying Exogenous Drivers and Evolutionary Stages in Floss Projects," *Journal of Systems and Software* (82:5), pp. 739-750.

Beer, S. 1975. *Platform for Change*. Wiley London.

Beer, S. 1979. *The Heart of Enterprise*. New York: Wiley Chichester.

Beer, S. 1981. *Brain of the Firm*. New York: Chichester

Belady, L.A., and Lehman, M.M. 1976. "A Model of Large Program Development," *IBM Systems Journal* (15:3), pp. 225-252.

Benbya, H., and Mckelvey, B. 2006. "Toward a Complexity Theory of Information Systems Development," *Information Technology & People* (19:1), pp. 12-34.

Bille, P. 2005. "A Survey on Tree Edit Distance and Related Problems," *Theoretical computer science* (337:1-3), pp. 217-239.

Bonaccorsi, A., and Rossi, C. 2003. "Why Open Source Software Can Succeed," *Research policy* (32:7), pp. 1243-1258.

Burton-Jones, A., Mclean, E.R., and Monod, E. 2014. "Theoretical Perspectives in IS Research: From Variance and Process to Conceptual Latitude and Conceptual Fit," *European Journal of Information Systems*.

Campbell, D.T. 1960. "Blind Variation and Selective Retentions in Creative Thought as in Other Knowledge Processes," *Psychological review* (67:6), p. 380.

Cant, S., Jeffery, D.R., and Henderson-Sellers, B. 1995. "A Conceptual Model of Cognitive Complexity of Elements of the Programming Process," *Information and Software Technology* (37:7), pp. 351-362.

Chakrabarti, S., Sarawagi, S., and Dom, B. 1998. "Mining Surprising Patterns Using Temporal Description Length," Citeseer, pp. 606-617.

Chandy, K., and Schulte, W.R. 2009. *Event Processing: Designing It Systems for Agile Companies*. McGraw-Hill Osborne Media.

Chang, L., Wang, T., Yang, D., and Luan, H. 2008. "Seqstream: Mining Closed Sequential Patterns over Stream Sliding Windows," IEEE, pp. 83-92.

Cho, S.B., and Park, H.J. 2003. "Efficient Anomaly Detection by Modeling Privilege Flows Using Hidden Markov Model," *Computers & Security* (22:1), pp. 45-55.

Christley, S., and Madey, G. 2007. "Analysis of Activity in the Open Source Software Development Community," *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*: IEEE, pp. 166b-166b.

Cohen, M.D., Burkhart, R., Dosi, G., Egidi, M., Marengo, L., Warglien, M., and Winter, S. 1996. "Routines and Other Recurring Action Patterns of Organizations: Contemporary Research Issues," *Industrial and corporate change* (5:3), pp. 653-698.

Comino, S., Manenti, F.M., and Parisi, M.L. 2007. "From Planning to Mature: On the Success of Open Source Projects," *Research policy* (36:10), pp. 1575-1586.

Conboy, K. 2009. "Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development," *Information Systems Research* (20:3), pp. 329-354.

Coverity. 2012. "Coverity Scan Open Source Report," *Coverity Scan Open Source Report*.

Crowston, K., Annabi, H., and Howison, J. 2003. "Defining Open Source Software Project Success."

Crowston, K., Annabi, H., Howison, J., and Masango, C. 2004. "Towards a Portfolio of Floss Project Success Measures."

Crowston, K., and Howison, J. 2006. "Hierarchy and Centralization in Free and Open Source Software Team Communications," *Knowledge, Technology & Policy* (18:4), pp. 65-85.

Crowston, K., Howison, J., and Annabi, H. 2006. "Information Systems Success in Free and Open Source Software Development: Theory and Measures," *Software Process: Improvement and Practice* (11:2), pp. 123-148.

Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., and Howison, J. 2007. "Self-Organization of Teams for Free/Libre Open Source Software Development," *Information and Software Technology* (49:6), pp. 564-575.

Crowston, K., and Scozzi, B. 2002. "Open Source Software Projects as Virtual Organisations: Competency Rallying for Software Development," IET, pp. 3-17.

Crowston, K., and Scozzi, B. 2008. "Bug Fixing Practices within Free/Libre Open Source Software Development Teams."

Cyert, R.M., and March, J.G. 1963. "A Behavioral Theory of the Firm," *Englewood Cliffs, NJ* (2).

Dabbish, L., Stuart, C., Tsay, J., and Herbsleb, J. 2012. "Social Coding in Github: Transparency and Collaboration in an Open Software Repository," *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*: ACM, pp. 1277-1286.

De Joode, E., Van Heugten, C., Verhey, F., and Van Boxtel, M. 2010. "Efficacy and Usability of Assistive Technology for Patients with Cognitive Deficits: A Systematic Review," *Clinical Rehabilitation* (24:8), p. 701.

Delone, W.H., and Mclean, E.R. 1992. "Information Systems Success: The Quest for the Dependent Variable," *Information systems research* (3:1), pp. 60-95.

Dierickx, I., and Cool, K. 1989. "Asset Stock Accumulation and Sustainability of Competitive Advantage," *Management science* (35:12), pp. 1504-1511.

Domingos, P., and Hulten, G. 2000. "Mining High-Speed Data Streams," ACM New York, NY, USA, pp. 71-80.

Dorst, K., Christianns, H., and Cross, N. 1996. "Analyzing Design Activity." Wiley West Sussex.

Elzinga, C.H., and Liefbroer, A.C. 2007. "De-Standardization of Family-Life Trajectories of Young Adults: A Cross-National Comparison Using Sequence Analysis," *European Journal of Population/Revue europ éenne de D émographie* (23:3-4), pp. 225-250.

Fang, Y., and Neufeld, D. 2009. "Understanding Sustained Participation in Open Source Software Projects," *Journal of Management Information Systems* (25:4), pp. 9-50.

Feldman, M.S. 2000. "Organizational Routines as a Source of Continuous Change," *Organization science* (11:6), pp. 611-629.

Feldman, M.S., and Pentland, B.T. 2003. "Reconceptualizing Organizational Routines as a Source of Flexibility and Change," *Administrative Science Quarterly* (48:1), pp. 94-118.

Ferrer-Troyano, F., Aguilar-Ruiz, J., and Riquelme, J. 2004. "Discovering Decision Rules from Numerical Data Streams," ACM New York, NY, USA, pp. 649-653.

Fickas, S., Robinson, W., and Sohlberg, M. 2005. "The Role of Deferred Requirements: A Case Study," *International Conference on Requirements Engineering (RE'05)*, Paris, France: IEEE.

Forney Jr, G.D. 1973. "The Viterbi Algorithm," *Proceedings of the IEEE* (61:3), pp. 268-278.

Frank. 2008. "Harness Networked Innovation," *Marketing Management* (17:5).

Frei, F.X., Kalakota, R., Leone, A.J., and Marx, L.M. 1999. "Process Variation as a Determinant of Bank Performance: Evidence from the Retail Banking Study," *Management Science* (45:9), pp. 1210-1220.

Gabadinho, A., Ritschard, G., Mueller, N.S., and Studer, M. 2011. "Analyzing and Visualizing State Sequences in R with Traminer," *Journal of Statistical Software* (40:4), pp. 1-37.

Gaber, M., Zaslavsky, A., and Krishnaswamy, S. 2005. "Mining Data Streams: A Review," *ACM Sigmod Record* (34:2), pp. 18-26.

Gacek, C., and Arief, B. 2004. "The Many Meanings of Open Source," *Software, IEEE* (21:1), pp. 34-40.

Galbraith, J.R. 1977. "Organization Design: An Information Processing View," *Organizational Effectiveness Center and School*, p. 21.

Gama, J. 2010. *Knowledge Discovery from Data Streams*. Boca Raton: Chapman & Hall/CRC.

Gama, J., Ganguly, A., Omitaomu, O., Vatsavai, R., and Gaber, M. 2009. "Knowledge Discovery from Data Streams," *Intelligent Data Analysis* (13:3), pp. 403-404.

Ganti, V., Gehrke, J., and Ramakrishnan, R. 2002a. "Mining Data Streams under Block Evolution," *ACM SIGKDD Explorations Newsletter* (3:2), pp. 1-10.

Ganti, V., Gehrke, J., Ramakrishnan, R., and Loh, W.Y. 2002b. "A Framework for Measuring Differences in Data Characteristics," *Journal of Computer and System Sciences* (64:3), pp. 542-578.

Gaskin, J., Berente, N., Lyytinen, K., and Yoo, Y. 2014. "Toward Generalizable Sociomaterial Inquiry: A Computational Approach for Zooming in and out of Sociomaterial Routines," *Mis Quarterly* (38:3), pp. 849-871.

Gaskin, J., Thummadi, V., Lyytinen, K., and Yoo, Y. 2011. "Digital Technology and the Variation in Design Routines: A Sequence Analysis of Four Design Processes."

Gaskin, J.E., Schutz, D.M., Berente, N., and Lyytinen, K. 2010. "The DNA of Design Work: Physical and Digital Materiality in Project-Based Design Organizations," *Academy of Management Proceedings*: Academy of Management, pp. 1-6.

Geiger, D., and Schröder, A. 2014. "Ever-Changing Routines? Toward a Revised Understanding of Organizational Routines between Rule-Following and Rule-Breaking," *Schmalenbach Business Review (SBR)* (66:2), pp. 170-190.

Ghosh, R.A., Glott, R., Krieger, B., and Robles, G. 2002. "Free/Libre and Open Source Software: Survey and Study." Maastricht Economic Research Institute on Innovation and Technology, University of Maastricht, The Netherlands, June.

Github.Com. "Github Help: About Stars." from https://help.github.com/articles/about-stars/

Giuri, P., Ploner, M., Rullani, F., and Torrisi, S. 2010. "Skills, Division of Labor and Performance in Collective Inventions: Evidence from Open Source Software," *International Journal of Industrial Organization* (28:1), pp. 54-68.

Godfrey, M., and Tu, Q. 2001. "Growth, Evolution, and Structural Change in Open Source Software," *Proceedings of the 4th international workshop on principles of software evolution*: ACM, pp. 103-106.

Godfrey, M.W., and Lee, E.H. 2000. "Secrets from the Monster: Extracting Mozilla’s Software Architecture," *Proceedings of Second Symposium on Constructing Software Engineering Tools (CoSET’00)*.

Godfrey, M.W., and Tu, Q. 2000. "Evolution in Open Source Software: A Case Study," *Software Maintenance, 2000. Proceedings. International Conference on*: IEEE, pp. 131-142.

Gousios, G., and Spinellis, D. 2012. "Ghtorrent: Github's Data from a Firehose," *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*: IEEE, pp. 12-21.

Grewal, R., Lilien, G.L., and Mallapragada, G. 2006. "Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems," *Management science* (52:7), p. 1043.

Guinan, P.J., Cooprider, J.G., and Faraj, S. 1998. "Enabling Software Development Team Performance During Requirements Definition: A Behavioral Versus Technical Approach," *Information Systems Research* (9:2), pp. 101-125.

Hambrick, D.C., Finkelstein, S., and Mooney, A.C. 2005. "Executive Job Demands: New Insights for Explaining Strategic Decisions and Leader Behaviors," *Academy of management review* (30:3), pp. 472-491.

Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. 2001. "Prefixspan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth," Citeseer, pp. 215-224.

Hansen, S., and Lyytinen, K. 2009. "Distributed Cognition in the Management of Design Requirements."

Hansen, S.W., Robinson, W.N., and Lyytinen, K.J. 2012a. "Computing Requirements: Cognitive Approaches to Distributed Requirements Engineering," *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 5224-5233.

Hansen, S.W., Robinson, W.N., and Lyytinen, K.J. 2012b. "Computing Requirements: Cognitive Approaches to Distributed Requirements Engineering " *Hawaii International Conference on Software Systems*, HI, USA: IEEE.

Herbsleb, J.D., and Grinter, R.E. 1999. "Splitting the Organization and Integrating the Code: Conway's Law Revisited," *Proceedings of the 21st International Conference on Software Engineering*: ACM, pp. 85-95.

Hertel, G., Konradt, U., and Orlikowski, B. 2004. "Managing Distance by Interdependence: Goal Setting, Task Interdependence, and Team-Based Rewards in Virtual Teams," *European Journal of Work and Organizational Psychology* (13:1), pp. 1-28.

Hertel, G., Niedner, S., and Herrmann, S. 2003. "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel," *Research policy* (32:7), pp. 1159-1177.

Hevner, A.R., March, S.T., Park, J., and Ram, S. 2004. "Design Science in Information Systems Research," *MIS Quarterly* (28:1), p. 75.

Hewitt, J., and Scardamalia, M. 1998. "Design Principles for Distributed Knowledge Building Processes," *Educational Psychology Review* (10:1), pp. 75-96.

Hirschheim, R., Klein, H., and Newman, M. 1991. "Information Systems Development as Social Action: Theoretical Perspective and Practice," *Omega* (19:6), pp. 587-608.

Hoang, X.D., Hu, J., and Bertok, P. 2003. "A Multi-Layer Model for Anomaly Intrusion Detection Using Program Sequences of System Calls," Citeseer.

Hollan, J., Hutchins, E., and Kirsh, D. 2000. "Distributed Cognition: Toward a New Foundation for Human-Computer Interaction Research," *ACM Transactions on Computer-Human Interaction (TOCHI)* (7:2), pp. 174-196.

Howison, J., Conklin, M., and Crowston, K. 2006. "Flossmole: A Collaborative Repository for Floss Research Data and Analyses," *International Journal of Information Technology and Web Engineering (IJITWE)* (1:3), pp. 17-26.

Howison, J., and Crowston, K. "Collaboration through Open Superposition," *MIS Quarterly*.

Hulten, G., Spencer, L., and Domingos, P. 2001. "Mining Time-Changing Data Streams," ACM New York, NY, USA, pp. 97-106.

Humphrey, W.S. 1989. *Managing the Software Process (Hardcover)*. Addison-Wesley Professional.

Hutchins, E. 1995. *Cognition in the Wild*. MIT press Cambridge, MA.

Hutchins, E., and Klausen, T. 1996. "Distributed Cognition in an Airline Cockpit," *Cognition and communication at work*, pp. 15-34.

Hutchins, E., and Lintern, G. 1995. *Cognition in the Wild*. MIT press Cambridge, MA.

Indyk, P., Koudas, N., and Muthukrishnan, S. 2000. "Identifying Representative Trends in Massive Time Series Data Sets Using Sketches," pp. 363-372.

Jain, G., Cook, D.J., and Jakkula, V. 2006. "Monitoring Health by Detecting Drifts and Outliers for a Smart Environment Inhabitant," in: *4th International Conference on Smart Homes and Health Telematics*.

Jarke, M., Loucopoulos, P., Lyytinen, K., Mylopoulos, J., and Robinson, W. 2011. "The Brave New World of Design Requirements," *Information Systems* (36:7), pp. 992-1008 (most downloaded, 2011).

Jensen, C., and Scacchi, W. 2005. "Process Modeling across the Web Information Infrastructure," *Software Process: Improvement and Practice* (10:3), pp. 255-272.

Johnson-Laird, P.N. 1989. *The Computer and the Mind: An Introduction to Cognitive Science*. Harvard University Press.

Joshi, S.S., and Phoha, V.V. 2005. "Investigating Hidden Markov Models Capabilities in Anomaly Detection," ACM, pp. 98-103.

Kifer, D., Ben-David, S., and Gehrke, J. 2004. "Detecting Change in Data Streams," VLDB Endowment, pp. 180-191.

King, A. 1998. "Transactive Peer Tutoring: Distributing Cognition and Metacognition," *Educational Psychology Review* (10:1), pp. 57-74.

Klarner, P., and Raisch, S. 2012. "Move to the Beat-Rhythms of Change and Firm Performance," *Academy of Management Journal*, p. amj. 2010.0767.

Kling, R. 1987. "Defining the Boundaries of Computing across Complex Organizations," *Critical issues in information systems research*: John Wiley & Sons, Inc., pp. 307-362.

Koch, S. 2004. "Profiling an Open Source Project Ecology and Its Programmers," *Electronic Markets* (14:2), pp. 77-88.

Koch, S. 2005. "Evolution of Open Source Software Systems–a Large-Scale Investigation," *Proceedings of the 1st International Conference on Open Source Systems*.

Kogut, B., and Metiu, A. 2001. "Open - Source Software Development and Distributed Innovation," *Oxford Review of Economic Policy* (17:2), pp. 248-264.

Krishnamurthy, S. 2002. "Cave or Community?."

Krishnamurthy, S. 2006. "On the Intrinsic and Extrinsic Motivation of Free/Libre/Open Source (Floss) Developers," *Knowledge, Technology & Policy* (18:4), pp. 17-39.

Kullback, S. 1997. *Information Theory and Statistics*. Dover Pubns.

Lane, T. 1999. "Hidden Markov Models for Human/Computer Interface Modeling," Citeseer, pp. 35-44.

Langley, A. 1999. "Strategies for Theorizing from Process Data," *Academy of Management review*, pp. 691-710.

Last, M. 2002. "Online Classification of Nonstationary Data Streams," *Intelligent Data Analysis* (6:2), pp. 129-147.

Laxman, S., and Sastry, P.S. 2006. "A Survey of Temporal Data Mining," *Sadhana* (31:2), pp. 173-198.

Lee, S.Y.T., Kim, H.W., and Gupta, S. 2009. "Measuring Open Source Software Success," *Omega* (37:2), pp. 426-438.

Lehman, M.M. 1980. "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE* (68:9), pp. 1060-1076.

Lehman, M.M., and Ramil, J.F. 2003. "Software Evolution—Background, Theory, Practice," *Information Processing Letters* (88:1), pp. 33-44.

Lerner, J. 2005. "The Scope of Open Source Licensing," *Journal of Law, Economics and Organization* (21:1).

Levitt, B., and March, J.G. 1988. "Organizational Learning," *Annual review of sociology*, pp. 319-340.

Lewin, A.Y., and Minton, J.W. 1986. "Determining Organizational Effectiveness: Another Look, and an Agenda for Research," *Management science* (32:5), pp. 514-538.

Lindberg, A. 2013. "Understanding Change in Open Source Communities: A Co-Evolutionary Framework," *Academy of Management Proceedings*: Academy of Management, p. 16619.

Lindberg, A., Berente, N., Howison, J., and Lyytinen, K. 2015a. "Variations in Information Processing Capacity: A Study of Routine Heterogeneity in Open Source Projects," in: *Academy of Management Meeting*. Vancouver, Canada.

Lindberg, A., Berente, N., and Lyytinen, K. 2015b. "Towards an Open Source Software Development Life Cycle: A Study of Routine Heterogeneity and Discourse across Multiple Releases," *Academy of Management Proceedings* Vancouver, Canada.

Lopresti, E., Mihailidis, A., and Kirsch, N. 2004. "Assistive Technology for Cognitive Rehabilitation: State of the Art," *Neuropsychological Rehabilitation* (14:1-2), pp. 5-39.

Lucas, H.C. 1981. *Implementation: The Key to Successful Information Systems*. Columbia University Press.

Lyytinen, K. 2009. "Data Matters in IS Theory Building," *Journal of the Association for Information Systems* (10:10), pp. 715-720.

Lyytinen, K., and Newman, M. 2008. "Explaining Information Systems Change: A Punctuated Socio-Technical Change Model," *European Journal of Information Systems* (17:6), pp. 589-613.

Mabroukeh, N.R., and Ezeife, C. 2010. "A Taxonomy of Sequential Pattern Mining Algorithms," *ACM Computing Surveys (CSUR)* (43:1), p. 3.

March, J.G., and Simon, H.A. 1958. "Organizations."

Markus, M.L., and Robey, D. 1988. "Information Technology and Organizational Change: Causal Structure in Theory and Research," *Management science* (34:5), pp. 583-598.

Mcafee, A., and Brynjolfsson, E. 2012. "Big Data: The Management Revolution," *Harvard business review* (90), pp. 60-66, 68, 128.

Mccoll, M., Carlson, P., Johnston, J., Minnes, P., Shue, K., Davies, D., and Karlovitz, T. 1998. "The Definition of Community Intergration: Perspectives of People with Brain Injuries," *Brain Injury* (12:1), pp. 15-30.

Mcdonald, N., and Goggins, S. 2013. "Performance and Participation in Open Source Software on Github," *CHI'13 Extended Abstracts on Human Factors in Computing Systems*: ACM, pp. 139-144.

Méndez-Durón, R., and García, C.E. 2009. "Returns from Social Capital in Open Source Software Networks," *Journal of Evolutionary Economics* (19:2), pp. 277-295.

Mockus, A., Fielding, R.T., and Herbsleb, J.D. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology (TOSEM)* (11:3), pp. 309-346.

Mohr, L.B. 1982. *Explaining Organizational Behavior*. Jossey-Bass San Francisco, CA.

Muffatto, M., and Faldani, M. 2003. "Open Source as a Complex Adaptive System," *Emergence* (5:3), pp. 83-100.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y. 2002. "Evolution Patterns of Open-Source Software Systems and Communities," *Proceedings of the international workshop on Principles of software evolution*: ACM, pp. 76-85.

Nelson, R., and Winter, S. 1982. *An Evolutionary Theory of Economic Change*. Cambridge: Belknap Press/Harvard University Press.

Newell, A. 1980. "Physical Symbol Systems," *Cognitive science* (4:2), pp. 135-183.

Newman, M., and Robey, D. 1992. "A Social Process Model of User-Analyst Relationships," *Mis Quarterly*, pp. 249-266.

Ourston, D., Matzner, S., Stump, W., and Hopkins, B. 2003. "Applications of Hidden Markov Models to Detecting Multi-Stage Network Attacks," IEEE, p. 10 pp.

Page, S.E. 2010. *Diversity and Complexity*. Princeton University Press.

Par é, G., Bourdeau, S., Marsan, J., Nach, H., and Shuraida, S. 2008. "Re-Examining the Causal Structure of Information Technology Impact Research," *European Journal of Information Systems* (17:4), pp. 403-416.

Pentland, B.T. 1995. "Grammatical Models of Organizational Processes," *Organization science* (6:5), pp. 541-556.

Pentland, B.T., Hærem, T., and Hillison, D. 2011. "The (N) Ever-Changing World: Stability and Change in Organizational Routines," *Organization Science* (22:6), pp. 1369-1383.

Pentland, B.T., and Rueter, H.H. 1994. "Organizational Routines as Grammars of Action," *Administrative Science Quarterly*, pp. 484-510.

Phua, C., Smith-Miles, K., Lee, V., and Gayler, R. 2007. "Adaptive Spike Detection for Resilient Data Stream Mining," *Proceedings of the sixth Australasian conference on Data mining and analytics-Volume 70*: Australian Computer Society, Inc., pp. 181-188.

Pree, W. 1994. "Meta Patterns—a Means for Capturing the Essentials of Reusable Object-Oriented Design," in *Object-Oriented Programming*. Springer, pp. 150-162.

Prochaska, J.O., Redding, C.A., and Evers, K. 1997. "The Transtheoretical Model and Stages of Change," *Health behavior an d health e—ducation. San Francisco*, pp. 61-83.

Qureshi, I., and Fang, Y. 2011. "Socialization in Open Source Software Projects: A Growth Mixture Modeling Approach," *Organizational Research Methods* (14:1), pp. 208-238.

Rabiner, L., and Juang, B. 1986. "An Introduction to Hidden Markov Models," *ASSP Magazine, IEEE* (3:1), pp. 4-16.

Rabiner, L.R. 1989. "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE* (77:2), pp. 257-286.

Raja, U., and Tretter, M.J. 2006. "Investigating Open Source Project Success: A Data Mining Approach to Model Formulation, Validation and Testing," *Investigating open source project success: a data mining approach to model formulation, validation and testing*.

Ravi Sen, S.S.S.a.S.B. 2012. "Open Source Software Success: Measures and Analysis."

Raymond, E. 1999. "The Cathedral and the Bazaar," *Knowledge, Technology & Policy* (12:3), pp. 23-49.

Reis, C.R., and De Mattos Fortes, R.P. 2002. "An Overview of the Software Engineering Process and Tools in the Mozilla Project."

Robbins, J.E., and Redmiles, D. 1996. "Software Architecture Design from the Perspective of Human Cognitive Needs," *Proceedings of the California Software Symposium (CSS'96)*, pp. 16-27.

Roberts, J.A., Hann, I.-H., and Slaughter, S.A. 2006. "Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects," *Management science* (52:7), pp. 984-999.

Robey, D., and Newman, M. 1996. "Sequential Patterns in Information Systems Development: An Application of a Social Process Model," *ACM Transactions on Information Systems (TOIS)* (14:1), pp. 30-63.

Robinson, W.N., and Akhlaghi, A. 2010. "Monitoring Behavioral Transitions in Cognitive Rehabilitation with Multi-Model, Multi-Window Stream Mining," *Hawaii International Conference on Software Systems*, Kauai, HI, USA: IEEE.

Robinson, W.N., Akhlaghi, A., and Deng, T. 2013a. "Transition Discovery of Sequential Behaviors in Email Application Usage Using Hidden Markov Models," *Hawaii International Conference on Software Systems*, HI, USA: IEEE, p. (best paper nominee).

Robinson, W.N., Akhlaghi, A., and Deng, T. 2013b. "Transition Discovery of Sequential Behaviors in Email Application Usage Using Hidden Markov Models," *46th Hawaii International Conference on System Sciences (HICSS)*, Maui, HI: IEEE, pp. 2656-2665.

Robinson, W.N., Akhlaghi, A., Deng, T., and Syed, A. 2011a. "Automated Differential Diagnosis of Behavioral Transitions in Stream Mining with Decision Trees," *Hawaii International Conference on Software Systems*, HI, USA: IEEE.

Robinson, W.N., Akhlaghi, A., and Syed, A. 2011b. "A Framework for Mining Behaviors in Clinical Data Streams," *Draft*, HI, USA: IEEE.

Robinson, W.N., and Deng, T. 2015. "Data Mining Behavioral Transitions in Open Source Repositories," *48th Hawaii International Conference on System Science (HICSS). Forthcoming*, Kauai, HI: IEEE.

Robinson, W.N., Syed, A.R., Akhlaghi, A., and Deng, T. 2012. "Pattern Discovery of User Interface Sequencing by Rehabilitation Clients with Cognitive Impairments," *Hawaii International Conference on Software Systems*, HI, USA: IEEE.

Rogers, Y., and Ellis, J. 1994. "Distributed Cognition: An Alternative Framework for Analysing and Explaining Collaborative Working," *Journal of information technology* (9), pp. 119-119.

Sabherwal, R., and Robey, D. 1993. "An Empirical Taxonomy of Implementation Processes Based on Sequences of Events in Information System Development," *Organization Science* (4:4), pp. 548-576.

Sabherwal, R., and Robey, D. 1995. "Reconciling Variance and Process Strategies for Studying Information System Development," *Information systems research* (6:4), pp. 303-327.

Salvato, C. 2009. "Capabilities Unveiled: The Role of Ordinary Activities in the Evolution of Product Development Processes," *Organization Science* (20:2), pp. 384-409.

Santos, C., Kuk, G., Kon, F., and Pearson, J. 2013. "The Attraction of Contributors in Free and Open Source Software Projects," *The Journal of Strategic Information Systems* (22:1), pp. 26-45.

Scacchi, W. 2002a. "Understanding the Requirements for Developing Open Source Software Systems," IET, pp. 24-39.

Scacchi, W. 2002b. "Understanding the Requirements for Developing Open Source Software Systems," *IEEE Proceedings - Software* (149:1), pp. 24-39.

Scacchi, W. 2004. "Free and Open Source Development Practices in the Game Community," *Software, IEEE* (21:1), pp. 59-66.

Scacchi, W. 2005. "Socio-Technical Interaction Networks in Free/Open Source Software Development Processes," in *Software Process Modeling*. Springer, pp. 1-27.

Scacchi, W. 2006. "Understanding Open Source Software Evolution," *Software Evolution and Feedback: Theory and Practice* (9), pp. 181-205.

Scacchi, W. 2009. *Understanding Requirements for Open Source Software*. Springer.

Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., and Lakhani, K. 2006. "Understanding Free/Open Source Software Development Processes," *Software Process: Improvement and Practice* (11:2), pp. 95-105.

Schroeder, R.G., Linderman, K., Liedtke, C., and Choo, A.S. 2008. "Six Sigma: Definition and Underlying Theory," *Journal of operations Management* (26:4), pp. 536-554.

Seddon. 1997. "A Respecification and Extension of the Delone and Mclean Model of IS Success."

Sen, R., Singh, S.S., and Borle, S. 2012. "Open Source Software Success: Measures and Analysis," *Decision Support Systems* (52:2), pp. 364-372.

Services, U.S.D.O.H.a.H. 2002. "Add Fact Sheet," Administration of Developmental Disabilities.

Shannon, C.E. 2001. "A Mathematical Theory of Communication," *ACM SIGMOBILE Mobile Computing and Communications Review* (5:1), pp. 3-55.

Silberschatz, A., and Tuzhilin, A. 1996. "What Makes Patterns Interesting in Knowledge Discovery Systems," *Knowledge and Data Engineering, IEEE Transactions on* (8:6), pp. 970-974.

Simon, H.A., and Kaplan, C.A. 1989. "Foundations of Cognitive Science."

Singh, P.V., Tan, Y., and Mookerjee, V. 2008. "Network Effects: The Influence of Structural Social Capital on Open Source Project Success," *SSRN eLibrary*.

Smith, N., Capiluppi, A., and Ramil, J. 2004. "Qualitative Analysis and Simulation of Open Source Software Evolution."

Snowdon, B., and Kawalek, P. 2003. "Active Meta-Process Models: A Conceptual Exposition," *Information and software Technology* (45:15), pp. 1021-1029.

Sohlberg, M., Ehlhardt, L., Fickas, S., and Sutcliffe, A. 2003a. "A Pilot Study Exploring Electronic (or E-Mail) Mail in Users with Acquired Cognitive-Linguistic Impairments," *Brain Injury* (17:7), pp. 609-629.

Sohlberg, M.M., Ehlhardt, L., Fickas, S., and Todis, B. 2002. "Core: Comprehensive Overview of Requisite E-Mail Skills," University of Oregon, Department of Computer and Information Science, Eugene, OR.

Sohlberg, M.M., Ehlhardt, L.A., Fickas, S., and Sutcliffe, A. 2003b. "A Pilot Study Exploring Electronic Mail in Users with Acquired Cognitive-Linguistic Impairments," *Brain Injury* (17:7), pp. 609-629.

Sohlberg, M.M., Fickas, S., Ehlhardt, L., and Todis, B. 2005a. "Case Study Report: The Longitudinal Effects of Accessible Email for Four Participants with Severe Cognitive Impairments.," *Journal of Aphasiology, in press*.

Sohlberg, M.M., Fickas, S., Ehlhardt, L., and Todis, B. 2005b. "The Longitudinal Effects of Accessible Email for Individuals with Severe Cognitive Impairments," *Aphasiology* (19:7), pp. 651-681.

Sohlberg, M.M., and Mateer, C.A. 1989. *Introduction to Cognitive Rehabilitation*. Guilford Press.

Sohlberg, M.M., and Mateer, C.A. 2001. *Cognitive Rehabilitation: An Integrated Neuropsychological Approach*. New York: Guilford Publication.

Srikant, R., and Agrawal, R. 1996. "Mining Sequential Patterns: Generalizations and Performance Improvements," *Advances in Database Technology—EDBT'96*, pp. 1-17.

Srivastava, A., Kundu, A., Sural, S., and Majumdar, A.K. 2008. "Credit Card Fraud Detection Using Hidden Markov Model," *Dependable and Secure Computing, IEEE Transactions on* (5:1), pp. 37-48.

Stamelos, I., Angelis, L., Oikonomou, A., and Bleris, G.L. 2002. "Code Quality Analysis in Open Source Software Development," *Information Systems Journal* (12:1), pp. 43-60.

Stewart, K.J., Ammeter, A.P., and Maruping, L.M. 2006a. "Impact of License Choice and Organizational Sponsorship on Success in Open Source Software Development Projects," *Information System Research* (17:2), pp. 126-144.

Stewart, K.J., Ammeter, A.P., and Maruping, L.M. 2006b. "Impacts of License Choice and Organizational Sponsorship on Success in Open Source Software Development Projects," *Information systems research* (17:2), pp. 126-144.

Stewart, K.J., and Ammeter, T. 2002. "An Exploratory Study of Factors Influencing the Level of Vitality and Popularity of Open Source Projects," Barcelona, pp. 853-857.

Stewart, K.J., and Gosain, S. 2006a. "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," *Mis Quarterly*, pp. 291-314.

Stewart, K.J., and Gosain, S. 2006b. "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," *Management Information Systems Quarterly* (30:2), p. 291.

Subramaniam, C., Sen, R., and Nelson, M.L. 2009. "Determinants of Open Source Software Project Success: A Longitudinal Study," *Decision Support Systems* (46:2), pp. 576-585.

Subramanyam, R., and Krishnan, M.S. 2003. "Empirical Analysis of Ck Metrics for Object-Oriented Design Complexity: Implications for Software Defects," *Software Engineering, IEEE Transactions on* (29:4), pp. 297-310.

Sutcliffe, A., Fickas, S., and Sohlberg, M.K.M. 2006. "Pc-Re: A Method for Personal and Contextual Requirements Engineering with Some Experience," *Requirements Engineering* (11:3), pp. 157-173.

Sutcliffe, A., Fickas, S., Sohlberg, M.M., and Ehlhardt, L.A. 2003. "Investigating the Usability of Assistive User Interfaces," *Interacting with Computers* (15), pp. 577-602.

Sutton, S.M. 2000. "The Role of Process in a Software Start-Up," *IEEE Software* (17:4), pp. 33-39.

Thummadi, B., Lyytinen, K., and Hansen, S. 2011. "Quality in Requirements Engineering (Re) Explained Using Distributed Cognition: A Case of Open Source Development."

Todis, B., Sohlberg, M.M., Hood, D., and Fickas, S. 2005. "Making Electronic Mail Accessible: Perspectives of People with Acquired Cognitive Impairments, Caregivers and Professionals," *Brain Injury* (19:6), pp. 389-402.

Tu, Q., and Godfrey, M.W. 2001. "The Build-Time Software Architecture View," *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*: IEEE Computer Society, p. 398.

Van Aardt, A. 2004. "Open Source Software Development as Complex Adaptive Systems," *Proceedings of the 17th Annual Conference of the National Advisory Committee on Computing Qualifications, Christchurch, New Zealand*, pp. 6-9.

Van De Ven, A.H. 1992. "Suggestions for Studying Strategy Process: A Research Note," *Strategic management journal* (13:5), pp. 169-188.

Van De Ven, A.H. 2007. *Engaged Scholarship: A Guide for Organizational and Social Research: A Guide for Organizational and Social Research*. Oxford University Press.

Van De Ven, A.H., and Poole, M.S. 1990. "Methods for Studying Innovation Development in the Minnesota Innovation Research Program," *Organization science* (1:3), pp. 313-335.

Virone, G., Alwan, M., Dalal, S., Kell, S.W., Turner, B., Stankovic, J.A., and Felder, R. 2008. "Behavioral Patterns of Older Adults in Assisted Living," *Information Technology in Biomedicine, IEEE Transactions on* (12:3), pp. 387-398.

Von Krogh, G., Spaeth, S., and Lakhani, K.R. 2003. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," *Research Policy* (32:7), pp. 1217-1241.

Wang, J., and Han, J. 2004. "Bide: Efficient Mining of Frequent Closed Sequences," IEEE, pp. 79-90.

Wang, Y. 2007. "Prediction of Success in Open Source Software Development." UNIVERSITY OF CALIFORNIA.

Weick, K.E., and Kiesler, C.A. 1979. *The Social Psychology of Organizing*. Random House New York.

Wilson, B.A., Emslie, H.C., Quirk, K., and Evans, J.J. 2001. "Reducing Everyday Memory and Planning Problems by Means of a Paging System: A Randomised Control Crossover Study," *Journal of Neurology, Neurosurgery, and Psychiatry* (70:4), pp. 477-482.

Winograd, T. 1987. "A Language/Action Perspective on the Design of Cooperative Work," *Human–Computer Interaction* (3:1), pp. 3-30.

Wright, P., Rogers, N., Hall, C., Wilson, B., Evans, J., Emslie, H., and Bartram, C. 2001. "Comparison of Pocket-Computer Memory Aids for People with Brain Injury," *Brain Injury* (15:9), pp. 787-800.

Wright, P.C., Fields, R.E., and Harrison, M.D. 2000. "Analyzing Human-Computer Interaction as Distributed Cognition: The Resources Model," *Human-Computer Interaction* (15:1), pp. 1-41.

Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., Mclachlan, G., Ng, A., Liu, B., and Yu, P. 2008. "Top 10 Algorithms in Data Mining," *Knowledge and Information Systems* (14:1), pp. 1-37.

Yan, X., Han, J., and Afshar, R. 2003. "Clospan: Mining Closed Sequential Patterns in Large Datasets," pp. 166-177.

Ye, Y., and Kishida, K. 2003. "Toward an Understanding of the Motivation of Open Source Software Developers," *Software Engineering, 2003. Proceedings. 25th International Conference on*: IEEE, pp. 419-429.

Yourdon, E. 2003. *Death March*. Pearson Education.

Zaki, M.J. 2001. "Spade: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning* (42:1), pp. 31-60.

Zhao, Q., and Bhowmick, S.S. 2003. "Sequential Pattern Mining: A Survey," *ITechnical Report CAIS Nayang Technological University Singapore*, pp. 1–26.

Appendix

Table A1: DCog Metric Table

	Definition	Metric
Social distribution	Distribution of social actors among the projects	domain knowledge distribution, application knowledge distribution, decision making distribution, role distribution, task focus distribution, distribution of access to information, communication channels, domain of use, reference system distribution(requirement related), constraint sources(RE related), percentage of co-development
Structural distribution	Distribution of internal and external(material or environmental) structure	**Internal:** Representations distribution in the internal minds of the developers; **External:** Artifacts and ecology including(emails, forums, threads, chats, documents, CVS, Mock-ups/Prototypes), existing platforms(programs)(RE related), system models(RE related)
Temporal*	Outcome of earlier actions influence the cognitive processes enacted in later efforts	*Interaction between people and artifacts for encoding/retrieving (RE) knowledge, interaction of the people/artifacts to retrieve/encode archival (RE knowledge), interaction mediated by computers instead of face-to-face, Use of external consultants, reliance upon the higher education user group forum for insights from earlier PeopleSoft initiatives

Table A2. OSS Success Constructs and Measurements

Category	Construct	Source	Measurement
Project Output	User satisfaction	(Lee et al. 2009)	User ratings, opinions expressed on project
	software quality	(Lee et al. 2009)	User ratings
	perceived ease of use and useful-ness	(Lee et al. 2009)	User ratings
	use and user interest/popularity	(Crowston et al. 2003; Crowston et al. 2004) (Crowston and Scozzi 2002; Lee et al. 2009), Grewal, Lilien et al. 2006, Stewart et all.06, (Méndez-Durón and García 2009) Subramaniam, Sen et al. 2009)	number of downloads, number of page views, number of subscribers, change of number subscribers

	community service quality	(Lee et al. 2009)	User ratings
	project/development status	(Comino et al. 2007) (Subramaniam et al. 2009), (Crowston and Scozzi 2002)	Status in the six stage model(planning, pre-alpha, alpha, beta, stable)
	Project Completion	(Crowston and Scozzi 2002),(Giuri et al. 2010; Mockus et al. 2002),(Stewart and Gosain 2006b)	number of release, number of bugs fixed, defect density, size achieved, number of commits, achieved identified goals
	Developer satisfaction	(Crowston and Scozzi 2002), (Crowston et al. 2006), (Ghosh et al. 2002) (Hertel et al. 2004)	Developer ratings
Process	Level of activity	(Subramaniam et al. 2009),(Stewart and Gosain 2006b),(Crowston and Scozzi 2002),(Stewart et al. 2006b),(Beecher et al. 2009; Crowston et al. 2003; Grewal et al. 2006)	Number of files released, number of bugs fixed, number of CVS commits, Proportion of bugs fixed
	Number of developers	(Beecher et al. 2009; Crowston et al. 2006; Stewart et al. 2006b; Stewart and Gosain 2006b; Subramaniam et al. 2009)	Number of developers
Process	Cycle Time	(Stewart and Ammeter 2002), (Crowston et al. 2006),	Time taken to fix the bugs, movement from alpha to beta to stable, time between releases, time to implement features
Outcomes for project members	Individual Impact	(Lee et al. 2009), (Crowston et al. 2003; Crowston et al. 2004; Crowston et al. 2006)	Individual job opportunities and salary, Individual reputation

	Knowledge creation	(Singh et al. 2008), (Lee et al. 2009), (Crowston et al. 2003; Crowston et al. 2004; Crowston et al. 2006)	New procedural and programming skills, improvement on knowledge, skill, productivity, performance

Table A3. Pseudo Codes for Tree Differencing

```
        levelOrder

        public int levelorder (Tree tree, LinkedList<Edge> resultTree, LinkedList<String>
        resultString, LinkedList<String> treeMatrix, LinkedList<Integer> matricsList) {
                {
        LinkedList<Edge> q = new LinkedList();
        Iterator<Edge> iter = tree.childIterator();
        While(the tree still has childInterator)
        {
                Push iter.next to the linkedlist;
        }
        .....
        While(the LinkedList q is not empty)
        {
        Poll an edge from q;
        Get the child tree of this edge;
        Initialize an childIterator for the child tree;
        While(the tree still has childInterator)
        {
                Push iter.next to the LinkedList q;
        }
        }
        ...
}
        treeDiff

        public int treeDiff(LinkedList<Edge> arraya, LinkedList<Edge> arrayb,
        LinkedList<NewEdge> depthTreeA,LinkedList<NewEdge>
        depthTreeB,LinkedList<String> DiffColumn)
        {
                While(arraya and arrayb are not empty)
        {
                Compare each item of arraya and arrayb;
                Stop when the first different edge is found;
        }

        }
```

Table A4. Git Events

Event/Activity	Description
CommitCommentEvent	A comment is posted to a commit
CreateEvent	A repository or a readme or a branch is created
DeleteEvent	A branch or a tag or a repository is deleted
DownloadEvent	Repository is downloaded
FollowEvent	A user starts to follow another user on GitHub
ForkEvent	A user forks a repository
ForkApplyEvent	A patch is applied in the fork queue
GistEvent	A gist is created
GollumEvent	A Wiki page is created or updated
IssueCommentEvent	A comment is created to an issue
IssuesEvent	An issue is created or closed
MemberEvent	Add a member to the repository
PublicEvent	A repository is open sourced
PullRequestEvent	A user notifies others about changes he has pushed to a GitHub repository
PullRequestReviewComment Event	A review comment made to the pull requests is posted
PushEvent	User submits a commit to a repository
TeamAddEvent	A user is added to a team, or a repository is added to a team
WatchEvent	A user subscribe to a repository to get its updates

Table A5. List of Projects and Associated Cluster

Cluster #	Projects
0	ActionBarSherlock, AFNetworking, android-bootstrap, authlogic, AwesomeMenu, backbone-boilerplate, capistrano, chosen, courser, docker, fabric.js, Font-Awesome, gitlabhq, GMGridView, history.js, jasmine, KineticJS, less.js, moment, netty, TimelineJS, phantomjs, platform_frameworks_base, SlidingMenu, wysihtml5, socket.io, storm, rubinius
1	android-bootstrap, annotated_redis_source, annotated_redis_source, async, atom, backbone-fundamentals, brackets, cocos2d-html5, CodeIgniter-Ion-Auth, colour-schemes, ember.js, grunt, hackathon-starter, handlebars.js, highlight.js, intro.js, jade, javascript-patterns, Jekyll, jquery-pjax, jquerytools, jScrollPane, masonry, Ghost, Modernizr, MWFeedParser, zepto, NewsBlur, normalize.css, phonegap-plugins, statsd, OpenTLD, reddit, underscore, Vundle.vim, raphael, sizzle, resque, retire, tag-it
2	Android-ViewPagerIndicator, AngularJS-Learning, async, bash-it, coffeescript, compass, fastclick, FlatUIKit, idiomatic.js, jQuery-menu-aim, libgdx, metrics, onepage-scroll, parallax, Probabilistic-Programming-and-Bayesian-Methods-for-Hackers, ProjectTox-Core, ratchet, pure, ReactiveCocoa, x-editable, typeahead.js
3	cw-omnibus, elasticsearch, flight, jqGrid, meteor, node-webkit
4	bootstrap-sass, devise, discourse, Front-end-Developer-Interview-Questions, gitflow, guzzle, Semantic-UI, Telescope

Table A6. Summary Statistics of Routines

	Issue Handling Routines	**Pull Request Handling Routines**	**Reopen Routines**
Number	68,095	21,776	3,117
Average Duration	53 days	17 days	46 days
Average number of Comments	2.68	2.27	3.83
Number of Unique Actors	2.7	2.5	2.6
Number of Events	5.4	5.7	7.2