Fall 2017

Process Mining Concepts for Discovering User Behavioral Patterns in Instrumented Software

Kevin Olson

PROCESS MINING CONCEPTS FOR DISCOVERING USER BEHAVIORAL PATTERNS IN INSTRUMENTED SOFTWARE

A THESIS

SUBMITTED ON 14 OF DECEMBER, 2017

TO THE DEPARTMENT OF INFORMATION TECHNOLOGY

OF THE COLLEGE OF COMPUTER & INFORMATION SCIENCES

OF REGIS UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF MASTER OF SCIENCE IN SOFTWARE ENGINEERING

BY

Kevin Harold Olson

APPROVALS

Kevin Pyatt, Ph. D., Thesis Advisor

Brian Lawler

Ishmael Thomas

Abstract

Process Mining is a technique for discovering “in-use” processes from traces emitted to event logs. Researchers have recently explored applying this technique to documenting processes discovered in software applications. However, the requirements for emitting events to support Process Mining against software applications have not been well documented. Furthermore, the linking of end-user intentional behavior to software quality as demonstrated in the discovered processes has not been well articulated. After evaluating the literature, this thesis suggested focusing on user goals and actual, in-use processes as an input to an Agile software development life cycle in order to improve software quality. It also provided suggestions for instrumenting software applications to support Process Mining techniques.

Acknowledgements

The author gratefully thanks his advisor, Kevin Pyatt, for his patience and guidance in reviewing and providing improvements to this thesis.

The author also thanks his wife, Shawn, for her proofreading, suggestions, and being a sounding board for ideas and rants. Her love and support were instrumental in winding this project towards its conclusion.

Finally, the author thanks his parents for all the assistance they have provided over the years. They have served as inspirations for achieving goals in life, and for demonstrating that all individuals are worthy of respect even when the majority may wish to relegate them to obscurity.

Table of Contents

Abstract 2

Acknowledgements 3

Table of Contents 4

List of Figures 7

List of Tables 8

Chapter 1: Introduction 9

- Alice and the Dilemma 11
- Software Is Not Just Features 12
- Software Employs Implicit Processes 15
- A Potential Solution 16
- Contributions and Research Questions 18
- Structure 20
- Terminology 21
- The Need to Better Understand the Users 25
- Can Agile Resolve the Situation? 29
- Why Understanding User Behavior Matters for Software Engineering 33
- A Repeatable Approach for Software Use Understanding 39
- Summary 40

Chapter 2: Literature Review 42

The First Thread – Process Discovery 44

Second Thread – Knowledge Discovery in Databases 45

Third Thread – Workflow Mining 48

Additional Contributions to the Third Thread 52

Early Exploration Summary 53

Process Mining Types 56

Maturation 59

Maturation Summary 64

Process Mining for Software 65

Summary 72

Chapter 3: Considerations for Event Log Creation 74

Example Event Log 77

Definition of a Case 79

Considerations of Activities 89

Instrumenting vs. Interception 96

Log Location and Persistence 102

Privacy Concerns 107

Conclusion 109

Chapter 4: Example Application of Process Mining to a Software Application 111

Chapter 5: Conclusion 123

Research Questions and Results 124

Support For Software Quality 126

DevOps 127

Future Research 127

Other Considerations 129

References 133

List of Figures

Figure 1. Process Mining Types. From van der Aalst et al. (2012). 56

Figure 2. Example Petri Net. From van der Aalst and Weijters (2004)..... 78

Figure 3. Structuredness and data-driven vs. process driven application. From van Dongen and van der Aalst (2005)..... 82

Figure 4. Event Logging to Remote Service. From (Rubin, Mitsyuk, et al., 2014, p. 7) 103

Figure 5. Sandia Analysis Workbench Problem Domain 112

Figure 6. Sandia Analysis Workbench Application. 113

Figure 7. Spaghetti Process from Cluster Analysis of SAW Usage 117

Figure 8. Petri Net of SAW Usage..... 118

Figure 9. Heuristic Miner Results..... 119

Figure 10. Fuzzy Miner Results..... 120

List of Tables

Table 1. CQI Quality Survey and Quality Nature..... 37

Table 2. Example Entries for an Event Log..... 77

Chapter 1: Introduction

When a user interacts with a software application, it is possible to discern behavioral patterns. These patterns reveal the user's intentions, as well as the processes by which the user traverses the application's functionality to achieve desired outcomes. It is insufficient to restrict analyzing a software application to the expected means of interaction because the user's expressed behavior is frequently more diverse than anticipated by developers. This divergence arises due to the dominant development paradigm that conceptualizes software as a collection of features. Instead of assuming a particular utilization of features, however, it is possible to understand the behavior of people when one thinks "about it in terms of their goals, needs, and motives" (as cited in Zakia, 1995). Rather than attempting to enumerate feature utilization in the abstract, it is important to understand the goal-revealing processes of users as demonstrated through their actual interaction with the software. Thus, one must couple the processes an individual follows when using an application with the individual's intentional outcomes. High quality software addresses the question of "can the user accomplish their [sic] goal?" (Summers, 2014). Consequently, quality software supports the efficient attainment of goals by individual users.

The primary argument of this thesis is the need to discover the actual, in-use processes that reveal intentional behavior, and to use this information to improve software quality. There are several reasons to believe that discovering intentional behavior can improve software quality. First, there is reasonable support in the literature to believe that a focus on goals, rather than on feature articulation, is the most appropriate locus of concern for software developers. Focusing on goals, however, is not typically the way in which an individual conceptualizes software; software is often described as a checklist of features divorced from actual desired outcomes. Therefore, this locus suggests a need for a paradigmatic change in how software – and therefore

software quality – is understood. Second, individuals' goals are revealed in a software application though the implicit processes in which individuals engage. By instrumenting a software package, one may collect and then analyze the processes by which individuals use a software application. Such a collection can reveal the ultimate goals individuals seek. Furthermore, these goals are *individually* based, and not necessarily wholly congruent with the goals as assumed or specified by stakeholders. There are numerous reasons for this lack of congruency, but stakeholders are, at best, representative of the user community and not the community itself. Third, Process Mining is a viable, defined, and repeatable methodology that has the potential to be applied to a software application in order to reveal user behavior. Process Mining is particularly useful in the "context of human-centric processes for which an information system does not enforce the activities to be carried out in a particular order" (Goedertier, De Weerdt, Martens, Vanthienen, & Baesens, 2011, p. 1698). This methodology provides data that may be analyzed for a variety of purposes. Process Mining is not a software development life cycle (SDLC) methodology; it is a methodology to support collecting data about process utilization. Finally, by understanding the actual software utilization and goals, software quality may be improved by incorporating data gathered into a SDLC.

The scope of this thesis, however, focuses on a definitional gap in the Process Mining literature. This gap concerns the criteria necessary in order to apply Process Mining to a software application. In general, this gap is a result of the history of Process Mining being primarily applied to process-aware workflow systems. It is also associated with the challenges of developing algorithms to recover actual processes. As noted in Chapter 2, the literature is moving from a focus on static, pre-defined processes towards broader considerations of user behavior exhibited in *ad-hoc* processes. This change in the Process Mining approach closely

parallels the argument developed in this thesis for the need to change from static, feature-based conceptualizations of software to an understanding of software as an artifact to support goal attainment though diverse, implicit, and individual processes.

As a means of introducing the topic of Process Mining and its application to software, an illustrative story is presented. The Alice scenario is drawn from an actual conundrum faced by an engineer at a Fortune 50 company. The story illustrates the current dependency of users on updates driven by the software development community. It also posits that the utilization of a software application varies depending upon the particular set of users. In addition, the story reveals the typical paradigmatic thinking that focuses on features rather than goals.

Alice and the Dilemma

Alice works for a manufacturing company that uses computational simulation software to verify and validate the safety of a delivered product. In essence, Alice uses software to simulate what would happen if an unexpected event — such as dropping a piece of equipment from some height — were to occur. As a part of this process, the simulation software itself must be validated. After spending a couple of months validating the new 2.0 version of the simulation software for acceptance by the company, Alice receives an e-mail informing her that version 2.01 is available, and it encourages her to upgrade. A question arises: should Alice invest the time to re-validate the simulation software, or continue with the 2.0 version?

One would think it possible to answer Alice's question with data relevant to her. Unfortunately, the standard software documentation consisting of a "User's Guide," "Release Notes," or lists of bug fixes (or resolved User Stories) does not directly address her concerns. After all, from Alice's perspective it intuitively seems that the upgrade is worth accepting only if a particular fault were resolved that addressed a feature she used in the 2.0 version. If version

2.01 adds features she does not use, or fixes an error in a feature she does not use, then the upgrade is likely not worth the investment. Unfortunately, without detailed information about what features Alice uses, and what features are fixed or added in the software upgrade, it is not possible to answer Alice's question using data. In fact, most software today cannot answer these questions in general, and certainly not in specific customer instances. Such situations led van Genuchten, Mans, Reijers, and Wismeijer (2014) to suggest that software vendors must change their approach to software updates by convincing users of "the necessity of an upgrade" (p. 99). According to these authors, an upgrade's value should be demonstrated through data, rather than marketing literature. In effect, the argument is to replace the existing push model of updates, wherein a release is driven by the software development company's needs, with a pull model of updating when the update is of demonstrable benefit to the end-user. As will be shown below, properly instrumented software may be analyzed through a specific methodology to address Alice's question.

For organizations that must maintain approved software, accepting updates imposes a burden (see, e.g., Ali, 2012; Bloch, 2013; Pogue, 2015; Sahin & Zahedi, 2001). This burden exists since the software must be verified prior to deployment. Conversely, not accepting the updates can create technical debt (see, e.g., Brown et al., 2010; Kim, Behr, & Spafford, 2013). This debt must be paid for in the future. Data that indicated whether a particular update directly addressed the needs of an organization could help resolve the debate about when to accept an update.

Software Is Not Just Features

The Alice story, however, reveals an implicit paradigm about software. A feature is "a distinguishing characteristic of a system item" (IEEE, 2008) or "a chunk of functionality that

delivers business value" (VersionOne, n.d.). It is the "unit of functionality of a software system that satisfies a requirement, represents a design decision, and provides a potential configuration option" (Apel & Kästner, 2009). A given collection of features composes a software application. Features are "abstractions that shape the reasoning of engineers and other stakeholders" (Classen, Heymans, & Schobbens, 2008, p. 16). Features are the basis of a worldview about how software is estimated, managed, modeled, developed, and marketed; they are the basic linguistic construct for discussing software.

However, as Cooper, Reimann, and Cronin (2007) argued, the purpose of software is to enable users to achieve goals. A goal is "an expectation of an end condition" (Chapter 1, "Goals versus tasks", para. 1) that has value to the user. Khodabandelou, Hug, Deneckère, and Salinesi (2014) suggested a similar definition with the additional need for a "clear-cut criteria of satisfaction, which can be fulfilled by the enactment of a process." Software exists not for providing features *per se*, since such features tend to be "activities and tasks [that] are intermediate steps" (Cooper et al., 2007, Chapter 1, "Goals versus tasks", para. 1). Instead, software should enable users to achieve an outcome that is of value to them via the software's implemented functionality. When the story speaks of the "features" Alice uses, it reduces the desired outcome (simulating a product's response to varying conditions) to a list of functionality divorced from a goal.

The Alice story is not the only attempt to illustrate the issues created by focusing on features rather than goals. Magnacca (2009) provided a different analogy by considering an individual purchasing a drill from a hardware store. Generally, if one asks a store employee for assistance in buying a drill, the probable response is a list of features. Yet the goal of the user is to make holes, and the feature set of the drill is a means by which to achieve the goal. Like

Cooper et al. (2007), Magnacca argued that a focus on features is misplaced, with the ultimate needs and processes by which an individual achieves a goal being subsumed by an emphasis on feature articulation. To a certain extent, Norman (2002) made a similar argument in suggesting that engineering constraints outweigh a focus on ensuring individuals can effectively achieve their goals. The Agile Manifesto (Fowler & Highsmith, 2001) used an analogy of car buying, and noted that many people conceptualize software as a list of features for which they pay. The problem, the Manifesto suggested, is that the analogy does not work for software. Instead, there must be a focus on value delivery as understood though actual, usable software. Ultimately, software usability implies the achievement of goals (Ferre, Juristo, & Moreno, 2004).

The focus on goals rather than features in software shifts the discussion from static functionality to dynamic user behavior. In understanding user behavior, one can also understand the functionality that is used in a software program. The inverse understanding, however, is not obtainable since it would attempt to derive the dynamic utilization from static definitions. Focusing on dynamic user behavior illuminates the ways in which software is actually utilized by individuals. In contrast, a focus on features *assumes* the means by which the software is used.

As Magnacca illustrated, and as the Alice story assumes, focusing on features provides insight into the means by which something may be achieved. In contrast, understanding why something is done provides a richer explanation. Addressing questions of why an individual utilizes software functionality in a particular way illuminates opportunities for software improvement. Such questions can provide support to answer the “is the upgrade worth it question,” since the focus on user behavior demonstrates whether the purported value of the upgrade meets the usage patterns of the particular individual. Addressing the issue of behavior, however, moves beyond the question of upgrade value and into the realm of software quality.

Software Employs Implicit Processes

At the base of the discussion regarding goal achievement via tasks (features) is the realization that all software has within it implicit processes. As an example, consider a word processor. One user might launch the program, open a document, add text, insert tabs and carriage returns for formatting, save the file, and exit the program. Another user might launch the program with a given file, add text, format the document using style sheets, save the file, and exit. In both cases, the word processor allowed the users to accomplish their specific goal of a formatted document. However, the users achieved their goals through different processes and features. Software that provided improvements to style sheets, for example, is not a worthwhile upgrade for those users who did not use style sheets. However, even for those users that utilized style sheets, capturing the specific program utilization can indicate how much value may be obtained. Moreover, this particular value could be calculated for each individual user based upon the specific way in which a given user achieved their goals through the application's pathways. In short, software allows individuals to achieve goals, but the ways in which individuals utilize software is not uniform. Further, an application's utilization may diverge from the way software designers and developers expect. This divergence is not necessarily a bad thing, since it may reveal additional value in the software.

In van der Aalst (2012c), the author suggested that most approaches to understanding processes are derived from a single, standard process-model. Applied to software, this single process model would likely equate to a UML sequence diagram that attempted to capture the interaction. However, van der Aalst argued that such single models fail when actors have a variety of purposes they may wish to achieve. Cooper et al. (2007) made a similar argument when they suggested that different communities involved in various aspects of software development favor different model types. In addition, maintaining and reconciling these different

models is difficult and error prone. Worse, these models tend to not be maintained, resulting in a false paradigm on the part of the developers with respect to how the software is actually utilized.

The single model approach that is common in software development relies upon another underlying assumption. This assumption states that the value of a particular set of features is uniform for all users. However, while it is true that software utilization is expressed through the particular sequence of features that are invoked, it is not merely selecting a sequence of features; rather, it is utilizing software in a particular way to achieve some desired end-goal. When one assumes software is a set of features, the user's goals are lost. As such, value can only be measured in terms of how well the software assists users in meeting their goals.

When van Genuchten et al. (2014) suggested quantifying the value of an upgrade, there was an implicit assumption that the quantified value is the same for all users. In other words, the article assumed the goals of all users are the same, and thus a single number (or perhaps set of numbers, though the article does not consider such a case) could be derived to indicate the upgrade value. However, as the Alice story shows, the problem is that the entire user community's goals cannot be assumed to be uniform since the particular goals of Alice's organization may be different than for other organizations. While van Genuchten et al. (2014) did suggest the importance of understanding how users interact with the software, it did not consider variations in the user community writ large.

A Potential Solution

Understanding how users interact with software provides multiple benefits to developers. For deployed software, it can assist with providing insight into the value of an upgrade. It can also assist with improving the understanding of the software to support new or refined functionality. For software that is under development, it can improve the developer's mental

models, allowing for the identification of divergence between expected and actual user behavior. This understanding can highlight the importance of certain software errors, or even reveal errors. It can also allow for refining the approach to support a more efficient goal achievement, or the recognition of valuable goals that were not understood in the initial application definition. Furthermore, interacting with software is also a voyage of discovery, in that software should not only allow for achieving some set of identified goals, but should also inspire new goals and suggest new ways to achieve them (Brooks, 1995; Raymond, 2008). In short, collecting information regarding user behavior may be done in multiple phases of a software development lifecycle (SDLC), and this information may be used to improve the software application.

Nonetheless, in order to facilitate this understanding, a methodology for collecting and analyzing the users' processes is necessary. One promising approach for illuminating user behavior is found in the application of Process Mining to software applications. Process Mining seeks to "discover, monitor, and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs" (van der Aalst et al., 2012, p. 1). By applying algorithms to collected data, it is possible to reconstruct users' processes, and thereby illuminate how they actually use the software. With sufficient costing data (e.g., timing, effort, etc.), it is possible to quantify users' activities, providing the answer to upgrade value, potentially on a per user basis. It is also possible to enumerate the functional points executed by the software, thus addressing Alice's issues. Process Mining, therefore, provides an avenue to assist with understanding user behavior and discovering their intentions. This understanding may be applied in a myriad of ways to support software improvement.

Process Mining is a technique that can discover the gap between assumed and actual utilization of the software. By analyzing event logs generated from the runtime utilization of the

software, one can discover the in-use processes (as opposed to the assumed normative ones) that users follow. Using Process Mining allows for a defined approach for uncovering the goals of the users (addressing, e.g., Cooper et al., 2007). It also provides for measurement and improvement during a software application's development and deployment phases. Process Mining can also answer specific value-based questions about particular end-users (the Alice story and van Genuchten et al., 2014).

Applying Process Mining to the comprehension of a software package in order to illuminate users' goals is a relatively novel approach. As will be seen in Chapter 2, there has been a dearth of literature analyzing an executing software program to highlight the implicit processes of users. Furthermore, the linking of such an approach to a general understanding of software quality appears to be, at best assumed — if not absent — in the literature. In general, when the literature has examined applying Process Mining to software, it has frequently examined the software development process rather than the actual software application. As an example of the former, consider Zimmermann, Weisgerber, Diehl, and Zeller (2004); Huo, Zhang, and Jeffery (2006); Cleland-Huang and Mobasher (2008); Duan, Cleland-Huang, and Mobasher (2008); Sun, Du, Chen, Khoo, and Yang (2013); and Gupta (2014; Gupta & Sureka, 2014; Gupta, Sureka, & Padmanabhuni, 2014). In contrast, the latter is represented essentially by Khodabandelou et al. (2014); Rubin, Lomazova, and van der Aalst (2014); Rubin, Mitsyuk, Lomazova, and van der Aalst (2014); and van Genuchten et al. (2014).

Contributions and Research Questions

This thesis makes four contributions to the overall Process Mining literature. First, it explicitly links the concept of user goals to the implicit software processes used to achieve these goals. This argument was suggested above, and it is an extension of several approaches. To the

best of the author's knowledge, it has not been definitively made elsewhere. Second, this thesis argues that Process Mining can mine and illuminate the goals and processes of users as expressed in a software application. The Process Mining literature is scant in this area. Thus, this research augments the tentative work done to date in mining user's intentions. Third, this thesis collects and comments upon the sparse and diverse suggestions concerning how to generate relevant event logs. Process Mining requires specific event logs in order to operate. Despite assertions that, "by definition, software runs on machines that can log user behavior" (van Genuchten et al., 2014, p. 94), it is not a given that software will log such behavior. Furthermore, many of the logs generated by software applications are focused on the application's behavior (containing, for example, error reports) rather than on the user's (see, van der Aalst et al., 2012). Creating a relevant event log is an important requirement for Process Mining (van der Aalst et al., 2012), and the general assumption in the literature is that such logs simply exist. Thus, this thesis defines criteria that are necessary in order to apply Process Mining to a software application. Finally, this thesis provides an illustrative case study of the application of Process Mining to a software application, demonstrating the feasibility of illuminating the implicit processes. A team instrumented a software package, applied Process Mining techniques to the event logs, produced visual graphics of a user community's interaction with a software package, and ultimately addressed a discovered software quality issue revealed through the Process Mining that was not raised by the user community itself. The presentation of mined activities validates the general argument that it is possible to extract user intentions via the implicit processes contained within a software application.

In seeking to make these contributions, this thesis is driven by three research questions.

RQ_1: Is there sufficient evidence in the Process Mining literature to support the idea that it can be applied to a non-Process Aware Information System (PAIS)?

RQ_2: What are the requirements of logging such that a non-PAIS application could be instrumented to apply Process Mining Techniques?

RQ_3: Is there evidence that applying Process Mining Techniques could support improved software quality?

Structure

This thesis is organized as follows. The remainder of this chapter provides an expansion of the argument for understanding users' goals in relation to a software application. Further, it suggests that goal comprehension is an important aspect of software quality. It also briefly examines whether the issue of goal comprehension can be resolved by using an Agile software development methodology. Chapter Two examines the existing literature of Process Mining. Therein, one will find a brief summary of the topic's historical development, as well as the benefits and assumptions that this history contains. Further, the chapter presents a summary of existing research that directly touches upon the application of Process Mining to the mapping of user intentions and its feedback to the development cycle. Chapter Three addresses the aforementioned shortcomings that assume the presence of properly constructed event logs. Process Mining requires the generation of event logs, yet there remains a dearth of recommendations on how these event logs are to be generated. This chapter presents this thesis' primary contribution to the literature. Chapter Four provides an example of an early application of Process Mining to an application, and what type of data may be obtained. Finally, Chapter Five offers some concluding remarks.

Terminology

Before proceeding with additional elaboration on goals, it may be helpful to provide definitions for a few terms and concepts. In this thesis, one may discern two different temporal perspectives when discussing software. The first perspective, exemplified in the Alice scenario or the word processor example, concerns *deployed* software. Deployed software is software that has been released to some set of end-users. A set of end-users is distinct from the developers and development-associated stakeholders in time and location. These end-users would use the software to achieve some set of goals. The second perspective, exemplified by the discussions of Cooper, Patton, and others, seeks to understand the goals and their achievement while the software is *in development*. This second perspective emphasizes representative users rather than actual users, since the software may not have been finalized (or the features to allow a goal to be realized have not yet been fully implemented).

Consequently, a methodology to assist with understanding user behavior would ideally be applicable during both software construction and after software deployment. In the construction phase, such a methodology would allow for capturing software utilization as developers and testers construct and validate the software. During the deployment phase, such a methodology would allow capturing utilization from a wider variety of end-users. This larger set of users would likely exercise the software in different ways (Raymond, 2008, pp. 32-36). This latter set is also likely to have a greater number of diverse goals. The various pathways exercised in the software would be different, assuming any degree of flexibility in the software. Nonetheless, the data collection and analysis methodology would be similar between the two phases. The deployed software, however, would presumably generate a greater amount of collected data. Managing the potentially large data collection is briefly considered in Chapter 3.

The primary use of a data gathering and analysis methodology designed to ascertain user intentions is to illuminate the means by which the users achieve their goals. As such, the methodology would address both of these temporal perspectives. When the methodology is applied during development, it may be seen as part of a comprehensive software validation exercise. Software validation exists to ensure the software conforms to stakeholder expectations (Christopher, 2003; Paula, 2009; Tomas, 2003; Wallace & Fujii, 1989). Capturing data as developers and testers exercise the software to ascertain the flow, performance, and goal realization provides another data point in the validation process. In contrast, when the methodology is applied to data captured from a variety of end-users, the methodology may be seen as an aspect of software analytics. Software analytics strives to "enable software practitioners to perform data exploration and analysis in order to obtain *insightful* and *actionable* information from data-driven tasks around software" (Zhang et al., 2011, p. 55).

Thus, while the timing and the incorporation of collected data is different, it is useful to have an approach that would cover both of these legitimate temporal perspectives. In van Genuchten et al. (2014), the authors examined information collected during a development phase in order to improve the software prior to delivery. In contrast, Khodabandelou et al. (2014) examined information collected from deployed software in order to ascertain users' goals. In both instances, the desired outcome was actionable insight into how users interacted with a particular software application. A methodology to support the gathering and analysis of data to support user behavioral analysis would provide benefits across the software life cycle.

A second point of potential ambiguity is in the use of the term "feature." Elsewhere, it was noted that a goal might be understood as an expected end condition with criteria for its satisfaction achieved via a process. In this formulation, a software feature was relegated to an

intermediary task or activity. It is important to emphasize that while a feature is a step in a process, the step itself must have value to the user. Returning to the Alice scenario, a "feature" might be the ability to apply load conditions to an initial model. In a word processor, a "feature" might be the ability to save a document. Therefore, a feature is, "a unit of functionality of a software system" (Apel & Kästner, 2009, p. 49). Features are not of uniform size, however, and are defined solely within the context of the software program. A feature thus has value to the end-user, and, as noted, it is frequently the definitional unit for stakeholder requests as well as developer implementation and testing.

A third point of potential vagueness is found in defining the target audience for a methodology addressing the concerns raised above. While various audiences could potentially benefit from a paradigmatic change from a focus on features to one on goals, and different software development methodologies could potentially benefit from a means by which to analyze the software-in-use, the primary audience for the application of a Process Mining methodology to software is an Agile development team. One finds that the literature (e.g., Rubin, Lomazova, et al. (2014); Rubin, Mitsyuk, et al. (2014); van Genuchten et al. (2014)) has tended to assume an iterative SDLC where standard routes allow feedback to be incorporated quickly and easily. In addition, Process Mining is a methodology designed to be executed frequently, and the data incorporated into a management activity. Agile SDLC's have distinct opportunities for incorporating the discovered data, whether in daily meetings or in routinely scheduled review activities. Plan-based approaches may not have the same level of flexibility to incorporate findings where gathered data deviate from expectations. Plan-based approaches, following Boehm and Turner (2004) "are characterized by a systematic engineering approach to software that carefully adheres to specific processes in moving software through a series of

representations" (Chapter 1, "Characteristics", para. 2). Such approaches assert that software quality improvement results from process improvement (Boehm & Turner, 2004, Chapter 1, "Characteristics"), rather than from understanding the software itself. One could apply Process Mining to the software process (and several authors have done so), and potentially utilize Process Mining to improve the SDLC process. However, the argument in this thesis is that Process Mining may be applied to a software application itself. The underlying philosophical differences between an Agile SDLC and a plan-based SLDC contribute to a bias towards Agile, or at least highly iterative SDLCs. In addition, when examining the application of Process Mining to process improvement many authors utilized language reminiscent of iterative conversations (e.g., van der Aalst, 2005).

Another construct is found in the term "stakeholder." A "stakeholder" may generally be understood as an individual who has a "right, share, claim, or interest in a system" (ISO/IEC/IEEE, 2010, p. 343). This definition may be augmented by conceptualizations of risk, or perceptions of being affected by a decision or activity (ISO/IEC/IEEE, 2010, p. 343). In this approach, the term is exceedingly broad, incorporating individuals who may directly participate in the definition of a software application, as well as those potentially affected by decisions removed in time and space. Ambler (2012a) provided a more expansive definition of a stakeholder as being:

> anyone who is a direct user, indirect user, manager of users, senior manager, operations staff member, the "gold owner" who funds the project, support (help desk) staff member, auditors, your program/portfolio manager, developers working on other systems that integrate or interact with the one under development, or maintenance professionals potentially affected by the development and/or deployment of a software project.

In addition to the direct incorporation of business-oriented individuals, Ambler also included developers into the definition of a stakeholder. While these definitions highlight the variety of potential stakeholders, their broad scope makes distinguishing a stakeholder difficult. Moreover, to support the Agile principle of, "Our highest priority is to satisfy the customer" (Fowler & Highsmith, 2001), it is potentially useful to reduce the definitional scope of a stakeholder.

As such, this thesis adopts a more restricted definition of a stakeholder. For the purposes of this research, a stakeholder is an individual who provides "product guidance and feedback" (Amiryar, 2012) during the development of a software application. A stakeholder is therefore different than the individuals responsible for designing, writing, or testing the application (consider, e.g., Crispin & Gregory, 2009). Stakeholders contribute the User Stories that are implemented. The general end-user community, however, is not a stakeholder since these individuals are not involved in the application development. Furthermore, this community tends to be more diverse and populous than the stakeholders involved in the development cycle (see, e.g., Boehm & Turner, 2004).

The Need to Better Understand the Users

Frequently, software is developed, and improvements are implemented and released, based upon expected behaviors and assumed comprehension of how users interact with a software program. Using assumed behavior as a basis for software development led Cooper et al. (2007) to lament that, "Most technology products get built without much understanding of the users" (Chapter 1, "Ignorance about users", para. 1). Stated slightly differently, software is developed based upon the assumed processes by which features are exercised in order to accomplish some goal. In contrast, Cooper argued for a "repeatable, predictable, and analytical

process for *transforming an understanding of users into products that both meet their needs and excite their imaginations* [emphasis in original]" (Chapter 1, "The lack of a process", para. 1). Such an approach is predicated upon an understanding of how users "will actually use the product" that is being build (Cooper et al., 2007, Chapter 1, "The lack of a process", para. 1). Adzic and Evans (2014) went further, arguing that software development should not merely describe the users' (expected) behavior, but rather incorporate a means to measure the change in the users' behavior between two points in time. However, the authors did not suggest a means for measuring or quantifying the differences. Khodabandelou et al. (2014) argued that it is important to understand users' intentions, where intentions are essentially equivalent to goals, in order to support software improvement. Buse and Zimmermann (2012) proposed understanding the actual utilization of software as a set of data to satisfy the analysis needs of software engineers. Thus, there are a number of authors calling for better insight into how software is actually utilized in order to support a plethora of diverse approaches for improving software quality and documenting its value.

In a similar vein, Patton (2014) argued that software is about changing the world by enabling the users to achieve their "goals as a consequence" by using a piece of software (Chapter 1, "Software Isn't the Point", para. 2). Moreover, Patton believed that it is desirable and possible to "measure what people do differently," as well as whether the users' lives are improved (Chapter 1, "Software Isn't the Point", para. 2). Adzic and Evans (2014) noted that valuable software produces "an observable change in someone's way of working." Though Adzic and Evans (2014) expressed observable behavior in terms of subjective value, their formulations are compatible with the concepts of goals as articulated by Patton (2014) or Cooper et al. (2007).

Actual, in-use software processes stand in contrast to static, feature-based software conceptualizations. Yet, the predominant paradigm for discussing software is one of feature articulation. There are multiple reasons why feature articulation is the predominant paradigm, even though it cannot provide insight into actual user behavior. First, users are frequently unable to clearly articulate their needs, and "when asked direct questions about the products they use, most tend to focus on low-level tasks or workarounds to product flaws" (Cooper et al., 2007, Chapter 1, "The creation of digital products today", para. 1). Second, software development itself is often structured around a set of features. Whether it is from the requirements to be implemented, or User Stories, most software development tasks implement and testing specific features. Buse and Zimmermann (2012), for example, documented how developers focused almost exclusively on information pertaining to discrete features, rather than on larger issues of how a user interacted with the product as a whole. Cooper et al. (2007) also noted the lack of global considerations in software development. Third, the legacy of software engineering as essentially a "Taylorian" process (Boehm & Turner, 2004; Mahoney, 2004; Maurer & Melnik, 2006) emphasizes discrete tasks defined by one set of specialists, but implemented by a different set. The discrete task approach removes the global outlook for a product and its uses, and instead focuses on the achievement of specific tasks.

In essence, the foregoing is an argument for understanding the actual behavior of users when they interact with a software application. Buse and Zimmermann (2012) asserted, "In any business endeavor, understanding customers is important; software development is no exception." In a similar vein, Cooper et al. (2007) noted that companies have access to a plethora of data about users, but little data on what makes them happy (Chapter 1, "Ignorance about users"). Patton (2014) also emphasized a need to understand what makes users happy, but with

the additional emphasis on goal attainment. Yet to delight the user (Denning, 2016), it is necessary to understand the user. Buse and Zimmermann (2012), Cooper et al. (2007), along with the others noted above, argued that understanding users — and ultimately what makes them happy — is derived from understanding "how a user is using [a] product" and whether the users are "performing tasks we [developers] expect" or "performing tasks we didn't anticipate" (Buse & Zimmermann, 2012, p. 992). If a user undertakes unanticipated tasks, it may be an example of inefficiencies in the way software supports goal attainment; it may reveal creative, new uses for the software. Buse and Zimmermann (2012) situated the need to measure actual software utilization within the framework of software analytics, but did not provide specific means by which to collect the data nor transform it into information. The approach also remained steadfastly "task" based. Cooper et al. (2007) proposed a process for data collection and analysis, but their method relied upon "techniques of ethnography, stakeholder interviews, market research, detailed user models, scenario-based design, and a core set of interaction principles and patterns" (Chapter 1, "A process overview", para. 1). This approach does not allow for understanding specific users in specific contexts, nor does it (necessarily) utilize the actual software to document user behavior (or the change in their behavior as suggested by Adzic and Evans (2014)).

Consequently, understanding how people actually use software in achieving their goals would provide feedback that could improve the software. Such an approach would sharpen the focus on the "intentional processes" (Khodabandelou et al., 2014) people use. Answering Alice's questions regarding the value of an upgrade could therefore be quantified, as van Genuchten et al. (2014) suggested. The value, however, would be demonstrable not only in terms of the feature set, but also in terms of the overall process improvements that may be obtained through the

upgrade. Such an understanding would assist in answering the direct question of understanding how users actually use the product (Cooper et al., 2007, Chapter 1, "Ignorance about users").

It therefore seems clear that obtaining better insight into how software is used in the realization of users' goals would provide many benefits. Applied correctly, a methodology to gather data would provide the repeatable and analytical process that Cooper et al. (2007) desired. With the correct analysis, differences between two models could be calculated, such as shown in van Genuchten et al. (2014). Extracting the process models that individuals utilize when working with the software would ensure that development assumptions are accurate, and that anticipated value is achieved. Such models can provide data-backed analysis of how a given set of changes in a software update will affect a given set of users.

Can Agile Resolve the Situation?

When confronted with potential issues regarding customer requirements, a frequent response in the current software development community is to "become Agile." Consideration of the Agile movement is important for two reasons. First, it is possible that problems with understanding users and their goals is rooted in the particular SDLC that a team uses, and is not an inherent gap in knowledge. If changing to an Agile SDLC would allow for appropriate insight into users and their goals as expressed through behavior, then investing in Process Mining techniques likely would not be appropriate. The second reason for considering the Agile SDLC is that when the Process Mining literature (e.g., Rubin, Lomazova, et al., 2014; Rubin, Mitsyuk, et al., 2014; van Genuchten et al., 2014) considers software applications, it situates Process Mining within an Agile SDLC. As these authors suggested, Process Mining can be a means with which to further engage and educate stakeholders about the actual utilization of a software

application. There does not appear to be any attempt in the existing literature to link Process Mining to a more plan-based (as defined by Boehm & Turner, 2004) SDLC.

A full consideration of whether applying Agile methodologies could completely obviate the need for Process Mining in an SDLC is beyond the scope of this thesis. Instead, the following considers whether there are sufficient objections in the literature to suggest that Process Mining could have an important role in revealing user behavior in a way that Agile methodologies cannot directly address. Ultimately, Agile development is about fostering conversations with stakeholders (Adzic & Evans, 2014; Hathaway & Hathaway, 2013). Process Mining could potentially contribute to the conversation by confirming usage assumptions, or raising questions about the means by which the software is employed. Such input would be another valuable data point in the refinement of the software's value proposition. This potential alone, however, is not necessarily sufficient cause to undertake the investment necessary to use the methodology.

Agile methodologies, following from the Agile Manifesto (see, Fowler & Highsmith, 2001), suggest two points that address understanding user behavior in a software application. First, one finds the interdependent principles of developers and stakeholders working together daily on a project, combined with the assertion that developers and stakeholders should have face-to-face conversations. Second, one finds an emphasis on satisfying stakeholders through the early and continuous delivery of valuable software. Early, in this context, is defined in opposition to the "big-bang" plan-based releases where all anticipated features are delivered at a single point in time. In addition, Agile promotes rigorous acceptance testing on the part of the stakeholders. Such testing should ensure that the software conforms to expectations, with one aspect of those expectations being the delivered value.

Unfortunately, it is not clear that Agile methodologies address the full range of concerns raised above. Cooper et al. (2007) argued that goal elicitation is not inherent in Agile approaches. Further, integrating stakeholders "directly into the development process on a frequent basis" has the "salutary effect of sharing the responsibility of design" with the stakeholder, but "ignores a serious methodological flaw: a confusion of domain knowledge with design knowledge" (Chapter 1, "The lack of a process", para. 4). Despite the fact that Agile User Stories should define value for the stakeholders, the value is frequently measured in terms of the single story, and not in reference to the program's totality. Adzic and Evans (2014) also noted that frequently value is assumed in a given User Story because it is something for which a stakeholder asked (Chapter 1, "Describe a behavior change", para. 2). Without external verification, value can often remain subjective, or limited to a small subset of the stakeholders. Further, as Cooper et al. (2007) noted, "Unfortunately, reducing an interactive product to a list of hundreds of features doesn't lend itself to the kind of graceful orchestration that is required to make complex technology useful" (Chapter 1, "The creation of digital products today", para 1).

Similar to the arguments of Cooper et al. (2007), Crispin and Gregory (2009) argued that developers lack sufficient domain knowledge to facilitate a comprehensive approach. They suggested developers lack a holistic view of the domain and of how the end-users operate. Crispin and Gregory (2009) urged that professional testers fill the gap between developers and stakeholders. However, this method lacks the systematic approach of Cooper et al. (2007) since it relies upon exceptional individuals (the Agile testers) rather than a repeatable process for obtaining the larger view of how the software is utilized.

Related data from Boehm and Turner (2004) suggested that a given set of stakeholders who advocate for a particular user story are not necessarily representative of the organization as

a whole. This lack of representation was also noted in Duan et al. (2008). In addition, the competence of the stakeholders who are closely aligned with the development team can vary across projects, or even over time within a project. In one case study, Boehm and Turner (2004) noted the early success of an XP Agile project when the embedded stakeholder was highly committed, motivated, and well informed. However, following a change in the stakeholder, the project suffered. The data in Boehm and Turner (2004) was not collected, however, with a focus on how well the software met the user community's needs. Given Boehm's historical writings (e.g., Boehm, 1978; Boehm, 1986, 1989), a lack of understanding the users' goals would be seen as a risk to be managed.

Agile approaches typically document the functionality of a software application in User Stories. A User Story is written from the perspective of a specific role that an individual has within an organization, and expresses a desired outcome of a single interaction with an application (Hathaway & Hathaway, 2013, Chapter 1, para. 7). Frequently, User Stories adopt a template of "As a [role], I want [feature] because [reason]" (Ambler, 2012b; Cohn, 2004; Pool, 2008). The emphasis in the template is on a *feature*, and not a goal. Nothing prohibits using a goal statement, rather than a feature, of course. An alternative template did substitute the word "feature" with "goal" (mountaingoatsoftware.com, n.d.). The textual description, however, reverted to the term "feature." This latter example demonstrates the difficulty of thinking in terms of goals as opposed to features when discussing software development. User Stories are designed to encourage discussion with the stakeholders; to the extent the discussion is inherently focused on features, engaging in a conversation does not transcend the potential paradigmatic restraints associated with feature-based thinking.

Even if the User Story were generated from a goal perspective, two other issues would remain. The first issue is the level of analysis associated with a User Story. Cooper et al. (2007) suggested that an application must be seen in total, and not merely as a series of independent parts. Sufficiently complex and flexible software will support multiple goals, and the means of obtaining these goals will likely vary by individual user. Optimization on only a subset of functionality can result in achieving efficient outcomes in measured cases, but inefficiency in the overall application (see, e.g., Floudas et al., 2013). The second issue is the assumption that all goals and functionality are captured in independent User Stories. There are many instances where software enables users to achieve novel, or unexpected, outcomes (see, Buse & Zimmermann, 2012, p. 992). Consequently, software that is implemented as satisfying a series of User Stories is not necessarily equivalent to "delighting" the end-users across the entirety of the application.

Thus, while Agile approaches may reduce the gap between requirements as documented, interpreted, and implemented by developers, and the actual expectations of stakeholders, it does not follow that the stakeholders' expectations are necessarily goal oriented or holistic. Further, the reliance upon specific stakeholders in the development cycle assumes the representativeness of these stakeholders to the overall user community. Finally, even to the extent that stakeholders are representative, a significant number of end-users may have different approaches to using the software, or divergent goals that are met with the software, but not through the implicit processes assumed by developers or the embedded stakeholders.

Why Understanding User Behavior Matters for Software Engineering

Applying methods to reveal user behavior is useful only if the data are ultimately applied to improve the means by which users achieve their goals. Such improvement addresses software

quality. Though quality is “an elusive term”, and perhaps “especially elusive in the software field” (Glass, 2003, p. 127), it is nonetheless the overarching measure most important to users. The field of software engineering was essentially created in the late 1960s to apply systematic efforts to the productive development of quality software in order to address a perceived “software crisis” (Pfleeger, 1998). In order to address software quality, the predominant paradigm of industrial manufacturing was referenced and the terminology and methodology for quality through process conformance and improvement was adopted. This paradigm essentially assumed that quality was the result of following a process, and maturity models (such as the Capability Maturing Model) and ISO 9000 evolved to capture process conformance (Pfleeger, 1998, p. 11).

The difficulty in discussing software quality is that it is not a single attribute (Boehm, 1978; Pfleeger, 1998, p. 472). Instead, quality is a constellation of attributes such as functionality, reliability, usability, efficiency, maintainability, and portability (ISO9126 in Pfleeger, 1998). This collection of attributes currently contains 20 measurable factors (Denning, 2016, p. 23). Despite acknowledging that subjectivity is present in any determination of quality, it seemed an article of faith that a precise definition of software quality resulting in quantitative measures was the ultimate goal (see, for example, Cavano & McCall, 1978). Such measures are the natural outcome of the statistical basis of science and traditional engineering (Shewhart, 1931). The end-user perspective is largely absent from most approaches, or it has been retroactively added (e.g., CMMI), and this lack of an end-user perspective stems from the underlying paradigm of process conformance and objective measurement.

Quality is not, however, necessarily a purely objective standard. John Locke (1959) in his work *An Essay Concerning Human Understanding* created a distinction between primary and

secondary qualities. Primary qualities are those that are independent of the observer. Software quality in such a formulation, therefore, would be intrinsic to the program. These primarily qualities would find realization in the standard “software product quality” characteristics as proposed by, e.g., (Boehm, 1978) or ISO 25010. Secondary qualities are those that have an effect on certain people. They are a result of “what we think, feel, or sense as a result of the objective reality” (Shewhart, 1931). Software quality, in this case, would be perceptual and subjective; they are extrinsic to the application. Locke also argued that knowledge of these secondary qualities does not provide objective facts about the objects themselves. Interestingly, the implication for software quality is that a user's perception of quality does not provide any insight into the objective quality of the program as might be measured by developers.

The results of a survey (Chartered Quality Institute, 2013) are illustrative of how diverse “quality” is as a concept. The Chartered Quality Institute, a "professional body dedicated entirely to quality," asked its members to define quality. Of the 20 proposed definitions of quality, 10 are extrinsically measured (e.g., “better than you would, or could have, expected,” or “what the customer perceives it to be”). Two of the definitions cannot be easily categorized, and eight are intrinsic. The intrinsic measures are primarily the result of following a process (e.g., “reducing the various around the target”, and “doing the right thing every time”). In contrast, the extrinsic concepts are focused on perceptual outcomes. The CQI survey results and the mapping to an intrinsic or extrinsic nature are shown in Table 1.

In van Solingen, Kusters, Trienekens, and van Uijtregt (1999), the authors argued that “when striving towards software product quality, two main approaches can be distinguished” (p. 475). These approaches are the process approach and the product approach. To a certain extent, the intrinsic (or primary) quality measures of a software program may be seen as the result of

following a process. To improve software quality, therefore, one must improve the software development process. Conversely, the extrinsic (or secondary) quality measures result from interacting with the end-users, and quality may not necessarily be achieved solely through a process. That is, these qualities are a found in the product and its utilization, not the development process.

Understanding the end-user's goals, as demonstrated through their actual use of a program, is ultimately a contribution towards software quality. Ferre et al. (2004), in citing ISO 9241-14, directly linked software quality with goal achievement. Usability (one aspect of software quality) is "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use." Satisfaction, in this approach, is essentially an extrinsic quality that results from the users' interaction with the software. The concepts of effectiveness and efficiency could be either extrinsic or intrinsic depending upon the precise definitions and measurement. However, these latter two concepts do not exist as qualities independent of goal achievement. Rather, these quality metrics are bound to goal achievement as expressed through the actual use of the software. Revealing this quality requires a methodology that will illuminate the goals of the users, and do so in a context that is globally aware of the software program, rather than point-focused on feature achievement. The actions of users themselves reveal the path towards quality.

In considering the revealed processes, the Process Mining literature tends to leave the linkage to software quality as an implicit assumption. Though van Genuchten et al. (2014) discussed a discovered error in the code tied to their case study, and error removal certainly increases quality, the authors did not otherwise directly address how applying Process Mining would contribute to software quality. The closest statement in the article was that "end-user

functionalities… can be compared before and after release" (p. 99). Presumably, this comparison can illuminate differences between development assumptions and actual utilization, though it could also be applied between an existing release and improvements made within the development phase of an SDLC. The authors, however, did not provide an explicit means for incorporating the information. Rubin, Lomazova, et al. (2014) specifically linked Process Mining to the Agile SDLC. As noted elsewhere, fostering conversations with stakeholders based upon the collected data is an important aspect of quality in an Agile methodology. These authors argued that outputs from Process Mining can contribute to improved communication with stakeholders, and can support continuous improvement via the reflection on how to be more effective. Nonetheless, these examples failed to link quality directly to the end-users' experiences. Quality is found in the effectiveness of goal attainment on the part of various users. Rubin, Lomazova, et al. (2014) did not appear to consider "legitimate peripheral participants" (Huang & Liu, 2005), nor did the account for the potential stakeholders to differ from the more general end-user community. The ultimate application of Process Mining is not just to collect data, but also to utilize this data to improve the quality of the software application. Consequently, further elaboration and research on the application of the data to software quality improvement remain necessary.

Table 1. CQI Quality Survey and Quality Nature

CQI Response	Intrinsic/Extrinsic Nature
A measure of excellence	Extrinsic
The characteristic of a product or service that bear on its ability to satisfy stated or implied needs	Both, Note this definition is from ISO 8402
Better than you would, or could, have ever expected	Extrinsic
What the customer perceives it to be	Extrinsic

Doing the right thing, every time	Indeterminate; may refer to following a process, which would result in intrinsic
The degree to which an item or process meets or exceeds the customer's requirements and expectations	Both; similar to the ISO definition in that requirements should be formalized and thus objectively measurable, but expectations are likely to be subjective
How closely a product or service meets its design specification	Intrinsic; the product is objectively measured
Surpassing customer needs and expectations throughout the life of the product	Extrinsic
A product or service free of deficiencies	Indeterminate; deficiencies is not defined, and could be free of defects (objectively measured) or it could mean not being deficient in the eye of the customer
Reducing the variation around the target	Intrinsic; highly measurable, and tied to the process. Six Sigma processes exhibit this characteristic
A state of mind	Extrinsic. It does relate to the point that quality is not something that can be done at the end of a project.
The extent to which products, services, processes, and relationships are free from defects, constraints, and items that do not add value for customers	Probably intrinsic. The value for a customer is likely to be objectively measurable.
Never having to say you're sorry	Extrinsic; or from a movie
An ever-evolving perception by the customer of the value provided by a product. It is not a static perception that never changes but rather a fluid process that changes as a product matures (innovation) and other alternatives (competition) are made available as a basis of comparison	Extrinsic
Peace of Mind	Extrinsic
Never an Accident	Indeterminate; implies that it is a result of a plan
The inherent features possessed by a product or service	Intrinsic
When what comes back is the client, not the product	Indeterminate
No surprises	Indeterminate

Not achieved by doing different things; it is achieved by doing things differently	Most likely a reference to following process, and thus intrinsic

A Repeatable Approach for Software Use Understanding

What is sought, therefore, is a methodology that can illuminate the processes that are utilized in a software package in order to achieve a particular goal. Furthermore, the methodology should provide insight into the goals themselves — that is, the end-state that users are attempting to achieve. The methodology should not be dependent upon specific stakeholders, but rather should allow for its application across a variety of users. It should be applicable across multiple stages in the software life cycle. This methodology should allow not only for the discovery of the implicit software processes, but for also the comparison of processes at different times. Finally, the methodology should be repeatable, predictable, and analytical. That is, rather than relying upon ad-hoc approaches for data collection and analysis, it should have a prescribed means for such activities. Ideally, the methodology should be available to any relevant member of development team. Further, the methodology should not require specific expertise in its application or interpretation. This latter point follows from an implicit assumption in many Agile approaches of the relative equality of all team-members, self-organization for specific work notwithstanding.

Advances in a technique known as Process Mining can potentially meet the above criteria. Process Mining can illuminate the pathways utilized in a software program. That is, it can *discover* the process model that is in actual use. Process Mining can allow for *conformance* checking between process models, whether the models are the presumed ways in which the software is used, or are models from different points in time, or models contrasted between users or a user and a reference group. Process Mining can assist with *enhancing* the implicit processes by highlighting areas where there are inefficiencies in the goal attainment. Process Mining can

help with understanding the intentions/goals of the users by documenting the end states that a user achieves. In sum, Process Mining is a repeatable methodology, allowing for predictability in the approach and the data presentation.

Process Mining is not a panacea, however. It may require expertise to interpret results. It requires specific data to be collected in order to generate the event logs that it mines. While software packages can creation such logs, a great deal of software today does not generate logs; and when they are created, logs are either generated only in exceptional cases, or in formats that are difficult or impossible to adapt to the Process Mining input specifications. The amount of data collected may be great, and the collected data must be carefully developed to provide value to the development team while respecting, as appropriate, various concerns regarding privacy.

Process Mining has been applied, as will be seen later, to a variety of domains. It is a relatively new discipline, but it has shown success in many domains. To date, however, it has not been rigorously applied to illuminating software utilization. Thus, the proposed application of Process Mining to enhance software development is a relatively novel utilization of the technique. The application fits well within the concepts of Process Mining, but it is a largely unexplored application. The ability to guide software development based upon in-use processes is a new frontier in software engineering.

Summary

Consider the creation of a pivot table in a spreadsheet program. Typically, a pivot table is considered a “feature” by which a user may summarize data. To add a pivot table to a spreadsheet, the user executes a series of steps in order to define the input data. A traditional measure of software quality might examine – based upon known test inputs – whether the spreadsheet contained the pivot table, and whether it had the correct results. Such measures

might be extended into the actual code itself, examining questions of code complexity or readability. Data regarding "usability" may have been collected and applied to, for example, implementing or improving wizards to assist with the creation of the pivot table.

It is not a stretch, however, to understand that frequently users must undertake creating the pivot table multiple times due to struggles with it. As one example, perhaps the data are presented as counts rather than sums (Debra, 2014). Other issues may require deleting and re-creating the table itself. In such cases, while the *feature* met certain aspects of *objective* quality, the user failed to meet her goal, and thus *subjectivity* the software would fail to "delight" (Denning, 2016, p. 24) the user. Furthermore, this example illustrates how even non-process aware software implicitly contains processes by which users operate in order to achieve desired outcomes. If one could examine the process of the user in this example, one might see loops in the process of table creation, deletion, creation, edit, etc. These loops might be indicative of quality issues. Yet the standard metrics of quality would not capture the actual, in-use behavior of the user, and thus such metrics could not be utilized to improve user-centric quality. While Process Mining applied to event logs created by the application could illuminate the issue, one is still confronted with the need to capture, analyze, and incorporate the data into the SDLC.

Consequently, this simple example sets forth the essence of this thesis. Whereas traditional frameworks of software quality have tended to focus on objective measures of features, conceptually newer models of quality explicitly focus on the subjective experience of the user. Furthermore, such subjective experiences are expressed through the processes by which a user interacts with a software application in pursuit of specific goals. These processes may be examined by applying Process Mining techniques to event logs if the application is specifically instrumented to emit logs suitable for analysis.

Chapter 2: Literature Review

This section reviews relevant literature on Process Mining. Process Mining is the application of computer algorithms to discover the as-is, runtime process as recorded in event logs. In examining the academic literature on Process Mining as applied to software applications, one may find an interweaving of three threads. The first thread is the initial work of "process discovery" initiation by Cook and Wolf. This thread also started with examining software applications directly. The second thread is the knowledge discovery in databases (KDD) approach that applied requirements to extracting, merging, and consolidating data contained in "databases" (of which event logs are one instance). The third thread is the evolution of Workflow Mining to Process Mining with its associated increase in definitional rigor, algorithms for process discovery, and the creation of Process Mining software to support process discovery via a variety of algorithms. This third thread can also be roughly divided into two phases. The early phase consisted of initial definitions and explorations of various approaches for re-creating a process from a set of inputs from workflow applications. The early phase generally addressed the question of how to re-discover a model from an event log. In order for Process Mining to be a viable approach, it needed to address a number of theoretical and practical issues to demonstrate process discovery from event-based logs. The later ("maturation") phase emphasized a consolidation of definitional terms, a shift from a frequent emphasis on workflow systems to broader process questions, the emergence of software packages to support Process Mining, and increasing industry. The IEEE has also backed a "Process Mining Manifesto," and the publication of this Manifesto demonstrated the integration of these three threads. Though both practical and theoretical challenges remain — and as such it is premature to label process mining as fully mature — continued research and the broader application of Process Mining to a variety of practical domains indicates a maturing paradigm.

To develop this review, a search was executed against EBSCO, the Association for Computing Machinery (ACM), and the IEEE databases. These searches were supplemented by queries against Google Scholar. These searches looked for articles pertaining to “Process Mining”, “Workflow Mining”, or “Process Discovery.” After filtering duplicate articles and articles that were not directly relevant (e.g., articles about geological mining extraction processes), the resultant collection was approximately 480 entries. These articles were then further filtered by eliminating studies focused on areas not directly relevant to Process Mining and software or algorithmic development (e.g., the plethora of studies examining Process Mining and health care or direct business processes). From the approximately 110 remaining articles, the references were examined to trace additional historical studies. Ultimately, 167 articles provided the general set of articles. From this collection, the representative articles tracing the development of Process Mining with special attention to software were used. There were 35 primary articles directly referencing software and process mining, though the 30 of these studies examined software development processes rather than software applications.

After examining the general Process Mining literature, this section reviews contributions relating directly to software development. Despite an early impetus around improving the software process, Process Mining became more strongly associated with business workflow systems. A few recent case studies, however, have returned to applying Process Mining to understanding software processes. Unfortunately, the literature around analyzing software applications, rather than the process by which they are developed, is scant, and therefore represents an interesting point for research.

The First Thread – Process Discovery

Cook and Wolf (1998a) conducted one of the first studies that encouraged the discovery of processes from event data. In this paper, the authors developed a data analysis technique they termed "process discovery." By applying grammar inference techniques to event logs, they intended to discover recurring patterns of behavior exhibited by software systems. In order to discover the processes, they described three methods for discovery: neural networks, a pure algorithmic approach, and a Markovian approach. The latter two were seen as more promising for future research. This article conceived of an "event" as a necessary atomic element that characterized "the dynamic behavior of a process in terms of identifiable, instantaneous actions" (Cook & Wolf, 1998a). An on-going, evolving software system would inevitably diverge from its original formal model, according to the article. Therefore, the choices faced in software development are twofold: expend increasing effort in maintaining the formal model, or allow algorithms to discover the underlying model as expressed in the actual execution of the software system. This work is the extension of earlier studies examining software validation that depended upon an explicit formal model and extended metrics (e.g., Cook & Wolf, 1994). Rather than assuming a static model was the basis for validation and software improvement, Cook and Wolf (1998a) suggested that a formal model for software validation may be continually updated through examining the actual behavior of the software. This article also noted the desirability of various attributes of an event, including time and actors. At the same time, Cook and Wolf introduced the concept of an "event stream", which is the continual sequence of events recorded by an application. The discovered model was presented as a finite state machine (FSM), which provided an initial graphical representation, but suffered limitations when modeling concurrent activities.

Though not applied to a software application itself, Agrawal, Gunopulos, and Leymann (1998), independently but essentially concurrently with Cook and Wolf (1998a), proposed creating a general directed graph from the log files. The impetus for the research was to provide data to support the quality and accuracy of business processes as executed within an organization (essentially "conformance" verification). Thus, this article fits the early approaches of advocating for process discovery in order to perform the conformance analysis.

They examined workflow systems, and the "small, unitary actions, called activities" performed during the execution of a workflow (p. 469). Agrawal et al. (1998) acknowledged that an activity modified the state of the process, whereas Cook and Wolf (1998a) did not emphasize this trait. As with the Cook and Wolf (1998a) results, the graphs were sequential. However, Agrawal et al. (1998) did not examine alternative discovery approaches, but used a single and specific algorithmic approach. They included suggestions for a specific input structure, consisting of the process, an associated activity, whether the event was the start or end of the activity, and a timestamp.

Second Thread – Knowledge Discovery in Databases

While the first thread gave the initial impetus to discovery, it did not consider issues of large number of event logs, nor logs that might be distributed across the organization. Furthermore, prior to the suggestions of the first thread, it was usually a manual task to develop understanding of a process, whether the process was within an organization or within a software application.

Fayyad, Piatetsky-Shapiro, and Smyth (1996) suggested that in order to deal with knowledge at a large scale, it would be necessary to rely on more automated approaches rather than on experts who turn data into knowledge via manual analysis. The field of extracting

knowledge from data may be understood as “Knowledge Discovery in Databases” (KDD) according to these authors. Fayyad et al. noted that the “basic problem addressed by the KDD process is one of mapping low-level data into other forms that might be more compact, more abstract, or more useful” (p. 39). The authors emphasized the need for a process consisting of several interactive and iterative steps to make knowledge discovery useful. These steps, including comprehension of the domain, data set creation, data cleaning and preprocessing, and so on, were frequently assumed in the general Workflow/Process Mining literature. However, the early work on Process Mining did not explicitly acknowledge the surrounding tasks needed to mine data. By focusing on workflow systems that emitted logs, early Process Mining literature ignored steps necessary if one is not addressing ready-made, consistent event logs.

Utilizing the same KDD terminology as Fayyad, Hill and Jones (1999) defined KDD as “discovering useful knowledge in data and also as ‘the nontrivial extraction of implicit, previously unknown and potentially useful information from data’” (p. 49). These authors noted that traditional data mining techniques resulted in a disconnect between “patterns of interest and the context of the original data” (Hill & Jones, 1999, p. 50). This distinction separated KDD from Data Mining in general. In addition, it provided additional definitional support to Process Mining as a focus on the specific context in which any discovered pattern would operate. Their research emphasized a “case based reasoning” technology that adopted nodes and semantic links between the nodes revealed through processing data. Despite their case-based approach, the authors did not attempt to discover particular processes; instead, the need to retain the linkages between extracted data points contributed to the paradigm of discovering knowledge rather than data correlations. As they noted, data mining is primarily concerned with searching for interesting patterns. As a result, data mining does not necessarily maintain semantic information,

nor does it likely reveal accumulated knowledge. While Process Mining terminology was not directly incorporated, the case-based approach and the need to retain linkages in order to understand patterns rather than correlations also appeared in the Process Mining literature. Consequently, Process Mining might be understood as a subset of KDD, though neither literature regularly made such an association. Both Process Mining and KDD argued that data mining alone is insufficient, as data mining must be located within a larger approach of knowledge discovery (see also, van der Aalst, 2012a). Each of these fields attempts to wrestle with positioning data mining as an aspect of their approach.

Schimm (2004) also situated his work in the context of KDD, akin to Hill and Jones (1999). However, Schimm was examining the development of concurrent workflow modeling, providing a bridge between KDD and Workflow Mining. This work utilized a block-oriented approach as opposed to the directed graphs or Petri Net approaches of the other authors. Schimm introduced "workflow algebra" to translate the input in a multi-stage procedure to the block diagram. The author suggested that block-structured models offered advantages over other approaches; a block model is always well-formed, safe and sound, and therefore the extracted models would not exhibit anomalies (p. 267). One drawback is that the input data requirements were greater than previous approaches (an issue also seen in Pinter and Golani (2004)). Moreover, processing event-based data required activities to have an associated life cycle. This life cycle must be explicitly emitted into the event log. This approach was similar to Pinter and Golani (2004) in its use of starting and ending points, but Schimm did not use interval information to derive a life span. The output model was presented in a block notation which was an approach the authors suggested would provide a more robust representation of parallel activities. However, van der Aalst et al. (2003) discussed the differences between graph-based

and block-based techniques, especially the latter's focus on rewriting requirements. In general, van der Aalst's summary indicated that a graph structure (such as a Petri Net) was the most prevalent approach in the literature, and that block approaches appeared to have limitations that had already been addressed by graph-based approaches, such as issues with hidden and duplicate task detection.

The importance of KDD in the early part of the intellectual evolution of the literature, and in the context of applying Process Mining to ad-hoc instrumented software in particular, was its insistence on situation knowledge extracted from a database within a particular context. In addition, in provided guidance and insights into managing and refining large datasets to support knowledge extraction. The early investigations into "Workflow Mining" relied upon the output of logs from Process Aware Information Systems – specifically workflow management applications. As these applications were generally designed to "manage the flow of work through the organization" (van der Aalst, 1998), they were deployed at an enterprise level with central logging and a constrained set of operations. Therefore, the early Workflow Mining literature was able to ignore many of the issues that KDD raised. Later work in the re-branded "Process Mining" literature (e.g., Bose, Mans, & van der Aalst, 2013; van der Aalst, 2011a) did examine the need for data cleaning and merging, though never to the extent as emphasized by the KDD literature.

Third Thread – Workflow Mining

The intellectual impetus for Workflow Mining was a paper by van der Aalst (1998). The author examined the potential application of Petri Nets to the modeling of workflow systems. Though the paper did not address questions of re-discovering processes from event logs, it advanced the argument that Petri Nets were sufficient for modeling workflow processes. In

contrast to a directed graph (Agrawal et al., 1998) or a Finite State Machine (Cook & Wolf, 1998a), a Petri Net has definitional advantages. Petri Nets represent a "system graphically by drawing a node for each state and an arrow to mark the transitions" (Pfleeger, 1998, pp. 157-158). In addition, Petri Nets have formal semantics, are expressive in that they support all primitives necessary to model a workflow process, and are amenable to formal analysis. These advantages would become important later in the evolution of the literature as questions regarding parallelism arose. Moreover, an initial justification for a focus on workflow systems was prevalent in the paper. Van der Aalst argued that workflow systems were critical for moving organizations beyond a focus on executing isolated tasks. Workflows, under the concept of a "case," allow for the execution of tasks — in serial or parallel — by different resources in a way that is efficient for the organization. Subsequent literature frequently ignored this van der Aalst paper, despite having formed an intellectual basis for future work by arguing for the presentation of workflow processes as Petri Nets. In addition, the paper defined the concept of a "Workflow Net." A Workflow Net (WFN) is a Petri Net with a single source and sink that represents the starting and ending point of the cases contained in the logs. That is, all the cases in workflow system have an identifiable starting and ending point. Later, van der Aalst, Weijters, and Maruster (2002) made the argument that a workflow system was not strictly necessary, only that event logs would need to contain data compatible with a WFN. The original van der Aalst (1998) paper, nevertheless, set the tone for several years of research emphasizing workflow systems. In addition, this article laid the foundation for the use of a Petri Net as the basis for modeling re-discovered processes.

The following year, Bae, Jeong, Seo, Kim, and Kang (1999) examined how workflow management systems can be an integration approach for traditionally stand-alone systems. This

study focused on how workflows may support business process re-engineering (BPR). A significant component of BPR is a thorough understanding of the existing "as-is" process. In order to support process verification, especially of the "to-be" process, the authors suggested translating an existing workflow definition to a simulation package in order to analyze process models. The authors noted, "Workflow and simulation [software systems] are both discrete-event systems" that share the concepts of activities and events (p. 205). Bae et al. further examined the process of translating formal "as-is" process models into a simulation suitable for execution and experimentation. In this sense, they were pre-cursors to the Process Mining concepts of process enhancement through simulation. In contrast to Cook and Wolf (1998a), the authors did not advocate examining existing log files to discover the process. Their ideas translating a model to support investigation and improvement, as well as emphasizing process behavior, was both reminiscent of Cook and Wolf (1998a) and van der Aalst (1998), but also an important piece of future development in the literature.

Maruster, van der Aalst, Weijters, van den Bosch, and Daelemans (2001) provided an early case study of process re-discovery by applying techniques to extract a process from hospital logs. The article provided an algorithm that could be applied to a workflow log to discover the process. The discovered process could then be presented as a Workflow Net. The algorithm was based on a detection of linkages between activities from a source to a sink, and then merging nodes that had the same past and future nodes. Weijters and van der Aalst (2001), also the co-authors on the Maruster et al. paper, proposed a different algorithm based upon a dependency-frequency table. Eventually, these algorithms would evolve into the "alpha algorithm" described in van der Aalst, Weijters, and Maruster (2004). The alpha algorithm was the basis for many of the Petri Net models used throughout subsequent literature.

In van der Aalst et al. (2003), one finds the early integration of discovery, workflows, and semantic links. In contrast to Cook and Wolf (1998a) and their use of an FSM, this van der Aalst et al. paper examined the application of Petri Net theory to the discovered processes. An FSM supports the basic structure of sequential processes, such as sequence selection and iteration (see, Cook, Du, Liu, & Wolf, 2004). A Petri Net formulation, however, supports modeling concurrency. Van der Aalst et al. (2003) documented three assumptions the logged data must meet to perform "workflow mining": an event in the log must refer to a task, each event must relate to a specific case, and events must be totally ordered (p. 241). Similar to Hill and Jones (1999), the emphasis was case-based reasoning, and like that of Bae et al. (1999), used workflow systems. The van der Aalst et al. (2003) paper also proposed an XML DTD, thereby laying the groundwork for a common log format for workflow mining.

In a similar vein to van der Aalst et al. (2003), Cook et al. (2004) also examined issues of concurrency in workflow modeling. Two shifts from their previous work are evident. First, they began to use workflow concepts, drawing some inspiration from Agrawal et al. (1998). Second, they shifted from their previous reliance on FSM to Moore (state-labeled) or Mealy (transition-labeled) state machines instead. The use of these state machines provided concise visualizations of discovered concurrent relationships, but are simpler than "more powerful notations such a Petri Nets" (p. 302). In order to fully capture concurrency, an event log must contain information associated with an event, as opposed to the mere existence of an event. Therefore, events should have related attributes, one of which should be the time of the event. In van der Aalst et al. (2003), event timing was initially present in the logs, but the derived process model did not depend upon the recorded time. Indeed, the log format van der Aalst et al. (2003) suggested did not define any attributes. Instead, the ordered events were sufficient for deriving the workflow

model. In van der Aalst (1998), the authors noted that it is possible to extend Petri Nets to model time elements, and suggested including the resources associated with an event. Hill and Jones (1999) also suggested coupling attributes with events. For Cook et al. (2004), including the time attribute not only supported event ordering, but also allowed them to distinguish between activities that were "instantaneous" and activities of longer duration – an idea seen in their earlier work. They modeled the latter as two events with a duration span. This distinction between instantaneous and longer running events would allow for understanding process throughput and potential inefficiencies. In addition to the time attribute, the authors suggested including agents, resources, and "any other information that gives character to the specific occurrence of that type of event" (p. 298) in the event log. For the purposes of the paper, however, the authors did not incorporate these other semantically relevant attributes, though they acknowledged the potential value of these additions for future work.

Whereas Cook et al. (2004) suggested the use of elapsed time, Pinter and Golani (2004), directly examined the application of elapsed time to workflow model development. Earlier works considered each activity to be an atomic (or instantaneous) event, Pinter and Golani considered an activity as having an explicit life span based upon a starting and ending event present in the logs. This approach allowed for explicitly recognizing concurrent activities and resulted in a more accurate process model graph when compared to other algorithmic approaches, such as those proposed by Agrawal et al. (1998). The resultant model in Pinter and Golani might have been more correct, but at an increased burden to be met in the logged information.

Additional Contributions to the Third Thread

A few other works provided additional contributions. Herbst (2000a, 2000b; 1998) examined the application of machine learning techniques to workflow systems. Schimm (2002)

provided an approach for mining processes, and an early software program to extract mined processes from algorithmically generated event-based data. Interestingly, Schimm situated his work in of KDD rather than workflow systems despite the focus on events. Cook and Wolf (1998a) noted LeBlanc and Robbins (1985)'s contributions for highlighting the concept of an event in a distributed system as a basic unit of analysis. Cook and Wolf (1998b) acknowledged the contributions of distributed debugging (e.g., Bates, 1988), with its focus on application behavior as exhibited in logs.

There have also been periodic attempts to summarize the evolution of Process Mining. Van der Aalst (2003) presented a summary of the field from 1998 through 2003. They credited Agrawal et al. (1998) with the first application of process re-discovery to workflow systems. They also acknowledged Cook and Wolf (1998a, 1998b) while noting potential deficiencies in the work, such as an inability to generate explicit process models. The summary article also noted the intersection of Process Mining with the Business Process Intelligence (BPI) literature, and provided a number of references in the BPI field. In general, the BPI literature operates against data warehouses rather than event logs, and focuses more on data mining to extract clustering and performance analysis; in contrast, Process Mining focuses on extracting causal relations (van der Aalst et al., 2003, p. 241). In van der Aalst (2012b), the authors re-visited Process Mining history, but acknowledged Datta (1998)'s work that was not elsewhere mentioned. Datta's focused on process re-engineering and the need to systematically and automatically discover the "as-is" model based upon an FSM.

Early Exploration Summary

The reviewed literature established the primary intellectual basis for Process Mining. In examining the literature, two distinct intellectual differences may been seen. In general, despite

acknowledging Cook and Wolf, van der Aalst and those who worked in his tradition, tended to focus on systems that "adopted workflow technology" (Weijters & van der Aalst, 2001). In contrast, another group following primarily in the Cook and Wolf tradition emphasized software behavior without necessarily relying upon workflow systems. Thus, there was an early difference between a focus on process re-discovery in order to understand a *system* versus an attempt to understand *behavior*. Eventually, these two traditions merged somewhat in that the van der Aalst tradition relaxed its focus on workflow systems, and began to consider process re-discovery from event logs generated by a variety of applications. For example, one may contrast the evolution in a span of ten years from van der Aalst (1998)'s argument based solely around workflow systems to Günther, Rozinat, van der Aalst, and van Uden (2008) and the application of Process Mining to medical devices. Nonetheless, the van der Aalst tradition continued to implicitly emphasize system documentation and process improvement rather than discovery of behavior.

As van der Aalst and Weijters (2004) suggested, workflow model discovery was an important part of the early research agenda. This focus on workflow-based approaches was ultimately limiting both practically and theoretically. In practice, organizations discovered workflow management systems that focused on process automation suffered failures, in part due to a lack of flexibility and diagnosis capabilities (van der Aalst & Weijters, 2004, pp. 231-232). The overly-specified (and overly controlled nature) of workflow systems also proved problematic for organizations, as problem resolution frequently required a more flexible approach (Bezerra & Wainer, 2008).

The literature therefore began to evolve, starting around 2006, moving away from strict workflow systems towards general process discovery. There were two main drivers for this change. First, industry practice moved from workflow systems to business process analysis

(BPA) systems (van der Aalst & Weijters, 2004). Second, workflow systems themselves contain subtleties – such as potential Boolean choices proscribing cycles – that constrained the ability to generalize process discovery. Despite changing focus, the basic logged entities were essentially fixed by the workflow systems and the experiences in deriving models from them in the context of the Workflow modelling approach. For example, a log file needed to contain activities that were tied to specific cases, and the events within a case must be ordered (or orderable). The log may be enhanced with actor or timestamp information. A log file might also contain additional relevant semantic contributions. These additions could help examine different "perspectives" within the data (e.g., Huang & Liu, 2005; van der Aalst, Reijers, & Song, 2005). These additions, however, might allow for adding additional information, such as weather conditions if the discovered process could potentially be influenced by such data (de Leoni, van der Aalst, & Dees, 2016). Thus, the earlier lessons were incorporated in terms of what should be present in a log, but the focus would henceforth be on more generalized process discovery rather than a more limited notion of workflow discovery. This approach to more general process discovery also allowed for an intellectual integration of examining a system and an ability to understand behavior as expressed in a process.

In terms of theoretical development, workflow systems represented only a subset of the data available that could be mined for important processes. Not all business processes are neatly contained in the context of a single workflow execution environment. Nonetheless, these various business processes are important from an overall organizational perspective. Useful data for understanding underlying processes might reside in logs generated from multiple disparate systems (Claes & Poels, 2014), or might reside outside the standard process system entirely. As an example of the latter, e-mail correspondence frequently contains important information not

necessarily tied to any particular system (Buffett & Geng, 2010). Most organizations have numerous process-based systems that are not directly "business" processes, but are amenable to analysis nevertheless. Later work on security analysis (e,g,, Accorsi & Stocker, 2012; Accorsi, Stocker, & Müller, 2013; Jans, Alles, & Vasarhelyi, 2013), medical device operation (e.g., van der Aalst, 2011a, p. 2), or health care provisioning (e.g., Montani, Leonardi, Quaglini, Cavallini, & Micieli, 2014; Perimal-Lewis, Vries, & Thompson, 2014; Vogelgesang & Appelrath, 2013) provided examples of Process Mining techniques applied to a diverse body of mostly unstructured data.

Process Mining Types

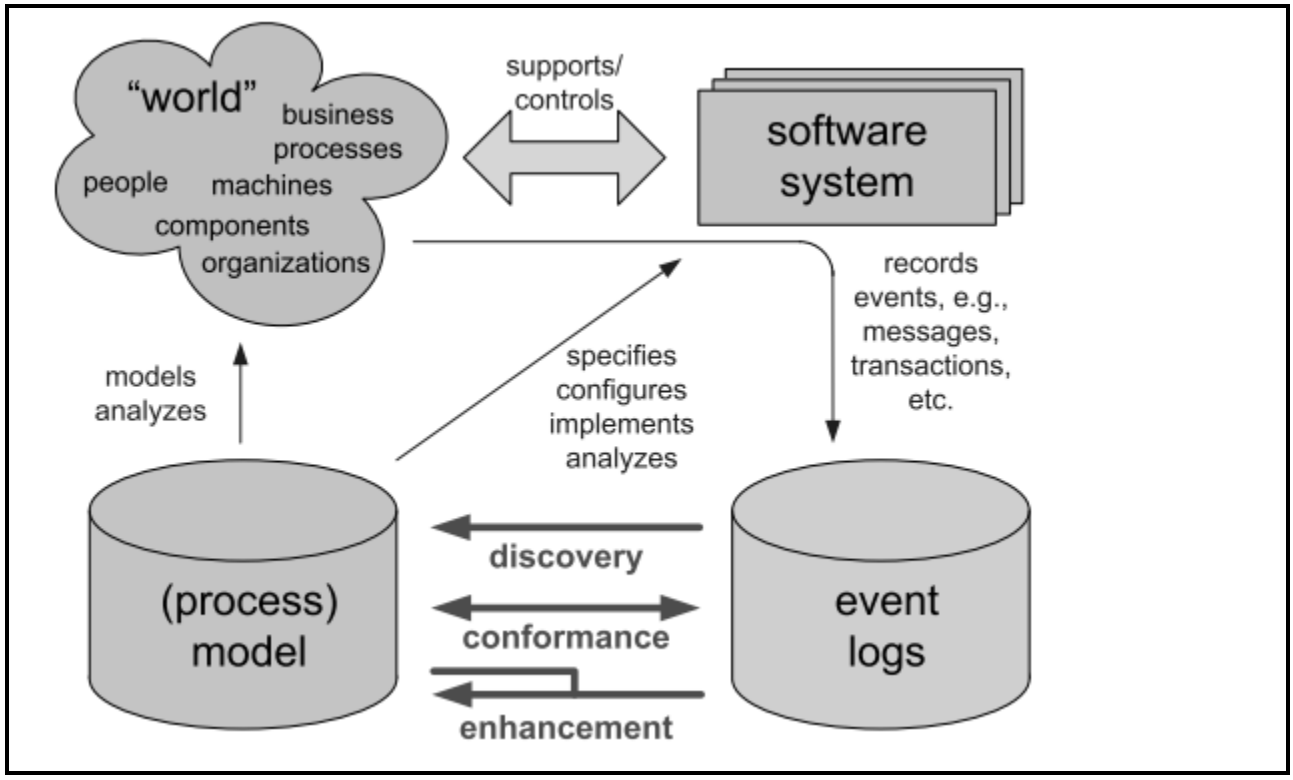

Figure 1. Process Mining Types. From van der Aalst et al. (2012).

In addition to expanding beyond workflow systems, the literature also began coalescing around a common understanding of three distinct "types" (van der Aalst, 2011b) of Process Mining. Eventually, these types (also referred to as techniques or approaches) would be termed

discovery, conformance, and enhancement. These types, and their relationships to event logs and process models, are shown in Figure 1 (van der Aalst, 2012b). The early Process Mining literature was essentially about the discovery of processes from a control-flow perspective mined from workflow system generated event logs. Van der Aalst (2005) was among the first to label the model re-creation from event logs as "discovery" events, though this terminology was reminiscent of Cook and Wolf (1998a). The discovery technique produces a model from an event log without using any *a priori* information (van der Aalst, 2011b); it is the extraction of a model from the event logs.

The conformance approach was the initial impetus for in-use/as-is process discovery. Initially Cook and Wolf (1994) investigated the conformance of a software development process to a given formal model, but used metrics rather than a mined – and therefore discovered – model. The initial investigations in what would be become discovery in the Process Mining terminology were seen in Agrawal et al. (1998) and Cook and Wolf (1998a). These approaches extracted a model from event logs to use as the point of comparison for conformance checking rather than using metrics. Thus the early metrics approach of Cook and Wolf (1994) or Henry, Henry, Kafura, and Matheson (1994) to validate conformance to a formal model contrasted starkly to later conformance checking approaches based upon discovered, behavior-based models such as seen in Rozinat and van der Aalst (2008). Ultimately, the conformance approach validates the actual discovered, in-use model against some other model. The comparison model may be a formal model (e.g., Huo et al., 2006) or another mined model.

The third type of Process Mining is enhancement, as the goal is "to extend or improve an existing process model using information about the actual process recorded in some event log. Whereas conformance checking measures the alignment between model and reality, this third

type of process mining aims at changing or extending the *a priori* model" (van der Aalst, 2011b). Process enhancement or process re-engineering that can be validated through simulation or subsequent mining may ultimately be the most useful application of Process Mining for an organization. The literature to date, however, has primarily focused around solving issues with process discovery and application of process mining to conformance checking. There are practical reasons for the dearth of literature around this enhancement type. First, many process enhancements entail improvements in an organization's competitive advantage. Publishing the exact enhancements would therefore be problematic. As an example, Hammori, Herbst, and Kleiner (2006) examined a discovered process from a commercial car company, but the supplied graphics had all specific details (such as the node labels) removed. Understanding the exact enhancements and benefits was therefore reduced. Second, the theoretical issues with process model discovery and the quantification of model equivalence for conformance checking are ongoing and non-trivial intellectual problems. In order for enhancement activities to be beneficial, these theoretical problems must be well addressed. Gupta and Sureka (2014) noted the "major research focus [of Process Mining] is towards designing algorithms for process discovery," thereby implying the algorithms require additional refinement. Third, process enhancement is the latest of the three approaches. In order to apply Process Mining for enhancement, issues with process discovery and conformance must be reasonably resolved. As Hammori et al. (2006, p. 43) noted, "To be able to redesign the process/workflow, the primary information needed is how the process participants work with the new system and where and why they deviate from the intended process." Early needs for conformance checking drove advancements in the algorithms for process model discovery; process enhancement did not fit as neatly into the initial investigative agenda.

Maturation

An article by van der Aalst and Weijters (2004) served as a definitional moment in the literature, proposing a consolidation of the efforts around “Process Mining” (and the associated terminology), as well as enumerating a number of challenges that needed to be addressed in order to continue to advance the process mining research. After this paper, one saw an increase in the number of domains to which Process Mining was applied, as well as the continued refinement of the techniques utilized by Process Mining practitioners. In addition, the literature itself began to adopt the “Process Mining” label rather than the previous workflow terminology. The intellectual origins of Workflow Mining, however, continued to have reaching effects. For example, Process Mining continued to emphasize Process Aware Information systems with more structured processes (as opposed to *ad hoc* ones; Dustdar, Hoffmann, and van der Aalst (2005) would acknowledge some of the challenges and opportunities in *ad hoc* processes in the year after this definitional article), and with that emphasis tended not to consider as deeply issues of data amalgamation. Nonetheless, the paean of “Process Mining” as a terminology and specific research agenda marked the division between the early phase and the on-going maturation phase.

As van der Aalst and Weijters (2004) noted, a range of theoretical issues needed to be addressed before Process Mining could be generally applicable. The article discussed issues with hidden or duplicate tasks, non-free-choice constructs, loops, accounting for time, different perspectives (e.g., control-flow, organizational/resource, etc.), noise, incompleteness, data from heterogeneous sources, and delta analysis (a part of the conformance approach). Folino, Greco, Guzzo, and Pontieri (2009) provided definitions for several of these issues. Hidden tasks are “routing activities that are not registered in the log (e.g., they can be used to skip the execution of some task).” Duplicate tasks occur “when a task identifier appears in two (or more) different points of the process at hand”. Non-free-choice constructs “occur when the choice of the next

activity to execute has to be synchronized with the execution of some activity different than the current one." Noise in the log refers to situations where the "log is incomplete, contains errors, or reflects exceptional behaviors" (Folino et al., 2009, p. 162; see also, van der Aalst & Weijters, 2004, pp. 235-239).

Early work in Process Mining focused almost exclusively on the discovery approach, and with a control-flow perspective (an issue critiqued by Calvanese, Giacomo, & Montali, 2013). It was assumed that a single entity created the logs; multiple input streams for log creation were not generally considered. Furthermore, the logs were assumed to be essentially clean (i.e., without noise, all tasks present in the log, etc.). Without resolution of the theoretical issues, Process Mining would have remained workflow mining; an exercise in extracting processes derived from a normative, monolithic model expressed in a limited range of possible event logs.

In this shift from monolithic, pre-defined processes, Dustdar et al. (2005) examined the application of mining techniques to ad-hoc business processes. In contrast to the assumptions that a normative model existed in some form, ad-hoc processes may be seen as a special category of processes without any underlying process definition. Nonetheless, the existence of ad-hoc processes directly addressed "in-use" process questions, and to a certain extent mining these processes addressed the hidden task mining challenge noted by van der Aalst and Weijters (2004). When considering user behavior as seen in application interaction, ad-hoc processes are the dominant process type. Günther and van der Aalst (2007) also considered ad-hoc processes, and believed that discovering these types of processes would "unveil previously hidden knowledge" (p. 328).

Alves de Medeiros, van der Aalst, and Weijters (2008) examined the issue of quantifying process equivalence. The general question was to indicate not only if two processes differ (a

binary true/false question), but by what degree. Furthermore, there were theoretical issues with what constituted similarities and differences between models. To examine these similarities and differences, they reduced models discovered via Process Mining to include only typical behavior. Utilizing concepts of precision (is a second model's behavior possible given the first model's behavior) and recall (how much of the first model's behavior is covered by the second), the authors presented a quantifiable measure of process equivalence. The work is important primarily in conformance checking approaches, since it is not enough to only understand if a newly mined model is different, but by how much it differs as well. The approach also has application to process enhancement, especially via simulation, since models may be repeatedly designed and simulated (or re-discovered), with a goal of maximizing the conformance coefficient. The issues of process equivalence were also examined in van der Aalst (2005), Hammori et al. (2006), Rozinat and van der Aalst (2008), and Caron, Vanthienen, and Baesens (2013).

Kovács and Gönczy (2008) examined the ability to formally validate process models. Similar to (Bae et al., 1999), they also looked at the ability to simulate the models in a deterministic fashion. Formal proof of a model supported the validity of Petri Nets, and provided a means of translation between modeling languages (such as Business Process Execution Language (BPEL)) and discovered models. BPEL is potentially problematic as a means for expressing process models that could be used in conformance checks. Issues arise from vendors not fully implementing the BPEL specification (van der Aalst, 2012c), the monolithic nature of the expressions ("all activities at all the different levels refer to status changes of the same process instance" (van der Aalst, 2012c, p. 564)), and a focus on a process at the expense of the operational data (Calvanese et al., 2013). In addition, BPEL models, or models generated from

the Business Process Modeling Language (BPML) are not necessarily valid, in that they may contain deadlocks or unreachable activities. Rozinat, Mans, Song, and van der Aalst (2009), Rozinat, Wynn, van der Aalst, ter Hofstede, and Fidge (2009), Giuseppe, Valerio, Teresa, and Carmela (2014), and Khodyrev and Popova (2014) examined similar issues with simulation and process modeling.

Folino et al. (2009) examined issues of duplicate and hidden tasks, noise and non-free choice relationships. In this paper, the authors proposed a new algorithm to deal with these various issues. Earlier approaches used a local search space in order to re-create the process model. A local search relies solely upon "knowledge about the tasks that directly precede or succeed one other given task in some process instance" (p. 162). In theory, an approach that utilized a more global approach would overcome some of the issues. For example, approaches based upon genetic algorithms address noise better than more simplistic local searches such as the standard alpha miner. Genetic miners can use a more global search space, but may have issues with producing a properly fitted model (p. 162). (see also, Alves de Medeiros, Weijters, & Van der Aalst, 2006). Folino's approach, therefore, relied upon an augmented local search strategy, culminating in an "Enhanced WFMiner" algorithm. Wen, Wang, van der Aalst, Huang, and Sun (2010) also examined the hidden tasks issue (called "invisible tasks"), and provided an extension to the traditional alpha algorithm to support their inclusion in a discovered model. Ma, Tang, and Wu (2011) examined issues of loops and parallelism based upon an incremental mining method.

In (van der Aalst & van Dongen, 2013), issues with noise and incompleteness were examined. The resultant model used a Petri Net, and raised arguments against block-based

approaches. The article also articulated four criteria a process discovery technique needed to balance:

> *fitness* (the discovered model should allow for the behavior seen in the event log), *precision* (the discovered model should not allow for behavior completely unrelated to what was seen in the event log), *generalization* (the discovered model should generalize the example behavior seen in the event log), and *simplicity* (the discovered model should be as simple as possible) (p. 376, emphasis added).

The article provided critiques of the standard alpha algorithm, noting its issues with loops, concurrency, and noise (see also, Weijters & van der Aalst, 2003). It discussed different mining approaches, such as genetic mining, heuristic mining, fuzzy mining, and synthesis based on regions, and provided a detailed review of region-based mining approaches. Genetic mining approaches were examined in (Alves de Medeiros et al., 2006; Alves de Medeiros, Weijters, & van der Aalst, 2007; Bratosin, Sidorova, & van der Aalst, 2010; Turner, Tiwari, & Mehnen, 2008). Fuzzy mining was explored in Günther and van der Aalst (2007). A heuristic based miner was developed in Weijters and van der Aalst (2003) to deal with noise and incompleteness. Evermann and Assadipour (2014) examined issues of the standard algorithms in the presence of large data sets, proposing the application of map-reduce approaches to support massive data sets. Yue, Wu, Wang, and Bai (2011) provided an overview of mining algorithms by author, noting some advantages and disadvantages of each algorithm. This article also provided information on algorithms developed in the Chinese literature that were not frequently cited or discussed elsewhere in the English literature.

In 2011, an IEEE task force published the “Process Mining Manifesto” (van der Aalst, 2011b; van der Aalst et al., 2012). The manifesto put forth a canonical definition of Process Mining and its components:

> The idea of process mining is to discover, monitor and improve real processes (i.e., not assumed processes) by extracting knowledge from event logs readily available in today's (information) systems. Process mining includes (automated) process discovery (i.e., extracting process models from an event log), conformance checking (i.e., monitoring deviations by comparing model and log), social network/organizational mining, automated construction of simulation models, model extension, model repair, case prediction, and history-based recommendations.

The manifesto also enumerated six guiding principles, a simplified lifecycle for Process Mining (similar to, but less robust than, the KDD process), a maturity model for event logs, and 11 challenges that remained problematic. Two of the challenges addressed non-experts utilizing Process Mining tools and techniques. Other challenges remained from the van der Aalst and Weijters (2004) article, such as incompleteness and noise. The manifesto also noted that data may be distributed and in need of merging, as well as the potential for large event logs. It also re-affirmed the need to balance the quality criteria of fitness, simplicity, precision, and generalization. Unfortunately, the balancing necessary to achieve quality currently requires experts during the Process Mining task, and it is difficult to specify the trade-offs *a priori*. Finally, it noted that an XML-based standard for event logs (XES) had been adopted.

Maturation Summary

From the van der Aalst and Weijters (2004) article until today, Process Mining has continued to mature. The literature surrounding Process Mining has addressed, if not fully

resolved, the challenges identified in that article. The Manifesto, backed by a number of academic and industry participants, provided definitional support for Process Mining concepts. Process Mining has been applied to a number of case studies, demonstrating its effectiveness in discovering processes from event logs in various domains. There are also several Process Mining tools, both open source (ProM) and commercial (e.g., Disco) that support the process discovery. Consequently, Process Mining is a viable approach for the recovery, visualization, and comparison of processes extracted from event logs.

Process Mining for Software

In the foregoing review, the focus was on general Process Mining techniques and challenges. A few case studies were noted, but no specific investigation of applying Process Mining to the software development and software quality question was presented. This section examines the literature with specific reference to the application of Process Mining to software, rather than business processes.

Process Mining seeks to discover actual processes from logged data. One potential application of Process Mining is to the software development process itself. Software is developed using a process, whether that process is formal and plan-based, a part of an Agile methodology, or simply ad-hoc. It is a general article of faith in the software engineering field that the process that is used “to develop and maintain software significantly affects the cost, quality, and timeliness of software products” (Henry et al., 1994, p. 67). Furthermore, “the software development process is widely recognized as a key factor that contributes to the quality of software” (M. Valle, A.P. Santos, & R. Loures, 2017, p. 19). In the traditional approach that relates development processes to software quality, a positive benefit to the resultant software should occur from understanding, monitoring, and improving the software development process.

Prior to the Process Mining movement, Henry et al. (1994) utilized a process-assessment methodology to evaluate software development at Martin Marietta. Though the techniques were based upon statistics rather than a re-created graph, the careful documentation of the normative and actual processes aligns with the general Process Mining literature. Their goal was not only to evaluate existing processes, but also to be able to quantitatively evaluate the effectiveness of process improvements. Had Process Mining, as a body of literature existed, the study at Martin Marietta would have been an interesting application. To a certain extent, the work in Alves de Medeiros et al. (2008) would have provided the underlying quantitative comparison approach that Henry et al. had to develop via statistical means. The delta analysis approach described in van der Aalst (2005) or the process equivalence detection of Alves de Medeiros et al. (2008) would also have been useful. Nonetheless, the article demonstrated that software development produces artifacts that are amenable to process analysis.

Rubin et al. (2007) examined the log of a software configuration management system, and applied Process Mining algorithms to derive the actual, in-use process model. The authors suggested that understanding the actual, discovered model addressed deficiencies with the standard process-centric software engineering environment. The primary deficiency is that the designed model is prescriptive, and does not "necessarily reflect the actual way of work in the company" (p. 173). The authors also addressed issues of abstracting and filtering the event log, as the document logs contained too many details or overly specific document names. The fundamental issue, therefore, was the need to derive a mapping from application behavior to user behavior. The authors examined the discovered process from the resource, performance, and information perspectives. When the log was mapped such that user behavior was the primary

event, and the event was associated with a particular author and timestamp, the authors were able to address additional perspectives to provide a richer description of the discovered process.

Several authors examined the history contained in source code control systems. Zimmermann et al. (2004) sought to find common related changes in source code. The goal was to build a program that could provide guidance to a developer when the developer modified source code. Essentially, the examination of the repositories revealed that source code modification contained a process whereby modifications in one part of the source code typically resulted in modifications in another part. Thus, if a given function/method were changed, it was likely that an associated function/method would need to be changed as well. Software quality could therefore be improved by ensuring that requisite coupled changes were properly addressed.

Huang and Liu (2005) examined version histories with the goal of finding "peripheral participants." This approach was essentially a social network/organizational perspective applied to the version logs. Peripheral participants may make significant contributions to a software project, but their lower profile due to fewer commits (absolute numbers or number of modified modules) was shown to be an impediment to their full inclusion in a project. These authors provided two insights. First, there were a great number of processes that were implicit but important. These processes may be discovered and utilized for improvement activities. Second, there were a number of legitimate peripheral participants who were often overshadowed by more vocal or aggrandizing individuals.

Gupta (2014; Gupta & Sureka, 2014; Gupta et al., 2014) examined software logs, including version histories and issue tracker entries. This series of articles explored software process improvement, as well as ways to identify various classes of participants involved in the

development of several open source projects. These works progress from process discovery in a single type of event log (version history) through the mining of multiple heterogeneous logs.

Mittal and Sureka (2014) mined a set of heterogeneous logs to provide insight into the "practices and procedures followed during various phases of a software development life-cycle" of undergraduate students in a software engineering class (p. 344). The goal was to provide feedback to the instructor on the development process and quality of the development activities. The work was similar to Gupta's, but with a focus on instructional improvement rather than software development improvement *per se*.

Requirements analysis was examined in Cleland-Huang and Mobasher (2008) and Duan et al. (2008). In these articles, the authors asserted that an increasing number of requirements from an increasing number of stakeholders would render traditional approaches to requirements analysis obsolete due to volume. In these "ultra large scale" projects, it will be impractical for requirements specialists to elicit, categorize, and prioritize requirements. Thus, the ability to cluster requirements into sets of related functionality, as well as to avoid issues of undue stakeholder influence in requirements prioritization, will be critical in the near future for software development and acceptance.

Huo et al. (2006) examined a discovered process from developer activity logs. The discovered process was then compared to an exemplar ISO 12207 software development process adopted by a firm. The observed differences were then used to modify practices on the part of the developers or suggest changes to the firm's exemplar process.

Sun et al. (2013) also sought to improve the software development process by extracting the in-use process as an input to a maturity calculation. Whereas Huo et al. (2006) contrasted the discovered process to an expected process, Sun et al. sought to reduce the dependency on a

manual evaluation performed by subjective experts to understand process maturity. The stated assumption was that process conformance would lead inexorably to improved software quality.

In contrast with the above authors who examined the software development process, a few authors have examined the application of Process Mining directly to a software application. Rather than analyzing a development process, the general intent was to analyze a software program and use the collected information to improve the software experience.

Günther et al. (2008) examined applying Process Mining techniques to "deployed applications, i.e., some application [that] is used at different sites" (p. 1). The authors suggested that it was possible to describe the actual use of the application, revealing typical usage patterns as well as certain features that were "never used or that… [were] only used by particular customers under particular circumstances" (p. 1). In addition to discovering user-behavior, the authors suggested that the discovered models could also be used for reliability improvement, usability improvement, and remote diagnostics. A key aspect of the approach relied upon the remote applications posting event logs to a central service. The case study concerned medical devices, and thus was similar to Maruster et al. (2001) or Montani et al. (2014). The article raised issues concerning data privacy. Günther et al. (2008) also noted the case study application posted events that were very "low level, representing single commands of the system" (p. 3). Thus, deciphering user behavior from the logged application behavior was a challenge, and required a complicated process (reminiscent of the KDD processes) to clean the logs given the diversity and complexity of the logged events. Low level application logging also generated a large amount of data. Such large amounts of data "excludes any kind of interactive analysis, since most mining algorithms scale with the size of the log" (p. 4). The authors also noted deployed applications exhibit "*ad-hoc* process[es]" (p. 4). Even when "short episodes of using the system may be well-

structured," the goals and means vary across invocations of the application. The authors concluded that:

> For any sufficiently advanced and complex application or machine, the manufacturer typically has a very limited understanding of how exactly the system is being used in practice. This is especially true for generic applications (e.g., tools) and highly configurable, interactive applications (e.g., software)" (p. 7).

Zhao, Liu, Ye, and Wei (2013) sought to understand the "characteristic pattern of daily activities" as recorded in event logs from the utilization of software on a local computer as well as website access. The authors did not utilize traditional Process Mining approaches, relying instead upon an expectation maximization algorithm. Nonetheless, the authors referenced the Process Mining literature, and the discovery of behavioral patterns from event logs fits within the overall approach of applying Process Mining to a software application.

Rubin, Mitsyuk, et al. (2014, p. 1) focused on user behavior in an application, and proposed that, "When the [software] systems are utilized, user interaction [should be] recorded in event logs." The authors suggested that understanding the recorded user interaction would allow for improving software, as well as monitoring and measuring its real usage. The article's focus was on "user workflows" and the associated expression of user behavior exhibited at runtime. The authors captured data from both acceptance testing and when the system was "productively used", providing input from testers or stakeholders, as well as from actual end-users. The authors concluded that software process mining (SWPM) was a valid domain for Process Mining techniques. SWPM provided critical insights into the actual (as opposed to theoretical/normative) utilization of a software program.

In Rubin, Lomazova, et al. (2014), the SWPM was used as a part of an Agile development process. The authors discovered that processes extracted from the event logs as users interacted with the system differed from expectations expressed in the functional specification. Thus, the in-use processes of users as they sought to achieve their goals varied from the normative expectations of the developers and the "representative" stakeholders that originally defined the system. SWPM also provided a global overview of the application that was not provided by the standard development approach. Applying SWPM to the application revealed critical performance issues, but these issues were identifiable only through the complete exercise of the program, rather than in isolated features or individual implemented user stories. The ability to provide actual in-use process maps allowed for extended conversations with the stakeholders. Interactive discussions with stakeholders during the development cycle are an integral aspect of Agile's philosophy (Boehm & Turner, 2004; Maurer & Melnik, 2006; Rubin, Lomazova, et al., 2014, pp. 72-73; Schwaber & Beedle, 2002).

As noted elsewhere, Khodabandelou et al. (2014) applied Process Mining to discover intentional actions on the part of users in the Eclipse IDE. Here the event logs generated from the Eclipse Usage Data Collector were utilized to map the means and goals of developers through the features they utilized in the software. In van Genuchten et al. (2014), the authors applied Process Mining to a software program that contained an embedded workflow process. Rather than the ad-hoc processes of Rubin, Lomazova, et al. (2014) or Khodabandelou et al. (2014), the dental software that was analyzed had an explicit process within it. Nonetheless, mining the process revealed unexpected pathways in the software, as well as performance issues. The authors concluded that employing Process Mining in their Agile development process gave "users a more dominant role in the development phase" (p. 99).

Summary

"Process mining deals with discovering, monitoring and improving real processes by extracting knowledge from event logs available in modern information systems" (Rubin, Lomazova, et al., 2014, p. 70). The history of Process Mining originated with the issues associated with process discovery, and it was heavily influenced by workflow systems. Nonetheless, a second intellectual tradition of behavioral rather than pure procedural process re-creation was evident. Recently the literature has been less emphatically restricted to workflow systems. The Process Mining Manifesto emphasized the need for *events* recorded in *logs*, and for the events have orderable *activities* associated with a particular *case* contained within them. The business process and workflow system restrictions are thus no longer as evident.

Utilizing Process Mining requires algorithms to analyze the event logs, and there continue to be theoretical issues associated with process discovery. Academic and commercial investigations continue to address these issues. In addition to the traditional control-flow process discovery, other perspectives such as organizational or resource cost models are also discoverable. Case studies have been applied to a variety of areas, some with a more business process focus (e.g., van der Aalst, 2011a), and some with a greater behavioral focus (e.g., Jans et al., 2013).

Process Mining has also been applied directly to software applications. In these investigations, the goal was the discovery of the pathways (processes) that users take in order to achieve goals. The software may include expected workflows, but the ad-hoc processes implicit in software applications can be discovered, compared, and analyzed.

Process Mining therefore appears to be a fruitful approach for understanding how individuals utilize software. From this understanding, it is possible to improve software quality by incorporating the lessons extracted from the process logs into the SDLC. The event logs may

be generated either during the development phase, or during normal program utilization. By utilizing the discovered processes, and the deviations from developer or stakeholder expectations, conversations may be generated in an Agile methodology. It is possible, though not explored in the literature, that more traditional plan-based SDLCs could utilize the discovered processes as well.

Chapter 3: Considerations for Event Log Creation

Process Mining requires input from an event log. The Process Mining literature is replete with phrases that suggested the logs "can be easily extracted" (Weijters & van der Aalst, 2001, p. 93) or "are readily available" (van der Aalst et al., 2012, p. 1). These logs may also be "referred to as ''history', 'audit trail', 'transaction log', etc." (van der Aalst et al., 2007, p. 713), and frequently the literature assumed a "Process Aware Information System" (PAIS) (Rozinat & van der Aalst, 2008, p. 64) that produced them. As noted above, the literature has evolved to not strictly require a PAIS to produce the log, but nonetheless an event log must exist.

The literature has provided rough guidelines as to the contents of an event log. Following (van der Aalst, 2012b, pp. 1-2), each event in the log refers to an activity. An activity is a well-defined step in some process. Each activity is bound to a specific case, which is a particular process instance. The events in a case must be orderable, and a case may be seen as one "run" of the process. Earlier, it was noted that the activity must also be one that provides value to the user.

One potential issue when considering an event log is the continued assumption of an underlying normative process. Despite the assertions that, "We do not assume the presence of a workflow management system. The only assumption we make is that it is possible to collect workflow logs with event data" (van der Aalst et al., 2004, p. 1128), the language as evinced in the Process Mining Manifesto continued to utilize terms such as "process instance." Pérez-Castillo, Weber, de Guzmán, and Piattini (2011) also noted this underlying assumption: "Unfortunately, process mining focuses on process-aware information systems" (p. 304). Yet these authors were interested in applying Process Mining to legacy applications that were not a PAIS, and demonstrated the generation of event logs with ordered events. This case study, among others, showed the need for ordered events related to a case was not necessarily an

unduly onerous burden, and need not be directly tied to some implicit process management within a software application.

When one considers applying Process Mining to a non-PAIS software application, the ready availability of event logs is not guaranteed. As noted above, such non-PAIS software may only emit logs of interest to developers, or the log may be application focused rather than user behavior focused. Interestingly, even in the case studies directly addressing the application of Process Mining to software applications, the fundamental definitions of what developers should consider emitting into an event log are absent. It is one thing to discover the process as seen from a PAIS with event transition logging; it is quite another when the application is desktop application implemented without consideration of generating an end-user, behavioral focused event log.

This chapter, therefore, collects and reviews the scattered suggestions for generating an application event log. Understanding how users interact with an application requires that the application log said interaction. Moreover, the interaction that is logged is most useful for documenting the implicit processes that users take in pursuit of their goals if the activities are meaningful and related to user-initiated activities. It is, in almost all applications, possible to collect every mouse move, mouse click, and keyboard interaction. In certain circumstances, this information could be valuable. For example, in User Experience (UX) design, understanding the mouse distance could provide insight into usability questions. The focus in this thesis, however, is not on UX directly, but rather on how users interact with the application in pursuit of goal realization. Thus, the driving questions are, as an example, related to the means by which data are created and modified in an application rather than, for example, how many menus must be navigated in order to access a feature. Applying Process Mining to UX is a potentially interesting

area of investigation, requiring a different sort of event logging, but likely amenable to Process Mining techniques. Nonetheless, such an investigation is beyond the scope of this thesis.

This section presents a basic example event log, and illustrates the discovered process as a Petri Net. The primary purpose of this presentation is to provide a visualization of the Process Mining approach reviewed above. This section then presents a definition of a case and a definition of an activity. It considers questions of instrumenting an application as opposed to intercepting activities. It then briefly discusses considerations of privacy in relation to event logging. Without the ability to collect information, which may be limited by privacy concerns, and the ability to log events as activities associated to a case, it is not possible to apply Process Mining techniques. Thus, while Process Mining has the potential to reveal insights into the behavior of users in relation to software applications, Process Mining has strict requirements in order to operate.

This section also has a particular bias towards understanding user behavior in a single, non-PAIS application. In addition, the implied purpose is to enable a development team to apply Process Mining to event logs in order to improve the software during a development cycle, as well as to capture data from a broader user community in order to validate development assumptions. At an enterprise level, Process Mining may be applied to a large number of disparate systems, involving a variety of log types, and requiring an expenditure of effort in order to reconcile these various logs. To a certain extent, in order to support smaller development teams, the issues associated with enterprise-level event reconciliation serve as the boundaries of what a team instrumenting an application should consider. For example, if one had an application that invoked a number of smaller constituent parts, it is conceivable for each part to emit an independent event log with different types of events unrelated to an overarching case. Such an

approach is, in effect, rejected in the suggestions below. While the text will review the literature on how such situations might be addressed, in general it is better to carefully define a case and associated activities than to apply various remediation techniques. Nonetheless, in understanding the struggles of such remediation, the rationale for the recommendations is made clearer.

Example Event Log

An event log need not be expansive: "The logging requirements for process mining are simple" (van Genuchten et al., 2014, p. 98). In van der Aalst and Weijters (2004), a table consisting solely of a case identifier and a task (elsewhere the "event" or "activity") was presented (p. 232). Table 2 provides such example entries. The table structure implied the order of events. From this table, a Petri Net may be discovered, as shown in van der Aalst and Weijters (2004). The Petri Net that was discovered from the entries in Table 2 is shown in Figure 2. In essence, from the simple logging of activities tied to a case and emitted in a sequential manner the process is discoverable and may be presented as a Petri Net. This discovered net also demonstrated the source and sink (start and finish) of the process despite the absence of specific activities associated with the process initiation or termination. In van der Aalst et al. (2004), the authors provided a formal definition of an event log.

Table 2. Example Entries for an Event Log

Case Identifier	Task Identifier
Case 1	Task A
Case 2	Task A
Case 3	Task A
Case 3	Task B
Case 1	Task B
Case 1	Task C
Case 2	Task C

Case 4	Task A
Case 2	Task B
Case 2	Task D
Case 5	Task E
Case 4	Task C
Case 1	Task D
Case 3	Task C
Case 3	Task D
Case 4	Task B
Case 5	Task F
Case 4	Task D

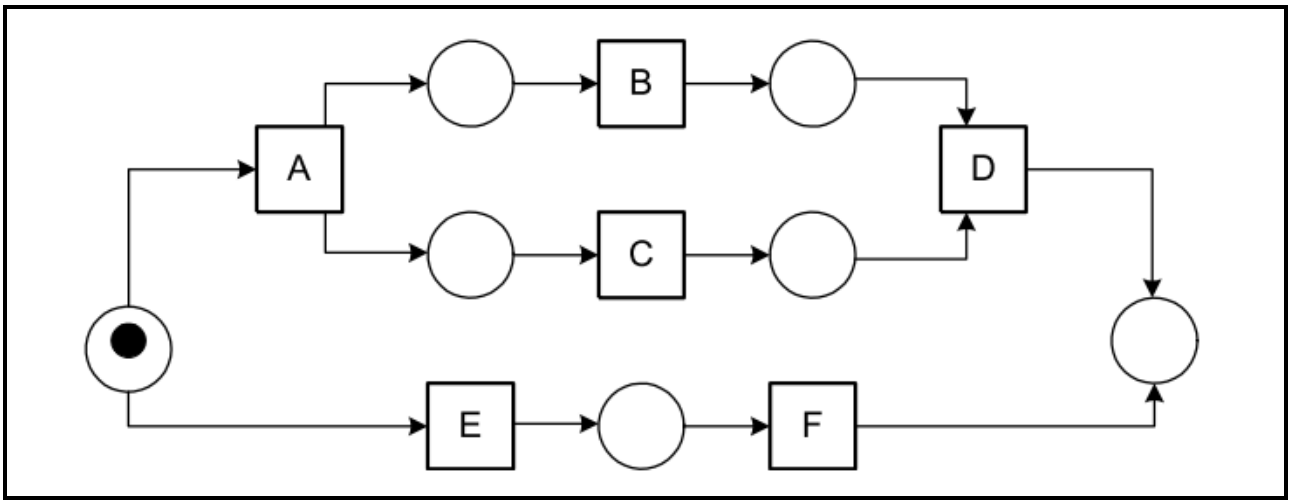

Figure 2. Example Petri Net. From van der Aalst and Weijters (2004)

In van der Aalst et al. (2005), the event table was augmented with the individual associated with the task (the performer). Dustdar et al. (2005) suggested that an event table should always contain the performer and a timestamp. Other relevant data entities may be present as well (Cook & Wolf, 1998a; van Dongen & van der Aalst, 2005), though the exact nature of this additional data is domain specific (Fayyad et al., 1996). To achieve a fully data-aware process discovery (Calvanese et al., 2013), an event log must be more than a mere ordered list of the tuple of cases and activities. The exact additional attributes contained in the log

requires input from the domain experts involved with the application. However, the inclusion and analysis of domain specific data are beyond the scope of this current investigation. The presence of these additional domain specific data attributes is potentially useful, but not required for process discovery. The presence of an actor or resource supports mining from additional perspectives. The presence of a timestamp not only assists with the ordering, especially in larger event streams (Appel, Kleber, Frischbier, Freudenreich, & Buchmann, 2014) where events could potentially arrive out of strict chronological order, but also provides meaningful insight into bottlenecks. As noted elsewhere, at least two case studies identified performance issues with the processes discovered from the event logs, and timestamp information is important for such throughput analysis.

Definition of a Case

At first glance, it would seem that defining a case should be simple. A case associates a set of events to a unique instance of a given process flow. However, the scope of a given process flow is not necessarily obvious. One advantage of the reliance upon a PAIS in the Process Mining literature is that a case is a natural part of the software system. In essence, a given workflow instantiation constitutes the case, and all events associated with that workflow instance are therefore associated with a case. The largest problem, therefore, would be ensuring a unique case identifier exists and is consistently utilized for a given process instance. The studies of health care (e.g., Maruster et al., 2001; Perimal-Lewis et al., 2014; Vogelgesang & Appelrath, 2013), for example, revolved around a single patient visit. When considering fine payment (van der Aalst et al., 2003; van der Aalst & Weijters, 2004), the case spanned from the issuance of the ticket to the payment of the fine. In examining an appeals process for property tax assessment (van der Aalst, 2011a, 2011b), a case started with a citizen’s form submission, and concluded

with a finding for or against the citizen. In these instances, a case is easily determined, contains the requisite source and sink points, and is not necessarily tied to a single software application since a workflow controlled the overall flow.

In contrast, consider an e-commerce site. Is the case a single web-browser session? If the system maintains a shopping cart across sessions (such as Amazon), is a case from the initial connection to a purchase? Maintaining information on a single session (identifiable by a session id) is potentially easier and would provide insight into the behavior of users at a given point in time, but may not capture the entire transaction history (see, Zou & Hung, 2006). Understanding the ways in which people interact with the site, and whether a session concludes with a purchase or departure is also highly relevant. In examining an application for the tour agent industry, Rubin, Mitsyuk, et al. (2014) used a single web-like session as the defining case, despite the fact that a customer might inquire one day, and return and purchase a ticket another day. In examining a dental software application, van Genuchten et al. (2014) utilized a patient's mold generation from the initial dentist visit to the installation of the dental prosthesis. In this study, the case is closer to the health care case studies, as the application could be launched multiple times, than to the tour agency. In addition, the software contained a type of workflow management. In the case of a word processor, should one attempt to define a case as all activity related to a particular file (if it were possible to unambiguously identify a file), or a given execution of the program? Returning to the Alice conundrum, is a case better defined by a complete analysis of a given product (which may have multiple simulations), or a given execution of the simulation software?

In van Dongen and van der Aalst (2005) the authors attempted to clarify these differences by creating a spectrum of PAIS applications (p. 3), and restricted their meta-model to essentially

PAIS applications. The article asserted that in contrast to standard PAIS systems, “Systems such as Lotus Notes provide a structured way to store and retrieve data, but no processes are defined at all” (p. 3). In the spectrum shown in Figure 3, applications such as Lotus Notes, or a word processor, would be located in the lower left corner. Yet these types of applications may still be of interest for mining user behavior.

The assertion that one would not find processes in applications such as Lotus Notes would be true only if one considers a formally defined workflow-like process, such as one would have in a PAIS. Further, it is an odd assertion that a data driven applications would necessarily preclude a type of process for dealing with the data over a life cycle, even if the processes were more idiosyncratic or *ad-hoc*. If one considers that in Lotus Notes an individual will utilize an *implicit* process in order to achieve a goal, then the statement is incorrect. This particular assertion was written, of course, earlier in the evolution of the literature. It was situated in the workflow process paradigm of software applications, as opposed to the behavioral view. Yet the article was fundamentally about the ontology required for event logging, and with the appropriate case definition and activity logging, there is no reason to exclude *a priori* applications such as Lotus Notes from Process Mining other than the fact such applications do not, as a matter of course, log user behavior.

A different approach to the division is found in Sun et al. (2014). These authors noted that attempts to classify software engineering data divided into approaches that focused on the relational aspect of the data and those approaches that focused on the textual and document oriented aspect of the data. In effect, the authors suggested that focusing on the how the data progressed through a series of status changes would afford a holistic view of a software project. Rather than attempting to approach the question from an initial perspective of a potential process

(recall Huo et al., 2006), the focus should be on how behavior was expressed in transforming data. This conceptualization is similar to Calvanese et al. (2013). These authors examined several

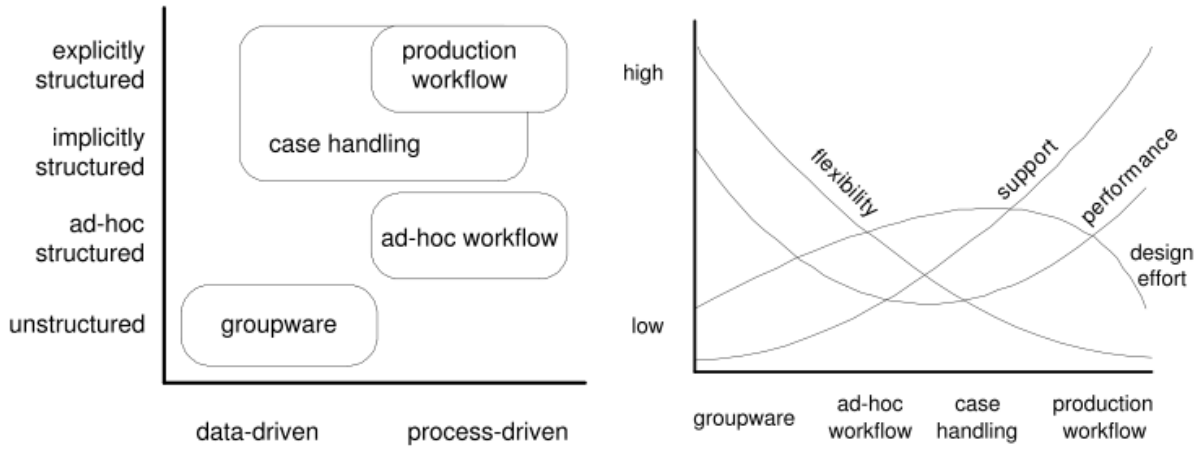

Figure 3. Structuredness and data-driven vs. process driven application. From van Dongen and van der Aalst (2005)

means by which business process analysis originating from a data perspective may be implemented. However, of particular note, the authors suggested that an "Artifact-centric approach provides a simple and robust structure for business process development, which has been advocated as superior to the traditional activity-centric approach, especially when dealing with complex and large process models" (p. 6). The artifact approach relies upon an implicit or explicit artifact lifecycle that ultimately has, "A procedural flavor, based on finite state machines whose transitions either create a new artifact, or modify/eliminate an existing one" (p. 6). In these document/artifact centered approaches, one can see a linkage back to the goals of users. A user does not utilize a process for the sake of the process, but rather because the process allows for some creation or transformation of an artifact. The underlying emphasis on PAIS applications in much of the Process Mining subsumes the goals of the user to the controlling process.

From the focus on data/artifact transformation, a more general approach to a case definition may be derived. Such an approach focuses on how users would transform data. This

approach remains compatible with PAIS approaches, since ultimately a business-process collects and transforms data in such approaches. Consider, for example, the property tax system case study noted above. A series of documents were collected, routed, examined, and used to render a judgment regarding an appeal. The authors mined a process controlling this document flow, but started from the process approach. It would have been equally valid to start with a behavioral approach of how users would interact with artifacts and their associated changes, and then discover the processes that would describe the means by which the documents are collected and transformed. In other examples, such as the tour agency, the hypothetical word processor, or the Alice story, assuming a process model first would be inappropriate. In these instances, the starting point should be on a bounded set of associated user behaviors focused on data creation or transformation. The events correspond, in these cases, to user actions that modify some ultimate artifact. It is certainly easier when a PAIS is present, since a case instance is more readily recognizable, but the "process first" paradigm is not a necessary pre-condition for discovering the behavioral processes utilized to interact with artifacts.

Another critical consideration surrounding a case definition is the need for an identifiable beginning (source) and end (sink) point. In van Dongen and van der Aalst (2005), the authors needed to infer the source and the sink of the audit log in the PAIS under investigation, and as such the ontology of the application suffered from a construct defect. Nonetheless, if the source and sink information can be inferred, it is possible to discover the process despite the ontological issue. If it is not possible to determine these starting and ending points, it will not be possible to utilize Process Mining techniques. The definition of a starting and ending point follow from the data manipulation considerations discussed earlier. However, there may be instances where complete data transformation cannot be easily tracked, and thus a case will need to be more

narrowly defined. In the tour agency case study, the TOMA protocol by which the software application clients interact with the back-end servers defined a start- and end-point for interaction. It does not appear this interaction necessarily spanned multiple instances of the client application. In the hospital study, the case was limited to a single patient visit, and did not attempt to span all visits by the patient. In the hypothetical word processor, a case is likely limited to – at most – a single invocation of the program, as tracking across multiple program launches would require a larger meta-conceptualization of the user's behavior. If one attempted to define the start point as the initial creation of a document in a word processor, how would one determine when the document was completed, since there is no necessary final state or time of such a document? This ambiguity drove the assertions that applications such as Lotus Notes lacked a process. Yet the fact that users exhibit behavior when interacting with a program simply indicates the need to define a case in a manner that allows for associating the behavior in a meaningful way with the application. If an application manipulates artifacts in a way that has a clearly defined start and end-point, then a case may be defined as relating to that artifact. Otherwise, other approaches must be considered, and it is possible a single web-session or application launch is the appropriate definition. As noted above, the ultimate purpose under consideration is the behavior that users exhibit in pursuit of their goals when interacting with a program. These goals must be achievable through interacting with the software application.

A case requires the ability to have events associated with it. In turn, every event in the log must be associated with a case. This requirement also provides a boundary for a case, as there must be a logical connection between the logged event and some case. Understanding the range of events under consideration may also provide insight into the proper delineation of a case. In van Dongen and van der Aalst (2005), the authors suggested a construction of a complete

ontology for events. The full ontology of events would provide guidance to the definition of a case. This approach is potentially useful, but also potentially quite involved, and it likely requires ongoing effort to maintain the vocabulary. In Rubin, Mitsyuk, et al. (2014), the events concerned tour reservations. The authors captured the events through the communication protocol, leading to a case bound to a session. Pérez-Castillo et al. (2011) also examined the binding of events to a case in legacy business applications that are not specifically workflow aware and do not, therefore, typically produce event logs. However, these authors suggested that, "Business experts and system analysts must provide this information to establish the scope of the business processes supported in the information system" (p. 306). While these authors had suggestions concerning what constituted an event (see below), they did not provide similar guidance to defining a case, with the exception of the tight coupling of an application activity to a domain-expert defined case. Such domain-experts might use a collection of Use Cases for the application as an input for the case boundary definition (van Genuchten et al., 2014).

If an application maintains state information associated with a given set of artifacts, and there is a definable end-point in the set of states, it is possible for a case to transcend a single instantiation of a given application. User behavior, therefore, could be tracked across multiple application launches, and events may be related to a case generated from the overall artifact lifecycle. However, if the application does not maintain a set of states across instantiations, or does not define a set of states, the most likely upper boundary on a case definition is a single application session, or some identifiable subset of behavior in the application. An application may allow for a variety of "tasks" to be accomplished in the pursuit of some particular goal (Pérez-Castillo et al., 2011), but without maintaining some type of state the implicit assumption is that some outcome, even if only partially realized, is achievable in a single application

instance. For example, it may be necessary to edit a file multiple times in a word processor. However, in any given session (which might constitute a case), some meaningful behavior may be documented and associated via event logging. A word processor does not (typically) maintain any state information regarding a given file. Thus, the unit of measure must be the observable behavior in a given session, which therefore defines the upper limit of a case boundary, such that the source is a program launch and the sink is program termination.

The event association to a case must also result in cases that are of a manageable size. Though the literature does not appear to address or define such a concept very well (though see complexity metrics in Benner-Wickner, Book, Bruckmann, & Gruhn, 2014), many of the noted "spaghetti" models (van der Aalst et al., 2004) were the result of cases with a large number of overly fine-grained events. Understanding user behavior is reduced when cases contain too many events. Models are not reality, and a degree of abstraction is required in order to derive knowledge from the data. In van der Aalst et al. (2004), for example, in the case study of health care, it was necessary to focus "on specific tasks and [abstract] from infrequent tasks" in order to successfully mine the process (p. 1140). In that case study, the amount of noise in the log reduced the effectiveness of the mining until pre-processing reduced the noise level. In general, the larger number of events associated with a case, the more likely it is the event log will contain noise. Some miners can deal with noise in the event log more effectively than the alpha-miner used in the health care case study. Nonetheless, it is worth considering defining cases so that the events reveal the essential behavior for the case. Returning to the word processor example, rather than using program launch and termination, in which one might find events associated with a number of documents, it might be better to define the case as the events associated with a single document in a given program execution. That is, the primary goal of a user is to produce a

document, and the primary artifact is a document, so rather than logging events across the entire application session, the case could be defined as a document in a given session.

A case must have a unique identifier associated with it. The case identifier must be unique in a given event log, rather than globally unique. Uniqueness in a log is required to associate events with a specific case. In a standard PAIS system, this identifier is driven by a particular process instance. In non-PAIS systems, it must be possible to identify or generate an identifier in some fashion. For example, in the health care case study, the patient number (e.g., a social security number) would not be sufficient, since a patient could visit multiple times. In the word document example, the name of the document is not necessarily unique, since many documents could have the same name. In the legacy application case study, the cases would need some generated unique identifier, though the article did not specify how such an identifier would be generated. Though it is not a strict requirement, there may be advantages if the identifier is stable across time as well. For example, the usual input into many of the Process Miner software packages is an XML file. If this XML file is generated from some other source (e.g., a relational database), cases could be uniquely numbered in a given file by sequentially numbering as the case is added to the XML file. However, there is no guarantee that a subsequently generated XML file would contain the same ordering. From the perspective of the process discovery algorithm for a particular mining run, it is irrelevant, but from a comparison and replay perspective, the inability to identify the same case over time could be problematic.

An additional consideration for the definition of case is whether the application in question contains sufficient functionality to support a case. It is likely there is a minimal complexity that an application must support before a case can be defined. Consider, for example, the *nix program “ls”. This program provides a listing of files. Though the program has a

number of “features” (extended attribute listing, recursive, color, etc.), it provides a single piece of functionality. From the perspective of this single application, the user only has a single goal, namely to display the files. Though it may be combined with other operations (e.g., piping the output to another application), the single supported outcome limits the usefulness of attempting to mine the application. There is an element of choice in how to achieve a given goal that must be present before a case may be defined. A case, therefore, is the collection of choices (behaviors) that are exhibited by the user. If a user has no choices in relation to a goal, the benefit of Process Mining is reduced. However, most interactive programs allow some degree of freedom in achieving goals. Furthermore, as noted elsewhere, highly controlled workflows exhibit reduced flexibility. The general trend is to allow users to optimize within some set of constraints (e.g., Bezerra & Wainer, 2008; Borrego & Barba, 2014). What is of interest, therefore, is the behavior of users in obtaining a goal, as well as uncovering these various goals. A case must defined in a way that allows for answering these questions, and where there is no question of user choice, there is no case to be defined.

A case, as shown above, has a number of requirements in order to be useful. In a PAIS, a case follows from the embedded workflow, events are associated with the case, a case has a relatively bounded lifespan and number of events, but is sufficiently complex to support choice. The bulk of the Process Mining literature starts from a PAIS assumption, and seeks to mine processes from these types of systems. However, there are a few articles investigating non-PAIS applications. In these applications, it requires effort to define a case. Many of these applications are data-driven, produce artifacts, and frequently these artifacts have a lifecycle associated with them. These applications, therefore, produce non-transient artifacts. Further, these applications play an important role within many organizations. Above, one has been encouraged to consider a

word processor, or an e-mail program. One might also consider a spreadsheet application that drives a tremendous number of decisions, but may not be integrated into any overarching PAIS. However, to understand the goals and the behavior associated with their achievement by using Process Mining techniques, one must define a containing "case" to which events may be associated. There is no single, complete approach to defining a case. Utilizing guidelines that a case must have an identifiable source and sink, a manageable number of associated events, and a focus on user behavior associated with data transformation (perhaps driven initially by use case analysis) rather than application behavior provides a starting point for the case boundaries.

Considerations of Activities

The second required element in an event log is the activity (or task, or more generally "event") itself. In order to successfully mine an event log, these activities must also have certain characteristics. The literature has shown slightly more concern with understanding the characteristics of an activity than it has with the definition of a case. There is, of course, a general bias towards an activity being a task defined in a workflow. Even in these PAIS applications, as noted above, ontological deficiencies have been observed, as not all activities required for mining are necessarily present in a workflow vocabulary. Consequently, PAIS systems may have issues with effective activity logging, or require additional processing in order to effectively delineate a case. Activities exist at some level of abstraction, and consideration of the granularity is important. An over-abundance of activities associated with an event can increase the noise in a log. Activities must have some identifier, and providing non-interpretable identifiers requires, at best, additional translation in order to interpret a mined process. At worst, a discovered process is unintelligible due to the identifiers not having any referent as experienced by the stakeholders. The identifiers should also be named such that a given name refers to one

activity only. While there could be, for example, two activities in an event log with the identifier "collect signature," unless these refer to the exact same activity, providing different identifiers is highly advisable. If two different activities share the same identifier, one must contend with the "duplicate task" issue.

Pérez-Castillo et al. (2011) noted the difficulty in specifying the granularity associated with activity logging in non-PAIS systems. In these systems, the general tendency is to utilize a method or procedure as a candidate for the locus of logging. However, source code procedures or methods have widely disparate granularity in their implemented functionality. If the granularity of the logged information is too fine, the event log will be unnecessarily bloated. Further, at this level the logging is tracing the application's behavior, and not necessarily the way in which the user is interacting with the application. Conversely, overly course grained logging will obscure the nuances in the process, leading to models that fail to reflect the potential diversity of approaches taken by users. The authors stated that finding the correct granularity is an important challenge that the literature needed to address, and ranked it in a similar fashion to defining the correct definition of a case (or "process scope") (p. 306).

Baier, Mendling, and Weske (2014) were also concerned with this granularity, or "level of abstraction" issue. For these authors, the issue was the potential incongruences between the events/activities that were recorded in a log and the business users' mental abstractions. For example, in their examined case study, a log might have a set of sequential activities identified as "Group, Classification, Detail, Detail, Group…" for a given case. However, the first four entries might map to a single "Incident Classification" step in a mental model. The second "Group" activity identifier (which also demonstrated a duplicate task issue), might indicate a "Functional Escalation" step. A mined process with an undue level of details, the authors suggested, inhibits

interpretation of the discovered process. The resultant process model would, at best, devolve into the "spaghetti" model, or at worst contain too much noise to be useful. In short, application level mining does not match the experience of the users based upon their interaction with the application. The authors therefore suggested the need to create a mapping between the event logging abstraction and the business model abstraction. The authors proposed a way to automate the mapping by relying upon external knowledge (i.e., documentation). However, when the goal is to discover a process that is not elsewhere documented, the suggestion of external knowledge is difficult to apply. The case study involved an ITIL application, and the application logged based upon application events, not necessarily user actions. However, the existence of the ITIL provided the external reference necessary to support the mapping. If one were considering event logging for a non-PAIS application, it would be preferable to log at an abstraction that more closely aligns with the user's behavior rather than the application's behavior. Otherwise, one must apply some sort of translation, mapping, or pre-processing to exclude events.

"One thing that much of the literature has failed to address, however, is the difficulty in simply identifying machine-level events as the high-level tasks they represent. That is, most research assumes that an accurate labeling of tasks is already obtained or is easily determined" (Buffett & Geng, 2010, p. 498). In contrast to the mapping approach of Baier et al. (2014), Buffett and Geng (2010) proposed utilizing clustering techniques based upon textual analysis. However, in order to apply the labeling, the authors assumed a pre-existing model to support the association of the machine-level events to the high-level tasks. To a certain extent, the question was also one of a mapping, and demonstrated the difficulty of associating tasks to a mental map when the level of abstraction is different between the two.

The basic difficulty stems from the desire to understand user behavior, but logging typically being application behavior. When van Genuchten et al. (2014) asserted that "software runs on machines that can log user behavior [and] these user behaviors are stored in a so-called *event log*" (p. 94; emphasis in original), the authors made an assumption about granularity. The authors examined events such as "import scan", "create CAD model", etc. In these cases, the events derived from the steps in high-level Use Cases rather than from application traces. These authors also suggested that "end-user functionalities" (p. 99) should be the proper focus for mining and comparison. Moreover, the authors suggested that event logging requires iterative refinement. Further, the analysis of the logs was applied immediately during development (by collecting information from testers and early adopters), as well as after deployment. The goal of Process Mining applied to applications should be to better understand the users, not the application.

In Rubin, Mitsyuk, et al. (2014), the authors also suggested focusing on user behavior as recorded from the interaction of the user with an application. A "user action" combined with some object (a widget or dialog) represented a user activity (p. 6). In this case, the logged events would be behavior as initiated through a user interface. Though this approach can place the granularity in line with user experiences, it also takes careful consideration of what objects constitute an appropriate event, and the authors did not provide further guidance. If every widget interaction is captured (e.g., every drop-down on every form), the granularity will decrease and the number of logged events will increase. Again, from a UX perspective, this logging is potentially useful, but from an overall user behavior perspective in pursuit of some goal it may be less so. Submitting a form, however, follows both a data approach (likely, some artifact is created, retrieved, or modified in response) as well as a user behavioral approach. In the article,

the authors provided sample events such as “make pre-reservation” or “get reservation” (p. 4). These logged events resulted from a form submission on the part of the user, and utilized an artifact. There was a translation applied from “T1-BA” to “make pre-reservation,” but the issues of mapping across levels of granularity were avoided by focusing on events corresponding to user behavior.

In summary, when the goal is to understand user behavior, logging activities that have a granularity congruent with the way a user interacts with an application will provide a more direct route to mining the logs. Users exhibit a variety of approaches to achieve goals, despite original assumptions about the processes they will use. Capturing the steps users take as they interact with an application to manipulate data or artifacts allows for discovering this variety. When the event log is generated following the application logic rather than the users’ behavior, the granularity of the logged events will not align with the mental maps of the broader user community, though it may match developer experiences. In such cases, it will be necessary to map application-logged events into behavioral events. While it is not impossible to generate such a mapping, it may be a manual process unless sufficient external documentation exists to assist with automating the mapping. Consequently, if one is attempting to use Process Mining as a part of application development, especially if the application is a non-PAIS one, careful consideration of the granularity will allow for mining without the extra mapping effort. In Rubin, Mitsyuk, et al. (2014), the authors were able to exhibit discovered process maps to various stakeholders multiple times per day. If the granularity were not closely aligned with the mental maps of the stakeholders then explaining the discovered processes would have been much more difficult or required additional effort to pre-process the event logs. By starting with a close alignment, the

case study illuminated the benefits that Process Mining applied to software development can deliver to developers, testers, and other stakeholders.

As noted above, Process Mining has several unresolved challenges in its mining algorithms Two of these challenges, “duplicate tasks” and “noise” can be partially or completed alleviated by what is emitted to the event log. To a certain extent, both of these issues are exacerbated by logging at an application trace level rather than a user behavioral level. Process Mining is made easier when the logs do not need extensive pre-processing.

The “duplicate task” issue arises when a given identifier binds to more than one point in the process. In Baier et al. (2014), the application event “Group” appeared in multiple places, even though it indicated different steps in the data flow. It is difficult for the mining algorithms to detect these conceptual collisions, since the appearance of the task label could represent a loop or a new step in the discovered process. One is better served by ensuring that activity labels are unique. Further, by logging user behavior rather than application behavior, one has a better chance of avoiding the duplicate labeling. As another example, in Rubin, Mitsyuk, et al. (2014) the log had entries such as “window/load.” It is probable that with additional data associated with the event one could derive which window was opened. This particular activity label itself, however, is ambiguous. This logged event is more akin to an application log than a behavioral log (where the latter would have been more akin to “search reservation”). There is thus a potential relationship between the concepts of “granularity” and “duplicate tasks” insofar as application-level logging is more likely to generate duplicate task entries. However, even in more traditional workflow systems the labels associated with a particular step are not necessarily unique. One should consider expending effort to ensure steps are uniquely named (e.g., not “collect signature” but “approve invoice” and “approve payment”), as this will enable more

efficient application of Process Mining algorithms. Though often not stated, the inverse problem of using a different identifier for the same activity should also be avoided.

Noise is another issue that can be related to granularity of the activity or the definition of a case. It can also be the result of missing log entries, or the presence of infrequent behaviors. The questions of granularity have been discussed above. The problem of missing entries is difficult to address. For example, assume a log contained activities for four cases of {A,B,C,D; A,B,C,D; A,B,D; A,B,C,D}. Does the third case indicate a potential (though perhaps infrequent) routing from B to D, or a missing log entry? Depending upon the particular question, the third case could be an example of fraud (e.g., Accorsi & Stocker, 2012; Accorsi et al., 2013; Jans, Lybaert, & Vanhoof, 2010), if, for example, step C were some type of oversight preceding a funds transfer in D. One should undertake efforts to ensure the completeness, integrity, and accuracy of logged events wherever possible. Logging via an external location if one is instrumenting a non-PAIS application, for example, can assist with the integrity question. However, if the external service is unavailable for some reason the completeness of the log could be an issue.

Human readable activity identifiers are also useful whenever possible. “Task A” is not an immediately understandable identifier, whereas “make pre-reservation” is potentially comprehensible. Many Process Mining algorithms will use the activity identifiers in the resultant process model diagram(s). If the identifiers are not immediately intelligible, additional effort to interpret the discovered model will be necessary. Conversely, one may apply “translations” to the identifiers prior to running a particular miner. However, one must invest effort in order to construct, execute, and maintain such translators. Therefore, wherever possible, human interpretable identifiers are preferable.

Successful mining relies upon activities in an event log that have a correspondence to the mental model of an application's process. The process may be explicit, as is the case in many PAIS applications, or it may be implicit. The actual processes in-use, as discovered through Process Mining, may not align completely with *a priori* expectations about the process even in PAIS applications. In addition, with the rise in flexible resolutions to issues, many PAIS systems are moving away from strictly normative approaches to prescriptive approaches. In these latter instances, the process to be discovered is more fluid but more representative of actual behavior than overly constrained workflow processes.

In addition to carefully defining the granularity of the activities present in an event log, the activities must also "be normalized on a comparable level of abstraction" (Benner-Wickner et al., 2014, p. 109). This requirement argues against using method or procedure tracing as a suitable level for event logging. Moreover, since the goal is to discover processes reflective of user behavior, focusing on application behavior inevitably leads to mapping considerations. The object-activity model of Rubin, Mitsyuk, et al. (2014), combined with considerations of artifact transformations, can assist in leveling the granularity of logged events. Ensuring that activities have unique, human interpretable identifiers can also assist with issues of abstraction. Differing activities with the same label inherently confuse the process model, increasing the difficulty in mining or interpreting the discovered model. As a result, ensuring that activities have a well-defined identifier is an important consideration.

Instrumenting vs. Interception

When considering event logging with non-PAIS applications, the means by which events are gathered should be considered. In general, applications may be divided into three categories: stand-alone clients, clients that communicate with a server, and web-applications (see Rubin,

Mitsyuk, et al., 2014). In order to emit the events to a log, there are two basic approaches. In the first, a developer may decide to "instrument" the application. Instrumenting an application adds specific commands in order to log the event, or it implements an "observer pattern" on user activities. The second approach is to "intercept" the communication between the client and the server. In this approach, some monitoring or pre-processing of the protocol between the client and the server is added and the event log is emitted by this interceptor.

In the case of completely stand-alone applications, interception of a communication protocol is not an option. These applications must have logic inserted into the application in order to capture activities to be emitted to the event log. Two main difficulties must be addressed when instrumenting an application. The first is "what" should be captured. The second is "how" to instrument the application. These two questions are not completely orthogonal. If one answers the "how" question as essentially procedure level tracing, the "what" turns out to be application level event logging, rather than user behavior logging. Observing only menu selections, on the other hand, precludes understanding system-generated events that result from an initial user selection. This approach may distort the discovered processes.

As Pérez-Castillo et al. (2011) noted, "Non-process-aware systems entail some challenges for obtaining meaningful event logs… [since] it is not clear which events should be recorded in the event log" (p. 304). These authors were concerned with recovering business processes from legacy systems. Their primary argument was that static analysis of legacy systems discarded knowledge "since some specific knowledge is only discovered at runtime" (p. 305). Part of the knowledge that is discovered through dynamic analysis is the actual way in which users interact with the application, as opposed to the theoretical approach that might be discoverable solely in the source code. In short, dynamic analysis captures user behavior that in

turn reveals in-use processes. However, to capture dynamic analysis, the code must be instrumented in order to generate event logs consisting of activities captured at runtime. The authors suggested that the code be instrumented "in a non-invasive way (i.e. small changes without affecting the behaviour and performance of the system) to enable the registration of events during system execution" (p. 307). The authors relied upon systems analysts to identify the correct events to log. In addition, the proposed model for activity delineation ultimately gave definition to the requisite case to which the activities would be related. This specific notion of a case was not as well articulated in other discussions of application instrumentation. The benefit of the relying on experts for activity definition approach is that it moves from application tracing (which is the essence of, e.g., Zou & Hung, 2006) to behavioral tracing. The activities are defined as the value-added steps undertaken by the user as expressed during the dynamic execution of the software. These authors indicated the most likely approach for capturing events in stand-alone applications is through instrumenting the code, though they did not specify exactly "how" the non-invasive approach should be implemented.

In a different approach, Snoeck, Poelmans, and Dedene (2000) and Michiels, Snoeck, Lemahieu, Goethals, and Dedene (2003) considered object-oriented thick client applications. These authors suggested that frequently in software design the events that drive the application are subordinated to objects. Further, they argued that almost all use cases and UML diagrams treat the interaction of the application as a sequence of activities, rather than patterns of events. For these authors, "events are atomic units of action that represent things that happen in the world" (p. 458), whereas activities are the messages between objects. The critical aspect, these authors suggested, is to understand that events form "a fundamental part of the structure of experience" (Michiels et al., 2003, p. 3). Though these authors were not directly considering

Process Mining event capture, the focus on events as structuring the experience provides another possible means for understanding how to instrument thick clients. By capturing said events rather than attempting to find the correct point for instrumenting the business logic invoked by the events, following user behavior is easier. In addition, proper event handlers are likely easier to "observe", supporting potentially less invasive instrumentation. It is possible that not all experiential events are appropriate for capture, of course, but it does provide a potential starting point for consideration, as these authors envisioned events as the basic glue between a more malleable GUI front-end and the more stable logic back-end. This architectural division is absent in Pérez-Castillo et al. (2011), but then the particular experiences that informed that article may have lacked the clearer architectural delineation that Snoeck et al. (2000) or Michiels et al. (2003) suggested was necessary in order to support maintainable applications.

For thick-client applications, therefore, two approaches seem to be evident. If the application has a sufficiently rich internal "event" mechanism, mostly likely because of a particular architectural design and emphasis on events as first-order participants in the application logic, then applying an "observer" pattern to capture these application events may provide the input for the mineable event log. Though not discussed in such terms, the experiential events are reminiscent of the object-activity model of Rubin, Mitsyuk, et al. (2014). On the other hand, since many applications likely fail to have such an architectural design, one is reduced to ascertaining the correct location in the code that corresponds to a valuable activity and inserting calls that will emit the requisite data to an event log.

The other two categories (client-server and web-applications) have similar considerations in terms of event logging. If the communication protocol contains "data about the user behavior" (Rubin, Mitsyuk, et al., 2014, p. 6), and the data may be related to some defined case, then

intercepting the protocol on the server and logging it is an option. However, the presence of a communication protocol does not necessarily indicate the best approach is “interception”. For example, the communication protocol may not reveal the actor and the selected event, but rather consist of lower-level communication (e.g., requests for a particular set of records). That is, the communication protocol must meet strict requirements in order to reveal exhibited user behavior rather than simply application logic. The tour agency case study logged events based upon the intercepted protocol. This logging was possible as the protocol corresponded to user-driven events. Elsewhere Rubin, Mitsyuk, et al. (2014) suggested that implementing logging on the client side even in web applications would result in a better correspondence between user activity and the logged events. The authors suggested considering Javascript-based logging, much akin to the way Google Analytics is added, for web-pages. The concept of the object-action model also drives one towards instrumentation rather than interception, since one would wish to log the actions applied to objects rather than the resultant stream.

The general approach, therefore, seems to be towards instrumenting at the client-side (either a client or in the web-application). Though protocol interception may be possible, it is only when the protocol is sufficiently well defined and has sufficient correspondence to user behavior that it is truly useful. Since event logs assume a complete trace of the activities, the protocol must be robust enough to cover all possible actions. In other words, a protocol that mixes user behavioral requests with lower level requests is not suitable since it would mix granularities and likely reflect only a subset of user behavior. Furthermore, there may be useful behavioral characteristics that should be captured but are not transmitted to a server. Such activities are likely to be present in AJAX applications, for example, where some user functionality is handled solely in the browser and not communicated to the back-end servers. As

interactiveness is added to web-applications via code executing in the browsers, the ability to capture user behavior is naturally driven towards logging originating from this code.

Instrumenting, however, has a higher cost in terms of code development and maintenance. When new functionality is added to the software, one must consider where to add instrumentation to capture the expressed behavior. Event driven software can potentially reduce the need for additional instrumentation, relying instead upon the ability to observe the events. However, Michiels et al. (2003) suggested most software is not designed from a perspective where events are first-class contributors to the software architectural model. The need for considering event capture when software is created or modified suggests another "-ility" in the software quality matrix suggested by Boehm (1978) and others. In addition to such things as "readability" or "maintainability", one must also now consider "measurability". This quality attribute is more than just user or feature counts, but rather the full range of behavioral expressiveness. In order to understand how software is actually used, one must provide sufficiently refined logging that can be subjected to analysis. Further, the goal is not simply a listing of "page" (e.g., Google Analytics) or "features", but the pathways through the application. What users are attempting to achieve (their "goals") is as important a question as how (the process) they are achieving the resultant outcomes.

The unfortunate conclusion, however, is that the literature only provides rough guidance regarding how to collect the event logs. The bulk of the literature has traditionally assumed a PAIS application with ready-made event logs. When the literature moved towards the investigation of user behavior rather than application event re-discovery (e.g., Günther & van der Aalst, 2007), the focus was on improving the mining algorithms, and still assumed the presence of logs produced from the execution history. Only a few articles have directly examined the

question of how to add event logging suitable for Process Mining. To a certain extent, the wide variety of potential application architectures, languages, and deployment approaches makes generalization difficult. The one potential thread is that instrumenting by inserting code to emit a log entry is likely the best approach, despite the need for careful consideration of where the event is actually invoked.

Log Location and Persistence

Where the event log should be stored is another important consideration. In general, the application of Process Mining to process aware information systems (PAIS) assumes the presence of not only a log, but a central log. Often the assumption is the event data is in a single database table (Fahland & van der Aalst, 2013), though more nuanced approaches recognized that, "Events may be stored in database tables, message logs, mail archives, transaction logs, and other data sources" (van der Aalst, 2012c, p. 557). When events are scattered among various logs, however, they must be merged before they can be useful. Merging entails a series of issues, from assuring events in different logs are associated with relevant cases to issues with non-homogeneity (Diamantini, Potena, & Storti, 2012; Günther & van der Aalst, 2007) and granularity differences in the recorded events. Non-PAIS applications, however, must explicitly define not only how to emit an event log, but where the event log should be stored. Dispersed logs will require collection and merging, if not additional processing effort. A centralized log avoids the collection and merging issues (there may still be pre-processing involved), but requires connectivity as well as an awareness of what data is being transmitted and in what way. In addition, the raw event logs are of little interest to a local user. It is the revelation of the process itself via Process Mining that provides the value.

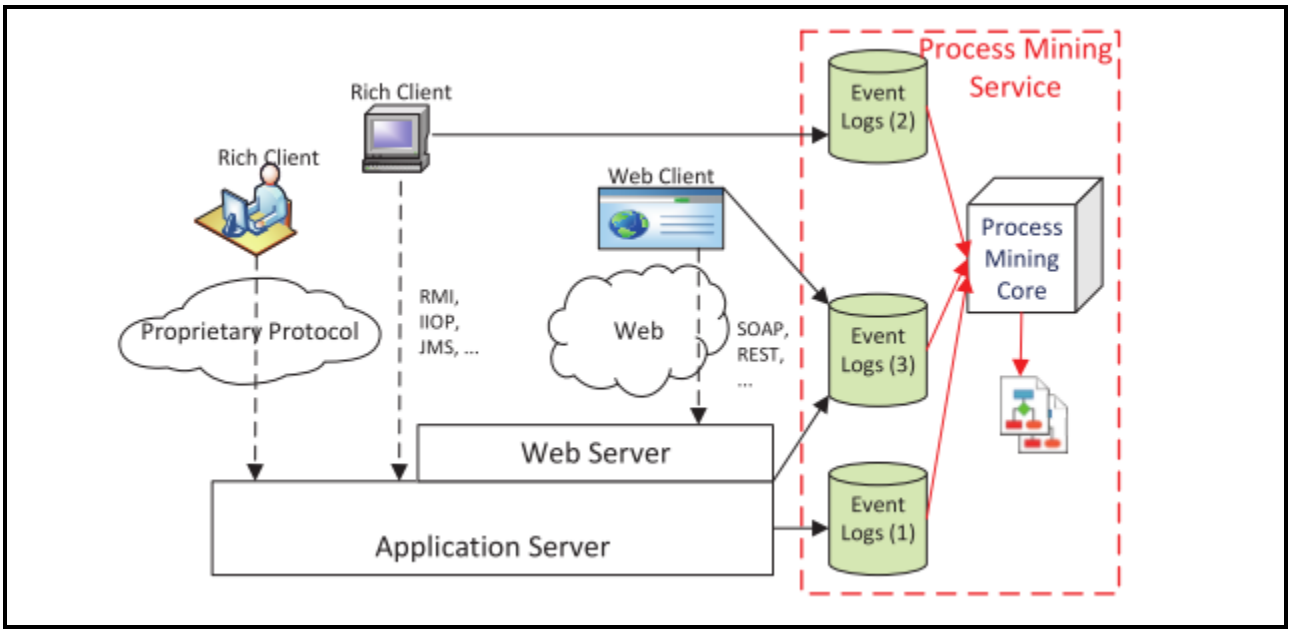

Figure 4. Event Logging to Remote Service. From (Rubin, Mitsyuk, et al., 2014, p. 7)

To a certain extent, Rubin, Mitsyuk, et al. (2014) presented the typical approach for most applications types. Figure 4 shows their approach for a remote event logging service to which different applications may post via a variety of protocols. The authors envisioned Process Mining as a separate service where the log "is stored separately and separated from the core of the software system" (p. 6). The general approach focused on a centralized log (or potentially set of logs) feeding a Process Mining core. The particular diagram depicted a single application with differing front-ends. One could also interpret the diagram as suggesting the means by which centralized event logging could be achieved by the various types of clients. Regardless, separating the event logging into a discrete, centralized service provides the easiest approach for subsequent Process Mining. That is, when developing event logging, posting an event to a centralized service that maintains the logs is the general recommended approach in the literature. Of course, the primary assumption in the diagram is that the application has the ability to communicate with a remote server. The issues of dealing with non-networked, stand-alone clients are not examined herein, nor in the literature in general.

When event logs are not centralized and generated in a uniform manner, a merged event log must be generated. Claes and Poels (2014) noted that merging can be accomplished at the raw data level (merging databases and/or files), structured data level (merging event logs), or model level (merging process models) (p. 7291). Each of these approaches has different issues, such as the differences in abstraction (or granularity), or the need to understand the meaning and relationship between different database records. Addressing these difficulties is not trivial. For example, issues of granularity were discussed above, with the conclusion that it is better to avoid the issue than attempt to apply mappings to rectify the situation. Claes and Poels (2014) attempted to address issues of merging at the structured data level, but despite the application of rules, the final merge required user interaction. One of the basic issues the user must address is aligning the definition of a case between the disparate logs. In van der Aalst (2011a), the ability to define the case was assumed, but the need to merge data from a variety of data sources was directly noted. In this instance, the preferred approach was to load disparate data sources into a single data warehouse, and then to extract XML-based event logs for a given process from this location.

In general, there is a tendency in the literature to assume that the event logs are stored in a relational database. The general language of the literature (assuming as it does a PAIS) discussed ERP systems (such as SAP) backed by relational databases. Rubin, Mitsyuk, et al. (2014) is an exception in making a passing reference to NoSQL databases, but did not discuss any implications of the specific storage approaches. Bui, Hadzic, and Hecker (2012) assumed the availability of XML event logs, and only made a short reference to the potential need to “extract and transform” the data (p. 111). Kwanghoon (2009) also presumed the presence of (distributed) XML files containing the event data. Constructing XML files from what is essentially an “event

stream" (Appel et al., 2014) would appear to be a difficult task, since the events might arrive in any order, but the XML files require all events for a specific case to be children of the parent case entity. Consequently, it is most likely preferable to first store the incoming events into some type of database that supports set operations, and then emit the XML files for processing in a separate step. Some Process Mining applications, such as ProM, support reading delineated files, though the general standard for Process Mining is the XES file format based upon XML (see Günther & Verbeek, 2014; van der Aalst et al., 2012). In addition, event logs require a high degree of completeness. One must consider the implications of "eventually consistent" databases versus ACID-compliant databases (e.g., Vogels, 2009) when implementing the storage strategy. In van der Aalst et al. (2012), the authors noted that events need not be stored in dedicated log files, but that the quality of the log files was critically important. Event logs, the authors argued, must not be a by-product of "debugging or profiling", but rather "first-class citizens." While how the logs should be stored, to a certain extent, is less important that ensuring they are of a high quality, one cannot escape the relationship between the storage strategy and the resultant quality, especially if logs become distributed and in need of merging.

When considering instrumenting a single, non-PAIS application, therefore, logging to a separate "Process Mining Service" located on a remote machine seems to be the preferred approach. This remote service should accept a posting of an event, and add it to a database. As discussed above, the posted event must consist of a minimum of two fields: an activity identifier and the associated case identifier. Other data may also be present if appropriate. If the case can be inferred via the activities (i.e., there is a definitive start and stop point for an identifiable session), it may be possible to omit the case identifier from the event posting, and generate it later. However, as noted, there are potential issues with case number in such an approach. A

remote service also increases the potential integrity of the data. Such considerations are extremely important if one wishes to apply Process Mining to anomalous event detection, such as fraud. Full consideration, however, of what would constitute a secure Process Mining Service is beyond the scope of this investigation.

Whether the timestamp should be passed as a field to the server or added automatically by the server is an interesting question. Ordering of the activities in a case is a critical requirement, as activities that are incorrectly ordered will result in noise or incorrectly discovered processes. Passing the timestamp from dispersed clients raises the possibility of such noise, since different clients may have different clock settings. Generally, of course, for a single application running on a single machine, the clock would be constant for that given machine. However, not all databases support fine-grained time storage, and thus two events generated only a few microseconds apart could conceivably have the same resolution in the database. Conversely, allowing the Process Mining Service to add the timestamp could introduce the very noise one is seeking to avoid, since there is no guarantee regarding the order in which nearly simultaneous events would be added to the database. That is, a multi-threaded service processing multiple inbound posted events is non-deterministic with regards to the order in which the events are ultimately processed (though see Chu-Carroll, 2016 for a discussion of Lamport Timestamps to address order). Consequently, a single client posting two events A and B to a service cannot guarantee the order in which the events will be processed by the service. The existing Process Mining literature does not provide specific guidance on this point. Since a timestamp may not be sufficient to ensuring ordering, alternative approaches may need to be considered. For example, an atomically increasing integer for a given case could be specified in the event posting. Conversely, a sufficiently refined timestamp generated on the client, backed by the appropriate

storage in the service, could suffice. Unfortunately, outside of the general desire to have a timestamp to support potential ordering as well as potential costing analysis (such as bottlenecks), the specific approach depends upon the implementation of the Process Mining Service.

One cannot undertake Process Mining without a well-formed event log. In general, the literature has assumed the presence of this log, without offering specific guidance on where to find the log. Rubin, Mitsyuk, et al. (2014) suggested an independent Process Mining Service as the appropriate receptor and storage location for events. The general assumption is that the events received by the Process Mining Service would be entered into a database, and then later extracted into relevant XML files. In van der Aalst (2011a), the larger enterprise approach acknowledged a variety of potential data sources for a given event stream, and recommended consolidation into a data warehouse. Centralizing the collection is, however, a common theme in the literature, to support easier extraction. Certainly to support Process Mining during an Agile iteration, a centralized repository provides benefits in that potentially dispersed logs do not need to be gathered, and potentially merged. From a defined central repository, batch processing to generate the XML event logs is straightforward, leaving only the execution of the miner itself as a potentially interactive step for a member of the development team.

Privacy Concerns

In van der Aalst et al. (2005), one finds the warning that if an event can be traced to a person, the person must be aware of the logging. Fayyad et al. (1996) also noted that when one collects any personal data, it is necessary to consider the privacy and legal issues. As one would expect, given the myriad of legal jurisdictions, the Process Mining literature provides little specific guidance. In the case studies dealing with medical records (e.g., Kaymak, Mans, van de

Steeg, & Dierks, 2012; Maruster et al., 2001; Montani et al., 2014; Perimal-Lewis et al., 2014; Sabhnani, Neill, & Moore, 2005; van Genuchten et al., 2014), the patient data was rendered anonymous. Beyond these considerations, however, one only has the stipulation in General Principle One of the Process Mining Manifesto that "the event data should be safe in the sense that privacy and security concerns are addressed when recording the events" (van der Aalst, 2011b, p. 7).

In general, within an organization, a sufficiently comprehensive acceptable use policy (AUP) should be sufficient (e.g., Siau, Nah, & Teng, 2002; Stanton, Stam, Mastrangelo, & Jolton, 2005), though of course competent legal advice should always be sought. The AUP should detail the ability of the organization to monitor and record activities. The situation is slightly more complex if the organization crosses international boundaries, since different legal jurisdictions may attempt to impose differences in what may be collected or the length of time data may be retained.

However, if one wishes to collect data from a broader audience, detailing what will be collected, how it will be used, and how it will be retained becomes critical. For example, the Eclipse Usage Data Collector (EUDC) had such a disclosure (Beaton, 2011; Khodabandelou et al., 2014). Many web applications, web sites, and operating systems collect usage data (Bright, 2015) without explicitly disclosing the data collection. There are potentials for understanding user behavior that arise from looking at a wider set of users. Conversely, one must also be aware of the amount of data that may be generated, and have a plan to actively utilize the data. The EUDC, for example, failed to capitalize upon the collected data, and eventually terminated the program. Questions of opt-in versus opt-out for contributing must be addressed, and different jurisdictions may have different requirements for the default approach. The collection of

personally identifiable information during events may also create additional requirements for the storage and handling of the data.

Unfortunately, since data privacy inevitably becomes intertwined with legal requirements, there is little guidance provided by the literature. Privacy is an important consideration, but cannot be determined outside of the specific application and its collection approach. Ignoring privacy concerns is inappropriate, however. When considering event log creation, it is imperative that privacy and data security be considered. Beyond that imperative, however, specific legal guidance is likely required.

Conclusion

An application need meet only three criteria when generating an event log. An application must be able to, "Record events such that (i) each event refers to an activity (i.e., a well-defined step in the process), (ii) each event refers to a case (i.e., a process instance), and (iii) events are totally ordered" (van Dongen & van der Aalst, 2005, pp. 1-2). Additional domain specific data may be present, if appropriate. Defining a case and defining an activity require definitional effort. The Process Mining literature has tended to assume a process aware information system (PAIS) as the basis of the application. Such applications contain an explicit workflow system. Given the explicit workflow, a case corresponds to a particular workflow instantiation, and an activity is a step in the workflow. However, there are a variety of non-PAIS applications that are amenable to Process Mining. In these instances, a "case" and an "activity" must be carefully considered and defined in order for the event log to contain useful information.

The potential of Process Mining to discern user behavior, and then to apply this knowledge to refine programs, is very intriguing. In the articles that directly examined the application of Process Mining to non-PAIS applications, the discourse focused on "user

behavior" rather than explicit processes. Activities, therefore, derive from the actions invoked by a user, rather than what an application might log as its traces through its internal logic. A case must collect a reasonable set of related activities undertaken by a user within some bounds. That is, the collection of user activities must have an identifiable start and end point.

One critique of the emphasis on user behavior is that it appears to rely upon interactive applications; events are the result of direct user interaction. There may be a number of batch-oriented applications that would also benefit from an application of Process Mining. Discovering user intent in batch applications, however, may be more difficult, since other than launching the application it is not immediately evident a user exhibits intent during the run. The questions of case definition (a single batch run, all runs related to some given set of data, etc.), and activity definition (granularity, labeling, etc.) are still important. However, the application of Process Mining to support application development is not investigated for such non-interactive applications in this thesis.

Chapter 4: Example Application of Process Mining to a Software Application

This section provides an example of Process Mining with a non-process aware information system (PAIS) application. It shows how event logging can be applied to discover user behavior associated with application usage. It is not a case study in the classic sense. First, the application examined in this section (the Sandia Analysis Workbench) was both originated and instrumented for feature logging without reference to the broader Process Mining literature. Second, the initial application of Process Mining reported in this section pre-dates the origin of this thesis. Third, it is not "an intensive study of a single unit for the purpose of understanding a larger class of (similar) units" (Gerring, 2004, p. 342). Rather, it provides some "life" to the above discussion by providing visual displays and general discussion of one instance of Process Mining on an application with the goal of supporting software quality improvement in an Agile development environment. All of the information relating to the Sandia Analysis Workbench and the discovered processes is taken from Olson et al. (2015).

The Sandia Analysis Workbench (SAW) is a thick-client application built on top of the Eclipse framework. It was developed at Sandia National Laboratories to support the computational simulation analysis community. Its vision is to provide an integrated working environment to enable analysts to more effectively deal with the artifacts and tasks necessary to perform computational simulation. Analysts face a complex environment requiring them to interact with and transferring data between a variety of individual applications across several different computer platforms. The general "process" is to start with an design engineering based solid model (usually produced in a CAD system), apply a number of transformations to that solid model, add specific information relevant to the particular simulation, submit the simulation to run on a high-performance compute cluster (HPC), obtain specific measures from the simulation (possibly including visualized results), and produce a final report. SAW brings together a

number of useful features, such as data management, secure team-based artifact sharing, model assembly, job submission and monitoring on the HPCs, distributed file management and remote visualization.

SAW (under a different original name) was originated in 2004. It has been developed by a core team of four developers utilizing an SCRUM based Agile development methodology. It supports a highly technical community of computational simulation analysts working in a variety of simulation physics, such as thermal, structural, and radiation emissivity.

In 2014, the developers instrumented SAW to produce logging for documenting feature utilization. In this sense, the original purpose to answer a question akin to the Alice story above: namely, what features are being utilized to what degree by the user community? The hope was to discover patterns of utilization to support software development efforts, and to foster conversations with the stakeholders pertaining to development priorities and opportunities. The collected data has not, to date, been used to "quantify" upgrade benefits (per van Genuchten et al., 2014), but some changes in user behavior over time have been documented as evidence of the effectiveness of changes to, or additions of, specific features.

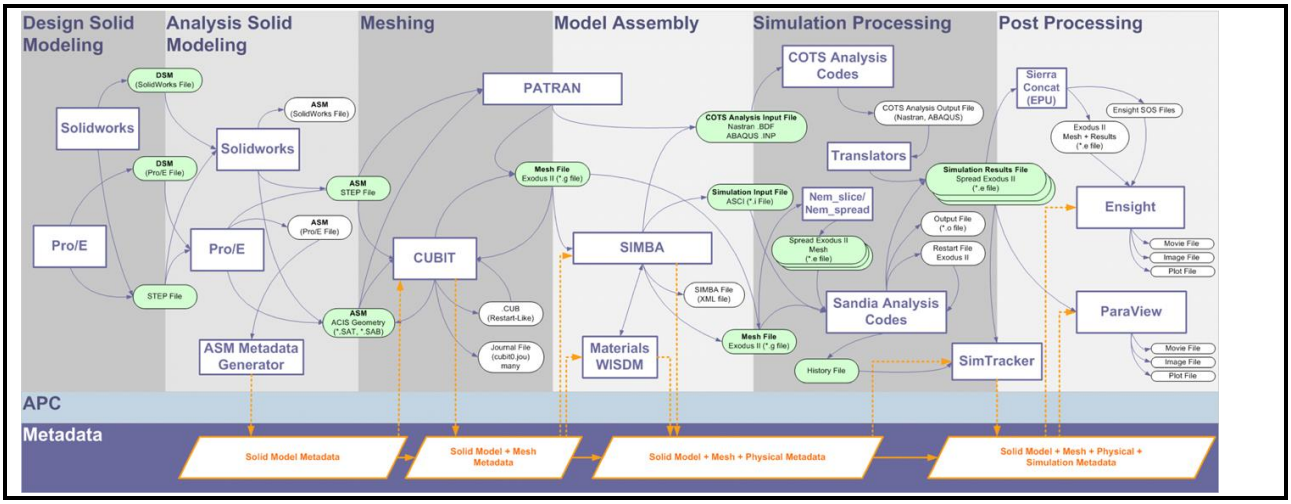

Figure 5. Sandia Analysis Workbench Problem Domain

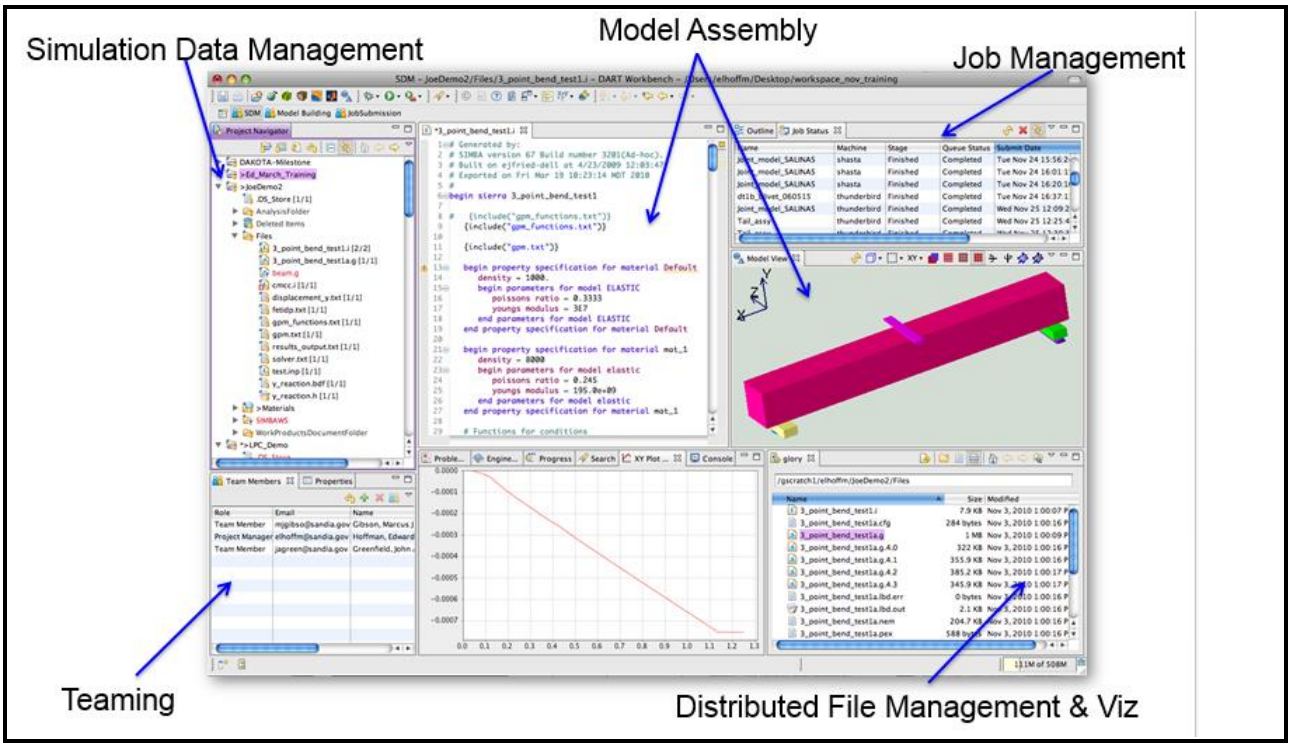

Figure 6. Sandia Analysis Workbench Application.

When the SAW development team first decided to explore Process Mining, it needed to confront several conceptual issues. First, it needed to understand a "case" from the perspective of Process Mining. When the application was first instrumented, associating events with a particular instance was not part of the consideration. Second, it needed to understand whether the logged activities were compatible with the general requirements of Process Mining. The logged features corresponded to user activities in the interactive application, but granularity questions were not considered during the initial log development. Finally, it was not clear that Process Mining would necessarily discover a model that would provide insights into the behavior of the user community.

A Process Mining case for SAW is defined as a single instance of the application from program launch to termination. The original goal of SAW was to support the analysts from design through the final report. From this perspective, it might seem that a more workflow like definition would be appropriate, tracking user behavior across program launches. However, there

are several factors that mitigate against such a higher level, PAIS-like approach. First, despite the concept of a "project" being integral to the application, and the "project" supporting lifecycle states, the analysts have not utilized the functionality to move a project through various stages. In addition, some analysis have created only a single "project" artifact, and co-mingled the various investigations in folders under this one project object. Thus, while the original use-cases pre-supposed a particular behavior on the part of the user community, the reality has been quite different. This behavioral deviation was observed early in the development of SAW, without the need to apply Process Mining. Indeed, the analysis community has focused more on the file versions and the sharing of files between the team members than any larger artifact lifecycle concerns. Second, the application is deployed on several networks that are disconnected from one another. A single "project" may be present on multiple networks, but there is no way to relate activity across the networks. Third, and related to the first, there is a distinct lack of a terminating event for a project on the part of the analysts. Finally, there was a desire to understand user behavior within the application, rather than user behavior in relation to a workflow process. Thus, the focus of investigation was the implicit processes of users in the application rather than a larger, more formal workflow. This focus is more akin to software application focus of Rubin, Mitsyuk, et al. (2014) or van Genuchten et al. (2014) than the process focused found in the case studies of ticket paying or property tax appeals found in van der Aalst (2011a; 2003; 2004).

The case definition, however, is not perfect. In the event log, the application launch and application termination are logged. However, the case lacks a unique identifier. Consequently, when the data are prepared for Process Mining, activities are assumed to be related to a case because they are sequentially logged for a user between these two events. Since there are

instances where an application termination event is absent (e.g., the application crashes), there are invalid cases present in the log (defined as two application launch events without an intervening termination event). In addition, there is nothing that prohibits users from launching multiple instances of the application at the same time (especially on different operating systems). The lack of a specific and unique case number allows for the potential interleaving of events from disassociated application instances. Consequently, the event logs may contain a type of "noise" in the form of extraneous events within a case.

An activity is defined as functionality invoked directly by the user. For example, "Open Project", "Edit File", or "Submit Job" are activities that are logged. The activities are uniquely named across the application, thus the issues associated with "duplicate tasks" are avoided. The granularity is not exactly the same across all events. For example, "Submit Job" requires file transfers to the HPCs, remote program invocations, and associated monitoring. While these lower-level activities provide a benefit to the user, they are part of a higher-level user activity that more closely aligns with what the user is accomplishing. Thus, while the application has certain "features" (such as file movement), this application level tracing is not logged. This level of logging avoids the need to map between levels of granularity. In addition, the activity names are sufficiently human readable as to avoid the need for translation of the activity identifiers.

As a thick-client, the application is instrumented to support the event logging. The events are logged via a separate service, and stored via a RESTful protocol invocation on a remote server. Developers must decide where in the code to insert an event logging posting. It is not, therefore, truly an "observer pattern" (Gamma, Helm, Johnson, & Vlissides, 1995) on the application as suggested as an appropriate approach by Rubin, Mitsyuk, et al. (2014). While the underlying Eclipse framework supports such a pattern, and in fact that approach was the basis of

the User Data Collector (see Beaton, 2011) that provided the input to Khodabandelou et al. (2014), the current implementation does not utilize such an approach. The advantage is that the "feature" that is being measured can be defined at precise locations, with control over what is added to the event log. The disadvantage is the need to specifically instrument the code, and adding new functionality requires discussions on the appropriate place to insert the event logging call.

In 2015, the development team became interested in applying Process Mining techniques to the event log to see if behavioral processes could be discovered. The hope was that Process Mining would reveal usage patterns that would be helpful in generating discussions with stakeholders. The decision to apply Process Mining was part of a larger effort to more formally understand the general workflow processes in-use by the user community. One driver of this effort was to increase the automatic documentation and (potential) re-execution of these processes. It is accepted that different analysts and simulation analyses will follow different pathways to completion. However, the documentation of these processes was frequently lacking, and further they could not be repeated by a different analyst. In essence, the verification and validation of a computational simulation came to include not just the utilized software and produced artifacts, but also the process by which the inputs to the simulation software were developed.

In order to mine the collected event logs, the events stored in the MySQL database, the events needed to be extracted and converted to the XES standard (Günther & Verbeek, 2014). A Java program was written to convert entries from the database to a well-formed XES document. This program had to account for the issues with the aforementioned case definition. In the following discussion, 633,510 initial MySQL records were processed into 3491 cases with

141,583 events. As the collected information contained user names, these cases came from 160 originators. The events were collected from a single network only, and it is possible that the mined behavior is different on the various networks. These questions were not investigated in this initial work.

After the XES document was generated, it was provided as an input to ProM, version 6.4. ProM is an Open Source miner supported by various academic researchers (see van der Aalst et al., 2012). After collecting the basic statistics concerning the input file, the first question was whether any basic process flow was evident. A cluster analysis was executed against the data. The results indicated a potential "spaghetti" process, but there was a general overall flow and identifiable clusters.

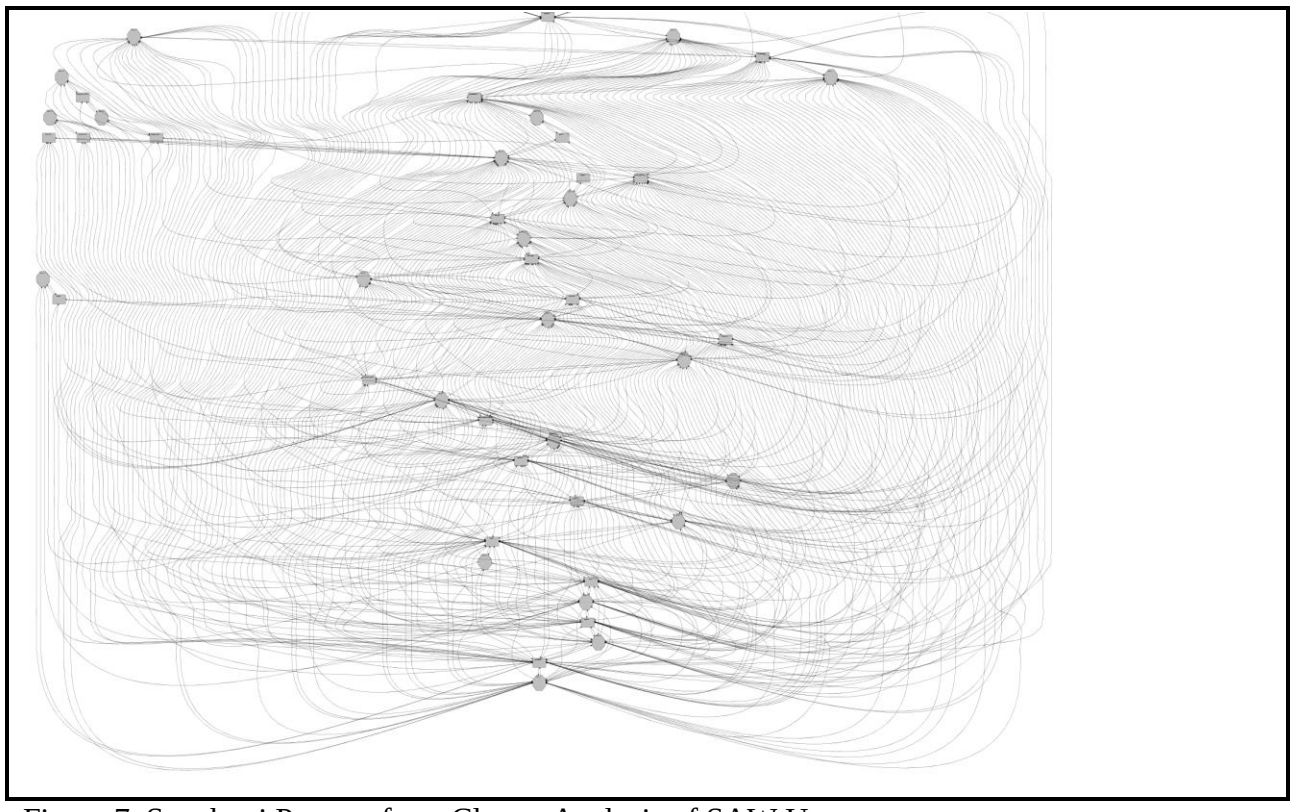

Figure 7. Spaghetti Process from Cluster Analysis of SAW Usage

The team then explored a standard Petri Net generation. The alpha algorithm approach failed to converge into a coherent Petri Net. As noted above, there are potential issues with the

alpha algorithm when confronted with loops and noise. Using an inductive Petri Net miner (Leemans, Fahland, & van der Aalst, 2014), a general Petri Net was discovered. The fact that a generally recognizable process flow from the event log could be discovered demonstrated the basic value of applying Process Mining. The number of loops and "hidden" process steps shown in the inductively mined Petri Net suggested why the alpha algorithm approach failed.

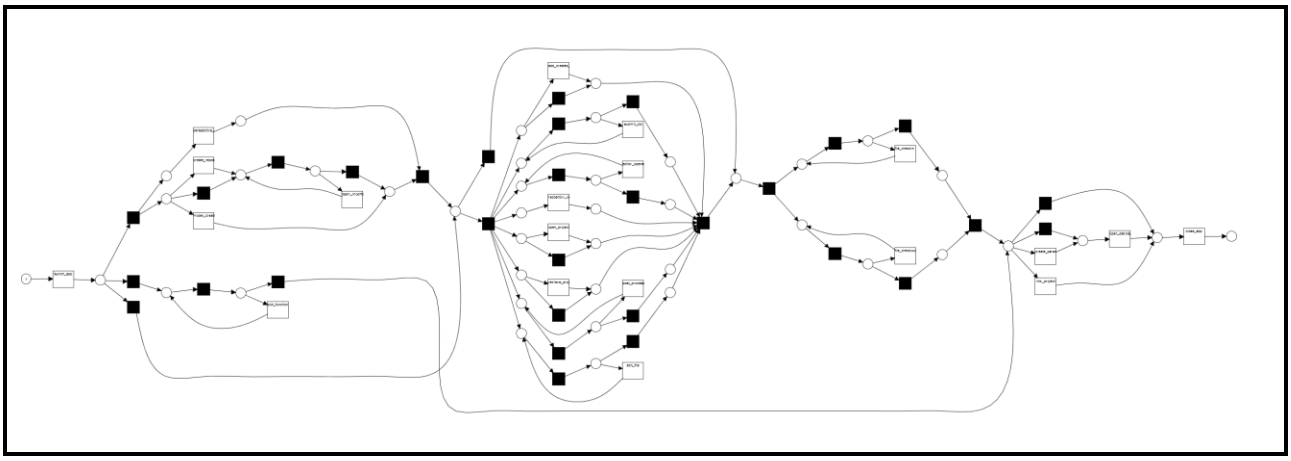

Figure 8. Petri Net of SAW Usage

Next, the team explored the application of a Heuristic Miner to the data. A Heuristic Miner utilizes an algorithm that is highly resilient to noise in the event log (Weijters & van der Aalst, 2003). It also produces output that supports loops as well as frequency counts. In addition, it calculates an overall fitness, indicating the number of cases accounted for in the output. The resulting model indicated two main flows through the application. It appears users tend to most frequently launch the application to either commit files (file_checkin) or retrieve files (file_checkout). The overall fitness of the mined model was .9532, indicating that 95% of the events were accounted for in the model. This model provided a good discussion point with stakeholders. Despite a plethora of requests for new functionality, if the primary utilization of the software as demonstrated by actual behavior is on one of these two predominant pathways, then

it is possible that User Stories relating to these areas should receive higher prioritization. This model also implicitly illuminated the goals of the users. Whereas many of the stakeholders have advocated, for example, for increased investment in editors for simulation code production, the currently demonstrated behavior indicates the primary goals of the users is to store or retrieve files in the application. Such file storage supports developing computational simulations with other team members, as well as providing permanent reference to particular artifact versions.

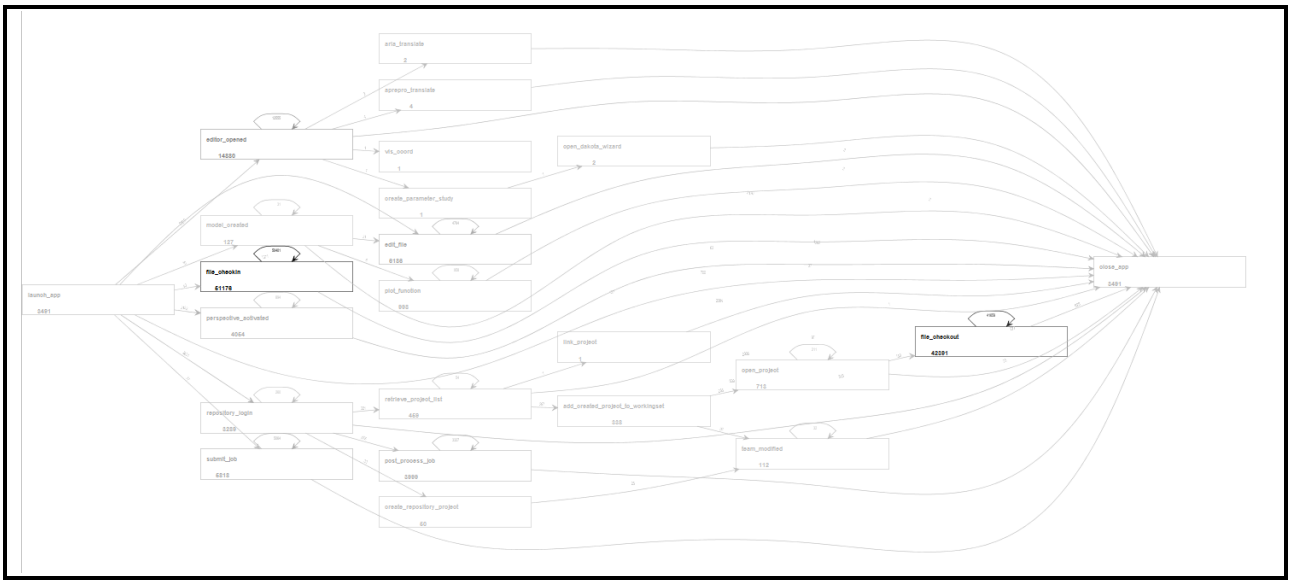

Figure 9. Heuristic Miner Results

The team then investigated the application of a Fuzzy Miner (Günther & van der Aalst, 2007) to the event log. The Fuzzy Miner was developed specifically to deal with less structured processes. The cluster analysis model presented above demonstrated the problem of too much detail in a model. As Günther and van der Aalst (2007) stated, “The problem is that the resulting model shows all details without providing a suitable abstraction” (p. 329). This model provides a visual indication via line thickness of the relative frequency of the pathway executions through the model while reducing the overall displayed noise. Whereas the Heuristic Miner demonstrated the prevalence of two specific goals, this mined result indicated other potential areas for improvement. For example, the relatively thicker loops associated with “open_project” and

“retrieve_project_list” tended to suggest that users are frequently attempting to find a project containing some set of files. Again, this discovery would be a good discussion point with the stakeholders, as developing functionality that allowed for a more efficient retrieval of the desired file from a given analysis project could provide a useful benefit. This discovery also indicates that just because a set of features is present, it does not follow the features are presented in a way that maximizes goal achievement. From a different example, consider how often one must navigate the save dialog in various Microsoft Windows applications, as the directory is not set to a desired location. The functionality to save a file is present, but it is not necessarily implemented in a way that is optimal for achieving a goal.

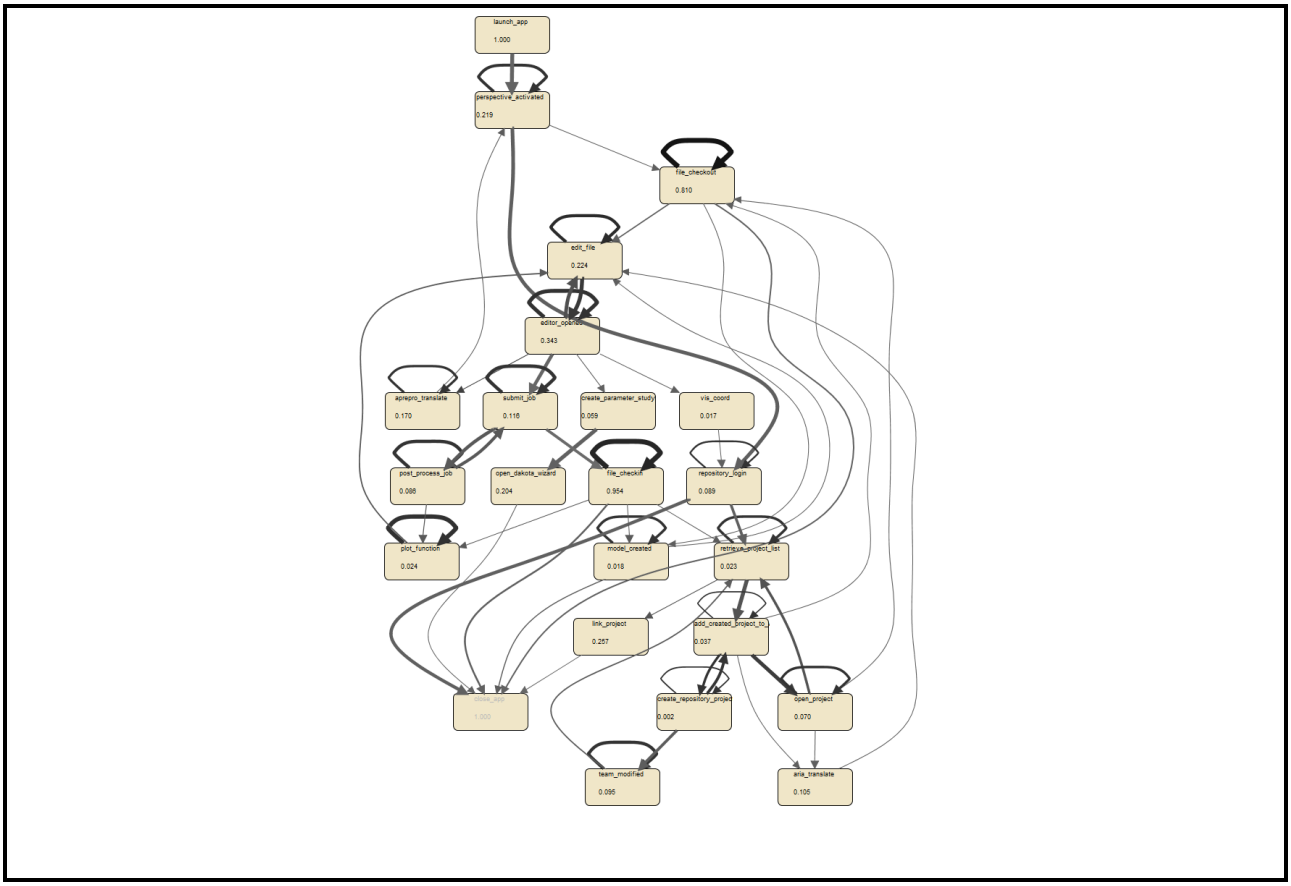

Figure 10. Fuzzy Miner Results

The logged events contained user information. The team has not investigated mining a social network. Other domain specific data is also present in the event log. However, it too has

not yet been examined. The logging of the user and the specific activities could potentially raise privacy concerns. However, the corporate computer usage policy specifically indicates that all activity may be logged and examined, and there should be no expectation of privacy. In contrast, however, some of the collected data is potentially restricted due to need-to-know (NTK) concerns. For example, when editing a simulation input file, it might be helpful to know if the same file is being edited multiple times. In order to answer that question, some tracking of the file would be necessary. The specific file names, however, are potentially NTK controlled. The application, therefore, stores a hash of the file path and name rather than actual name. The hash allows, to a certain extent, for addressing questions of repeated editing without divulging the actual filename. Other such information, such as the project name, is similarly encoded. Though not exactly equivalent, the need to protect information relating to patient data in aforementioned case studies required similar approaches to anonymizing or encoding data. Therefore, while collecting usage information is covered under the corporate policy, it is still necessary to protect certain pieces of the collected information.

This section provided an example of applying Process Mining to a non-PAIS application. Though it is not a case study, it does fit within the general literature of Rubin, Mitsyuk, et al. (2014) or van Genuchten et al. (2014). It provided the definitions for a case and an activity as used by the Sandia Analysis Workbench development team, and noted some issues with how the team has implemented these concepts. In short, while the definitions have provided value as seen in the mined processes, they are not necessarily optimal.

Despite the ability to obtain some visual mined processes, the full benefits for this approach have not yet been realized by the team. For example, the extraction and transformation is not systematically integrated anywhere in the current Sprint cycles. As a result, the extracted

information is used in a very *ad-hoc* fashion. Further, the use has been entirely within the team, and has not been utilized to develop conversations with the stakeholders. Since it is not incorporated into the Sprint tasks, it is not being used to refine the software as it is being developed. There is also an issue related to expertise as well. It takes expertise in the Process Mining literature to properly develop the data transformations from the MySQL repository to the requisite XES file format, as well as how to execute the mining itself. Adjustments to the miners to achieve the best results require in-depth knowledge of the algorithms. These problems were identified in the Manifesto, but in the small team size of many Agile projects (and the SAW project fits into that category), it is a serious barrier to its adoption. Lastly, the information is not being correlated with software quality in general. Though some differences in user behavior have been noted across releases (though not explored in the foregoing), this information has not been quantified, nor has it been related to any measures of improved quality from the user community.

Chapter 5: Conclusion

When a user interacts with a software application, the user has a specific desired goal to achieve. Process Mining can provide insight into the actual utilization of a software program, discovering these ad-hoc processes. In addition, Process Mining can reveal the intentions of the users.

Process Mining requires specific input in order for the algorithms to be effective. The basic input is an event log, consisting of a sequence of events, with each event associated with a case. The event may be augmented with additional attributes, such as a timestamp or a user. Though any software application (or more generally, any information system) has the potential to generate an event log, not all applications are designed to emit such logs. Furthermore, there is a distinct difference between application logging and logging user behavior. The former tends to focus on application logic and error conditions at a level that is overly fine-grained and not reflective of the domain experience of the user or analyst. In contrast, user behavior logging emits behavioral interaction as the logged events. It may be possible to map application behavior to user behavior, but doing so introduces additional effort in terms of developing and maintaining the mapping.

Despite the requirements of Process Mining, it is nonetheless remarkable that one can discover intentional usage and correlated patterns of behavior from logged process traces. These traces may be mined by applying algorithms to “yield an aggregate description of…observed behavior, e.g. in form of a process model” (Günther & van der Aalst, 2007, p. 329). Behavior may be modeled as a process for achieving a desired goal.

Research Questions and Results

The initial research objectives of this thesis were to understand the potential to apply Process Mining to standard, non-process aware software applications (RQ_1). Further, if it appeared reasonable based upon the literature that such an endeavor were possible, to then illuminate the requirements of event logs in order to support such an endeavor (RQ_2). In contrast to Process Mining where — to a certain extent — the ultimate goal is process improvement, this thesis suggested that Process Mining when applied to such instrumented applications could provide data to improve software quality by understanding the experience of a user (RQ_3). In order to improve software quality, however, the focus of software quality must shift from a development process basis focused on features to a user focus based on behavioral interaction with the software application.

The history of Process Mining began with (Cook & Wolf, 1998a) and the desire to understand the behavior of software as expressed by a sequence of actions performed by agents. The captured event data could then be examined in order to construct a finite state model that graphically represented the recurring patterns of behavior. One of the founding documents in the literature, therefore, emphasized the need to understand *behavior* as expressed by agents, and furthermore applied the approach in some instances to non-process aware software.

From this auspicious beginning, however, Process Mining primarily evolved by focusing on workflow systems. Workflow systems have two advantages that supported early investigation: they had a defined reference model for comparison and they were generally designed to emit some type of event log around a series of activities. These advantages allowed Process Mining to focus on the algorithms necessary for accurate process discovery and useful representations of the discovered process. However, the fact workflow systems had defined processes that generally emitted events also obscured the underlying requirements for applying

Process Mining to non-Process Aware Information Systems. In addition, Process Mining has tended to accept or promote the idea that quality is seen in conformance to a (perhaps ideal) process, and that the contribution of Process Mining is in demonstrating variances between an ideal model and the discovered in-use model.

Nonetheless, the difficulty in developing the algorithms to support Process Mining should not be underestimated, and the reduced scope afforded by a focus on workflow systems assisted in this endeavor. Yet as a result, and despite the wide-range of case studies that accompanied the algorithmic development, the literature moved away from user behavior in software. Only a few studies examined software applications directly (e.g., Kwan Hee, Boram, & Jeonghwan, 2015; Rubin, Mitsyuk, et al., 2014; van Genuchten et al., 2014), but these studies did not link the discovered processes with software quality improvement.

In surveying the literature, there was, therefore, some indication that applying Process Mining to a standard software application would be possible even though it was not the primary focus of the literature. This survey provided sufficient support for RQ_1. Based upon this initial condition, it was then possible to extract and expand upon the requirements for instrumenting an application in order to successfully generate event logs that could be analyzed. In generating these requirements, the focus remained on instrumenting an application to highlight user behavior in order to apply discovered user interactions to improve software quality. The discovered process models are most likely to be of benefit if easily incorporated into an Agile-based SDLC. The elaboration based upon the literature provided an answer to RQ_2.

The quasi-case study demonstrated a practical application of an instrumented software application and Process Mining to support software quality improvement, thus providing some initial, albeit tentative support, for RQ_3. An instrumented application emitted event logs that

could then be processed by specific algorithms resulting in a visual process diagram. By analyzing the diagram, potential software quality issues that were not reported by the user community were identified during a SCRUM-based Sprint Review, and the team generated user stories to ameliorate these issues. Consequently, the interweaving of user behavior as documented in event logs, Process Mining, and software quality was achieved in this thesis. The result is a demonstration of a potentially enhanced mechanism for improving subjective, user-based quality by understanding the way users actually interact with a software application.

Support For Software Quality

Software quality is difficult to achieve. Quality itself is not uniformly defined, and has subjective interpretations. Basili (1989) argued that software development has "evolved from focusing on the project, e.g. schedule and resource allocation concerns, to focusing on the product, e.g. reliability and maintenance concerns, to focusing on the process, e.g., improved methods and process models." For Basili, the goal was to improve software quality by implementing a measurement program throughout the entire life cycle of an application. This measurement program was, in effect, an attempt to improve the process of software development.

To a certain extent, the argument for applying Process Mining to instrumented software applications is the need to move beyond the focus on a development process to a focus on how individuals achieve desired goals when interacting with an application. The focus therefore is on the *user behavior* and *user goals*, rather than the process that develops the software. Quality is therefore maximized not through a particular development process but by measuring how well people achieve goals.

DevOps

A newly emerging group of concepts collected under the auspices of "DevOps" (Mueller, 2017) provides another aspect to the foregoing. Among various points of emphasis, DevOps encourages continual application monitoring. This monitoring is not only to assist with operational issues, but also to provide demonstrate to software developers instances where quality (especially standard non-functional requirements such as responsiveness) may be substandard. The emphasis on using metrics feed to the development team to support quality resonates with the general themes of this thesis. However, it is important to note the general description of the collected metrics do not generally favor understanding how a user is interacting with an application, but tend to focus on general responsiveness on page loads, database inserts, etc. This emphasis may be seen, for example, in the types of tools presented in the "Periodic Table of DevOps Tools" (XebiaLabs, 2017). As such, while DevOps is an interesting additional source of encouragement to developers to utilize in-use data to guide quality considerations, without careful additions or additional processing the general discussion in DevOps about application monitoring suffers from many of the defects discussed in Chapter 3.

Future Research

In examining the Process Mining literature, a few areas for future research are evident. One such area, as noted in the Process Mining Manifesto (van der Aalst et al., 2012), is the continued refinement of the algorithms used to discover the processes. Process Conformance algorithms, especially in terms of quantified differences, are also a noted area for continued research. While the discovery of processes from PAIS systems has enjoyed success, the application of Process Mining to non-PAIS systems is a continued area of research (Günther & van der Aalst, 2007).

Several articles noted the need to refine data prior to applying Process Mining techniques. For example, Günther et al. (2008) noted the need to filter and map low-level events from medical devices. Rubin et al. (2007) also required pre-processing of event data in order to derive user-behavior from the logged application data. Khodabandelou et al. (2014) reduced the data set size in order to focus on specific events relevant to source code configuration management. In van der Aalst (2011a), a basic "L*" lifecycle was proposed for event analysis. A part of this lifecycle concerned data gathering and cleaning. The book also noted the general approach of "extract, transform, and load" (p. 97) as a necessary step in order to generate acceptable input for the mining. These proposals, however, are very high-level and simple in their definition. The bulk of the Process Mining literature has concerned itself with the techniques necessary to discover, monitor, or improve processes. As a result, the literature has not presented a well-defined approach to the complete process for data transformation.

The range of needed transformations is potentially large. In addition, in order to allow Process Mining to be a defined and repeatable methodology, the extraction, transformation, and mining steps must be fully defined. The example study examined in Chapter 4 demonstrated a small subset of issues associated with a centralized event logging service. Chapter 3 noted the potential mapping issues associated with cases and activities. Several authors in the literature, as noted elsewhere, have briefly described extraction and transformation steps, but generally have not located these steps within a repeatable process.

As noted in Chapter 2, a few authors (e.g., Hill & Jones, 1999; Schimm, 2002) have attempted to situate Process Mining in the Knowledge Discovery in Databases (KDD) field. The KDD literature(e.g., Fayyad et al., 1996) appears to have a more refined understanding of the issues associated with transforming data into knowledge. Like Process Mining, the goal of KDD

is to discover patterns (processes) that reflect domain specific knowledge (Fayyad et al., 1996, p. 41). The KDD literature, however, has typically presented a more robust approach to the full process for data collection, extraction, transformation, and mining. Consequently, further research into merging these intellectual traditions is likely useful.

Other Considerations

There are a number of limitations in this study. Chief among these are its limitation to a single, quasi-case study. While the investigation into an instrumented software application and the resultant ability to glean information relevant to software quality was fruitful, further research into whether the application of Process Mining in an easily repeatable fashion is clearly needed. Bolt, de Leoni, and van der Aalst (2016) applied Process Mining techniques to online collegiate coursework in a way that provided weekly data summaries for both mined data and process conformance. However, this study is almost unique so far for its easy repeatability.

The Bolt et al. (2016) study is also interesting in that it pursued new types of data visualization, using novel charting rather than the Petri Net or other static model output as mentioned above. Other authors (e.g., de Leoni, Suriadi, Hofstede, & van der Aalst, 2016) have begun to examine animating the discovered processes in order to produce additional insights. One limitation of Process Mining may be found in the relatively limited ways in which the discovered models are represented, and further research into more effective presentations and visualizations would likely assist Process Mining in general. Such improvements might be especially beneficial to applying Process Mining to instrumented software applications as the very nature of the ad-hoc processes lend themselves to more “spaghetti-like” discovered processes. Thus, another limitation of this work is in its very traditional presentation of the discovered processes in the quasi-case study. This visualization limitation, combined with the

difficulty in cleaning and mining the data, likely contributed to the lack of a broad adoption of Process Mining by the team.

If consistent repeatability and a better visualization output mechanism were discovered, it might be easier for teams to incorporate Process Mining into their SDLC. While it has been argued that it should be possible and desirable to incorporate the discovered, in-use processes exhibited by users into an SDLC in order to verify and improve software quality, this thesis did not provide support for the long-term actual inclusion of such techniques. Studies that did directly examine Process Mining and software interaction, such as van Genuchten et al. (2014) and Rubin, Mitsyuk, et al. (2014), also had a single iteration of gleaned information. Thus, while an Agile SDLC that emphasis "early feedback from the user" (Rubin, Lomazova, et al., 2014, p. 71) would seem appropriate for incorporating Process Mining discoveries, to date the literature has not documented on-going efforts by teams to routinely integrate Process Mining into their processes. Further examination of the barriers to such integration would likely be useful.

This thesis also focused on stand-alone applications, and some of the issues with creating a secure Process Mining Service. It may be the case for stand-alone applications that the data collection will be limited to development teams or confined to a single organization, as wide-scale deployment could prove problematic. While the general statements about the requirements for instrumenting an application are applicable to web-based deployments as well, the specifics of such applications were not directly considered. One benefit of web-based applications is the likely increase in the ability to securely collect data, tough potentially coupled with the difficulty in clearly delimiting the start and end points of a user traversal (see also Rubin, Mitsyuk, et al., 2014). Kwan Hee et al. (2015) applied Process Mining techniques to analyze a university website, but the study could not definitively link visits to a user, and had to assume that visits on

different days had particular implications of user behavior (i.e., additional information acquisition which was therefore an implication about how the website was organized). Nonetheless, with the continued rise of Software as a Service (SaaS) applications, additional work on how to expand the initial recommendations detailed herein may prove useful for web-based applications as well.

Finally, it should be noted that the examined Process Mining literature was drawn from academic sources, and looked specifically at the specialized technique of "Process Mining." If non-academic industry utilizes Process Mining techniques and publishes results other than in the academic databases, such knowledge was not utilized. In addition, if the terminology of industry were different from the academic approaches, such articles would not have been located. However, if such articles exist, it is important to note that the references of the academic Process Mining literature are also not providing pointers to them.

Overall, Process Mining can be applied to understanding the behavior of a user when interacting with a software application. It is, however, "an advanced technique" (Rubin, Mitsyuk, et al., 2014, p. 7) for understanding such behavior, and standard applications must be instrumented to emit the requisite event logs in order to support Process Mining. However, in contrast with simply understanding the process that a user undertakes within an application, integrating these discoveries within an Agile-based software development lifecycle has the strong potential to feed continual software quality improvements by documenting potentially deficient areas in an application or by comparing user behavior (and associated information such as timing/effort) between software releases. The ability of Process Mining to provide insight at various levels of analysis — from a single user to groups to a whole community — and to allow

for comparing various discovered models is a potential advancement in the field of software engineering.

References

Accorsi, R., & Stocker, T. (2012). On the exploitation of process mining for security audits: The conformance checking case. *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. Trento, Italy (pp. 1709-1716). doi: 10.1145/2245276.2232051

Accorsi, R., Stocker, T., & Müller, G. (2013). On the exploitation of process mining for security audits: The process discovery case. *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. Coimbra, Portugal (pp. 1462-1468). doi: 10.1145/2480362.2480634

Adzic, G., & Evans, D. (2014). *Fifty quick ideas to improve your user stories* (Kindle ed.). London: Neuri Consulting, LLP.

Agrawal, R., Gunopulos, D., & Leymann, F. (1998). Mining process models from workflow logs. *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology*. Valencia, Spain (pp. 469-483).

Ali, A. (2012). A framework for using cost-benefit analysis in making the case for software upgrade. *Issues in Informing Science & Information Technology, 9*, 399-409.

Alves de Medeiros, A.K., van der Aalst, W.M.P., & Weijters, A.J.M.M. (2008). Quantifying process equivalence based on observed behavior. *Data & Knowledge Engineering, 64*(1), 55-74. doi: http://dx.doi.org/10.1016/j.datak.2007.06.010

Alves de Medeiros, A.K., Weijters, A.J.M.M., & Van der Aalst, W.M.P. (2006). *Genetic process mining: A basic approach and its challenges.* Paper presented at the Business Process Management Workshops.

Alves de Medeiros, A.K., Weijters, A.J.M.M., & van der Aalst, W.M.P. (2007). Genetic process mining: An experimental evaluation. *Data Mining and Knowledge Discovery, 14*(2), 245-304.

Ambler, S.W. (2012a). Active stakeholder participation: An agile best practice. Retrieved from http://agilemodeling.com/essays/activeStakeholderParticipation.htm

Ambler, S.W. (2012b). Introduction to user stories. Retrieved March 3, 2014, from http://www.agilemodeling.com/artifacts/userStory.htm

Amiryar, H. (2012). Agile scrum roles and responsibilities. Retrieved from http://www.pmdocuments.com/2012/09/15/agile-scrum-roles-and-responsibilities/

Apel, S., & Kästner, C. (2009). An overview of feature-oriented software development. *Journal of Object Technology, 8*(5 (July/August)), 49-84.

Appel, S., Kleber, P., Frischbier, S., Freudenreich, T., & Buchmann, A. (2014). Modeling and execution of event stream processing in business processes. *Information Systems, 46*(0), 140-156. doi: http://dx.doi.org/10.1016/j.is.2014.04.002

Bae, J.-S., Jeong, S.-C., Seo, Y., Kim, Y., & Kang, S. (1999). Integration of workflow management and simulation. *Computers & Industrial Engineering, 37*(1–2), 203-206. doi: http://dx.doi.org/10.1016/S0360-8352(99)00055-8

Baier, T., Mendling, J., & Weske, M. (2014). Bridging abstraction layers in process mining. *Information Systems, 46,* 123-139. doi: 10.1016/j.is.2014.04.004

Basili, V.R. (1989, 20-22 Sep 1989). *Software development: A paradigm for the future.* Paper presented at the Computer Software and Applications Conference, 1989. COMPSAC 89., Proceedings of the 13th Annual International.

Bates, P. (1988). Debugging heterogeneous distributed systems using event-based models of behavior. *Proceedings of the 1988 ACM SIGPLAN and SIGOPS workshop on parallel and distributed debugging*. Madison, Wisconsin, USA (pp. 11-22). doi: 10.1145/68210.69217

Beaton, W. (2011). Bye bye mon UDC. Retrieved February 16, 2015, from https://waynebeaton.wordpress.com/2011/09/16/bye-bye-mon-udc/

Benner-Wickner, M., Book, M., Bruckmann, T., & Gruhn, V. (2014, 1-2 Sept. 2014). *Examining case management demand using event log complexity metrics.* Paper presented at the Enterprise Distributed Object Computing Conference Workshops and Demonstrations (EDOCW), 2014 IEEE 18th International.

Bezerra, F., & Wainer, J. (2008). Anomaly detection algorithms in logs of process aware systems. *Proceedings of the 2008 ACM symposium on Applied computing*. Fortaleza, Ceara, Brazil (pp. 951-952). doi: 10.1145/1363686.1363904

Bloch, H. (2013). Use simplified lifecycle-cost computations to justify upgrades. *Chemical Engineering, 120*(1), 53-56.

Boehm, B.W. (1978). *Characteristics of software quality*. New York: American Elsevier.

Boehm, B.W. (1986). A spiral model of software development and enhancement. *SIGSOFT Softw. Eng. Notes, 11*(4), 14-24. doi: 10.1145/12944.12948

Boehm, B.W. (1989). *Software risk management*. Washington, D.C.: IEEE Computer Society Press.

Boehm, B.W., & Turner, R. (2004). *Balancing agility and discipline: A guide for the perplexed*. Crawsfordsville, Indiana: Addison-Wesley.

Bolt, A., de Leoni, M., & van der Aalst, W.M.P. (2016). Scientific workflows for process mining: Building blocks, scenarios, and implementation. *International Journal on Software Tools for Technology Transfer, 18*(6), 607-628. doi: 10.1007/s10009-015-0399-5

Borrego, D., & Barba, I. (2014). Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Systems with Applications, 41*(11), 5340-5352. doi: 10.1016/j.eswa.2014.03.010

Bose, R.P.J.C., Mans, R.S., & van der Aalst, W.M.P. (2013, 16-19 April 2013). *Wanna improve process mining results?* Paper presented at the Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on.

Bratosin, C., Sidorova, N., & van der Aalst, W.M.P. (2010, 18-23 July 2010). *Distributed genetic process mining.* Paper presented at the Evolutionary Computation (CEC), 2010 IEEE Congress on.

Bright, P. (2015). Windows 10's privacy policy is the new normal. Retrieved August 8, 2015, from http://arstechnica.com/information-technology/2015/08/windows-10s-privacy-policy-is-the-new-normal/

Brooks, F.P. (1995). *The mythical man-month : Essays on software engineering* (Anniversary, Kindle ed.). Reading, Mass.: Addison-Wesley Pub. Co.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., . . . Zazworka, N. (2010). *Managing technical debt in software-reliant systems*. Paper presented at the Proceedings of the FSE/SDP workshop on Future of software engineering research, Santa Fe, New Mexico, USA.

Buffett, S., & Geng, L. (2010). Using classification methods to label tasks in process mining. *Journal of Software Maintenance & Evolution: Research & Practice, 22*(6/7), 497-517. doi: 10.1002/smr.463

Bui, D.B., Hadzic, F., & Hecker, M. (2012). Application of tree-structured data mining for analysis of process logs in xml format. *Proceedings of the Tenth Australasian Data Mining Conference - Volume 134*. Sydney, Australia (pp. 109-118).

Buse, R.P.L., & Zimmermann, T. (2012, 2-9 June 2012). *Information needs for software development analytics.* Paper presented at the Software Engineering (ICSE), 2012 34th International Conference on.

Calvanese, D., Giacomo, G.D., & Montali, M. (2013). Foundations of data-aware process analysis: A database theory perspective. *Proceedings of the 32nd symposium on Principles of database systems*. New York, New York, USA (pp. 1-12). doi: 10.1145/2463664.2467796

Caron, F., Vanthienen, J., & Baesens, B. (2013). Comprehensive rule-based compliance checking and risk management with process mining. *Decision Support Systems, 54*(3), 1357-1369. doi: http://dx.doi.org/10.1016/j.dss.2012.12.012

Cavano, J.P., & McCall, J.A. (1978). *A framework for the measurement of software quality.* Paper presented at the ACM SIGMETRICS Performance Evaluation Review.

Chartered Quality Institute. (2013). Evolution of quality thinking, post c1970. Retrieved June 2, 2013, from http://www.thecqi.org/Knowledge-Hub/Knowledge-portal/Concepts-of-quality/Evolution-of-quality-thinking/

Christopher, F. (2003). *Software verification and validation within the (rational) unified process*.

Chu-Carroll, M. (2016). Time in distributed systems: Lamport timestamps. Retrieved March 18, 2016, from http://www.goodmath.org/blog/2016/03/16/time-in-distributed-systems-lamport-timestamps/

Claes, J., & Poels, G. (2014). Merging event logs for process mining: A rule based merging method and rule suggestion algorithm. *Expert Systems with Applications, 41*(16), 7291-7306. doi: 10.1016/j.eswa.2014.06.012

Classen, A., Heymans, P., & Schobbens, P.-Y. (2008). *What's in a feature: A requirements engineering perspective.* Paper presented at the Fundamental Approaches to Software Engineering, Budapest, Hungary.

Cleland-Huang, J., & Mobasher, B. (2008). Using data mining and recommender systems to scale up the requirements process. *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems*. Leipzig, Germany (pp. 3-6). doi: 10.1145/1370700.1370702

Cohn, M. (2004). *User stories applied : For agile software development*. Boston: Addison-Wesley.

Cook, J.E., Du, Z., Liu, C., & Wolf, A.L. (2004). Discovering models of behavior for concurrent workflows. *Computers in Industry, 53*(3), 297-319. doi: http://dx.doi.org/10.1016/j.compind.2003.10.005

Cook, J.E., & Wolf, A.L. (1994, 10-11 Oct). *Toward metrics for process validation.* Paper presented at the Third International Conference on the Software Process.

Cook, J.E., & Wolf, A.L. (1998a). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology, 7*(3), 215-249. doi: 10.1145/287000.287001

Cook, J.E., & Wolf, A.L. (1998b). *Event-based detection of concurrency.* Paper presented at the Sixth International Symposium on the Foundations of Software Engineering, Lake Buena Vista, FL, USA.

Cooper, A., Reimann, R., & Cronin, D. (2007). *About face 3 : The essentials of interaction design* (3rd, Kindle ed.). Indianapolis, IN: Wiley Pub.

Crispin, L., & Gregory, J. (2009). *Agile testing : A practical guide for testers and agile teams*. Upper Saddle River, NJ: Addison-Wesley.

Datta, A. (1998). Automating the discovery of as-is business process models: Probabilistic and algorithmic approaches. *Information Systems Research, 9*(3), 275-301.

de Leoni, M., Suriadi, S., Hofstede, A., & van der Aalst, W.M.P. (2016). Turning event logs into process movies: Animating what has really happened. *Software & Systems Modeling, 15*(3), 707-732. doi: 10.1007/s10270-014-0432-2

de Leoni, M., van der Aalst, W.M.P., & Dees, M. (2016). A general process mining framework for correlating, predicting and clustering dynamic behavior based on event logs. *Information Systems, 56*, 235-257. doi: 10.1016/j.is.2015.07.003

Debra. (2014). 5 annoying pivot table problems. Retrieved November 11, 2017, from http://www.pivot-table.com/2014/09/03/5-annoying-pivot-table-problems/

Denning, P.J. (2016). Software quality. *Communications of the ACM, 59*(9), 23-25. doi: 10.1145/2971327

Diamantini, C., Potena, D., & Storti, E. (2012). Mining usage patterns from a repository of scientific workflows. *Proceedings of the 27th Annual ACM Symposium on Applied Computing*. Trento, Italy (pp. 152-157). doi: 10.1145/2245276.2245307

Duan, C., Cleland-Huang, J., & Mobasher, B. (2008). A consensus based approach to constrained clustering of software requirements. *Proceedings of the 17th ACM conference on Information and knowledge management*. Napa Valley, California, USA (pp. 1073-1082). doi: 10.1145/1458082.1458225

Dustdar, S., Hoffmann, T., & van der Aalst, W.M.P. (2005). Mining of ad-hoc business processes with TeamLog. *Data & Knowledge Engineering, 55*(2), 129-158. doi: 10.1016/j.datak.2005.02.002

Evermann, J., & Assadipour, G. (2014). Big data meets process mining: Implementing the alpha algorithm with map-reduce. *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. Gyeongju, Republic of Korea (pp. 1414-1416). doi: 10.1145/2554850.2555076

Fahland, D., & van der Aalst, W.M.P. (2013). Simplifying discovered process models in a controlled manner. *Information Systems, 38*(4), 585-605. doi: 10.1016/j.is.2012.07.004

Fayyad, U., Piatetsky-Shapiro, G., & Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI magazine, 17*(3), 37-54.

Ferre, X., Juristo, N., & Moreno, A.M. (2004). Improving software engineering practice with HCI aspects *Software engineering research and applications* (pp. 349-363): Springer.

Floudas, C.A., Pardalos, P.M., Adjiman, C., Esposito, W.R., Gümüs, Z.H., Harding, S.T., . . . Schweiger, C.A. (2013). *Handbook of test problems in local and global optimization* (Vol. 33): Springer Science & Business Media.

Folino, F., Greco, G., Guzzo, A., & Pontieri, L. (2009). Discovering expressive process models from noised log data. *Proceedings of the 2009 International Database Engineering & Applications Symposium*. Cetraro - Calabria, Italy (pp. 162-172). doi: 10.1145/1620432.1620449

Fowler, M., & Highsmith, J. (2001). The agile manifesto. Retrieved August 31, 2013, from http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns : Elements of reusable object-oriented software*. Reading, Mass.: Addison-Wesley.

Gerring, J. (2004). What is a case study and what is it good for? *American Political Science Review, 98*(02), 341-354.

Giuseppe, C., Valerio, M., Teresa, M., & Carmela, S.L. (2014). A simulation approach in process mining conformance analysis. The introduction of a brand new BPMN element. *IERI Procedia, 6*(0), 45-51. doi: http://dx.doi.org/10.1016/j.ieri.2014.03.008

Glass, R.L. (2003). *Facts and fallacies of software engineering*. Boston, MA: Addison-Wesley.

Goedertier, S., De Weerdt, J., Martens, D., Vanthienen, J., & Baesens, B. (2011). Process discovery in event logs: An application in the telecom industry. *Applied Soft Computing, 11*(2), 1697-1710. doi: http://dx.doi.org/10.1016/j.asoc.2010.04.025

Günther, C.W., Rozinat, A., van der Aalst, W.M.P., & van Uden, K. (2008). Monitoring deployed application usage with process mining. *BPM Center Report BPM-08-11*, 1-8.

Günther, C.W., & van der Aalst, W.M.P. (2007). Fuzzy mining–adaptive process simplification based on multi-perspective metrics *Business process management* (pp. 328-343): Springer.

Günther, C.W., & Verbeek, E. (2014). Openxes: Developer guide. Retrieved March 3, 2015, from http://www.xes-standard.org/_media/openxes/openxesdeveloperguide-2.0.pdf

Gupta, M. (2014). Nirikshan: Process mining software repositories to identify inefficiencies, imperfections, and enhance existing process capabilities. *Companion Proceedings of the 36th International Conference on Software Engineering*. Hyderabad, India (pp. 658-661). doi: 10.1145/2591062.2591080

Gupta, M., & Sureka, A. (2014). Nirikshan: Mining bug report history for discovering process maps, inefficiencies and inconsistencies. *Proceedings of the 7th India Software Engineering Conference*. Chennai, India (pp. 1-10). doi: 10.1145/2590748.2590749

Gupta, M., Sureka, A., & Padmanabhuni, S. (2014). Process mining multiple repositories for software defect resolution from control and organizational perspective. *Proceedings of the 11th Working Conference on Mining Software Repositories*. Hyderabad, India (pp. 122-131). doi: 10.1145/2597073.2597081

Hammori, M., Herbst, J., & Kleiner, N. (2006). Interactive workflow mining—requirements, concepts and implementation. *Data & Knowledge Engineering, 56*(1), 41-63. doi: 10.1016/j.datak.2005.02.006

Hathaway, T., & Hathaway, A. (2013). *Writing effective user stories* (Kindle ed.): BA-Experts.

Henry, J., Henry, S., Kafura, D., & Matheson, L. (1994). Improving software maintenance at Martin Marietta. *IEEE Software, 11*, 67-75.

Herbst, J. (2000a). *Dealing with concurrency in workflow induction.* Paper presented at the European Concurrent Engineering Conference. SCS Europe.

Herbst, J. (2000b). A machine learning approach to workflow management *Machine learning: Ecml 2000* (pp. 183-194): Springer.

Herbst, J., & Karagiannis, D. (1998). *Integrating machine learning and workflow management to support acquisition and adaptation of workflow models.* Paper presented at the Ninth International Workshop on Database and Expert Systems Applications, 1998.

Hill, A.E., & Jones, W.T. (1999, April 15-18). Retrospective case base browsing: A data mining process enhancement. *Proceedings of the 37th annual Southeast regional conference (CD-ROM)*. Mobile, AL (pp. 49-53). doi: 10.1145/306363.306425

Huang, S.-K., & Liu, K.-m. (2005). Mining version histories to verify the learning process of legitimate peripheral participants. *SIGSOFT Softw. Eng. Notes, 30*(4), 1-5. doi: 10.1145/1082983.1083158

Huo, M., Zhang, H., & Jeffery, R. (2006). An exploratory study of process enactment as input to software process improvement. *Proceedings of the 2006 International Workshop on Software Quality*. Shanghai, China (pp. 39-44). doi: 10.1145/1137702.1137711

IEEE. (2008). IEEE standard for software and system test documentation. *IEEE Std 829-2008*, 1-150. doi: 10.1109/IEEESTD.2008.4578383

ISO/IEC/IEEE. (2010). Systems and software engineering -- vocabulary (ISO/IEC/IEEE 24765:2010(e)). Geneva/New York: ISO/IEEE.

Jans, M., Alles, M., & Vasarhelyi, M. (2013). The case for process mining in auditing: Sources of value added and areas of application. *International Journal of Accounting Information Systems, 14*(1), 1-20. doi: http://dx.doi.org/10.1016/j.accinf.2012.06.015

Jans, M., Lybaert, N., & Vanhoof, K. (2010). Internal fraud risk reduction: Results of a data mining case study. *International Journal of Accounting Information Systems, 11*(1), 17-41. doi: http://dx.doi.org/10.1016/j.accinf.2009.12.004

Kaymak, U., Mans, R., van de Steeg, T., & Dierks, M. (2012, 14-17 Oct. 2012). *On process mining in health care.* Paper presented at the Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on.

Khodabandelou, G., Hug, C., Deneckère, R., & Salinesi, C. (2014). Unsupervised discovery of intentional process models from event logs. *Proceedings of the 11th Working Conference on Mining Software Repositories*. Hyderabad, India (pp. 282-291). doi: 10.1145/2597073.2597101

Khodyrev, I., & Popova, S. (2014). Discrete modeling and simulation of business processes using event logs. *Procedia Computer Science, 29*(0), 322-331. doi: http://dx.doi.org/10.1016/j.procs.2014.05.029

Kim, G., Behr, K., & Spafford, G. (2013). *The phoenix project: A novel about IT, DevOps, and helping your business win*. Portland, OR: IT Revolution Press.

Kovács, M., & Gönczy, L. (2008). Simulation and formal analysis of workflow models. *Electronic Notes in Theoretical Computer Science, 211*(0), 221-230. doi: http://dx.doi.org/10.1016/j.entcs.2008.04.044

Kwan Hee, H., Boram, H., & Jeonghwan, J. (2015). A navigation pattern analysis of university department's websites using a processing mining approach. *Innovations in Education & Teaching International, 52*(5), 485-498. doi: 10.1080/14703297.2013.832634

Kwanghoon, K. (2009, 22-25 Sept. 2009). *Mining workflow processes from XML-based distributed workflow event logs.* Paper presented at the International Conference on Parallel Processing Workshops, 2009.

LeBlanc, R.J., & Robbins, A.D. (1985). *Event-driven monitoring of distributed programs.* Paper presented at the ICDCS.

Leemans, S.J., Fahland, D., & van der Aalst, W.M.P. (2014). *Discovering block-structured process models from event logs containing infrequent behaviour.* Paper presented at the Business Process Management Workshops.

Locke, J. (1959). *An essay concerning human understanding*. New York,: Dover Publications.

M. Valle, A., A.P. Santos, E., & R. Loures, E. (2017). Applying process mining techniques in software process appraisals. *Information & Software Technology, 87*, 19-31. doi: 10.1016/j.infsof.2017.01.004

Ma, H.U.I., Tang, Y., & Wu, L. (2011). Incremental mining of processes with loops. *International Journal on Artificial Intelligence Tools, 20*(1), 221-235.

Magnacca, M. (2009). *So what? : How to communicate what really matters to your audience*. Upper Saddle River, N.J.: FT Press.

Mahoney, M.S. (2004). Finding a history for software engineering. *Annals of the History of Computing, IEEE, 26*(1), 8-19. doi: 10.1109/MAHC.2004.1278847

Maruster, L., van der Aalst, W.M.P., Weijters, A.J.M.M., van den Bosch, A., & Daelemans, W. (2001). *Automated discovery of workflow models from hospital data.* Paper presented at the 13th Dutch-Belgian Artificial Intelligence Conference, De Rode Hoed, Amsterdam, The Netherlands.

Maurer, F., & Melnik, G. (2006). *Agile methods: Moving towards the mainstream of the software industry*. Paper presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China.

Michiels, C., Snoeck, M., Lemahieu, W., Goethals, F., & Dedene, G. (2003). *A layered architecture sustaining model-driven and event-driven software development.* Paper presented at the Perspectives of System Informatics.

Mittal, M., & Sureka, A. (2014). Process mining software repositories from student projects in an undergraduate software engineering course. *Companion Proceedings of the 36th*

International Conference on Software Engineering. Hyderabad, India (pp. 344-353). doi: 10.1145/2591062.2591152

Montani, S., Leonardi, G., Quaglini, S., Cavallini, A., & Micieli, G. (2014). Improving structural medical process comparison by exploiting domain knowledge and mined information. *Artificial Intelligence in Medicine, 62*(1), 33-45. doi: 10.1016/j.artmed.2014.07.001

mountaingoatsoftware.com. (n.d.). User stories. Retrieved February 12, 2015, from https://www.mountaingoatsoftware.com/agile/user-stories

Mueller, E. (2017). What is DevOps? Retrieved November 1, 2017, from https://theagileadmin.com/what-is-devops/

Norman, D.A. (2002). *The design of everyday things* (1st Basic paperback. ed.). New York: Basic Books.

Olson, K.H., Friendman-Hill, E.J., Hoffman, E.L., Gibson, M.J., Greenfield, J.A., & Clay, R.L. (2015). *Process mining and agile methods in the Sandia Analysis Workbench*. Paper presented at the National Laboratories Information Technology Summit, Seattle, WA. https://www.fbcinc.com/e/nlit/presentations/Monday/OlsonKevin-NLIT2015.pdf

Patton, J. (2014). *User story mapping* (Kindle ed.). Sebastopol, CA: O'Reilly.

Paula, M. (2009). Inception of software validation and verification practices within CMMI level 2. In J. M. Ricardo & K. Rick (Eds.), *International conference on software engineering advances* (pp. 536-541).

Pérez-Castillo, R., Weber, B., de Guzmán, I.G.-R., & Piattini, M. (2011). Process mining through dynamic analysis for modernising legacy systems. *IET Software, 5*(3), 304-319. doi: 10.1049/iet-sen.2010.0103

Perimal-Lewis, L., Vries, D.D., & Thompson, C.H. (2014). Health intelligence: Discovering the process model using process mining by constructing start-to-end patient journeys. *Proceedings of the Seventh Australasian Workshop on Health Informatics and Knowledge Management - Volume 153*. Auckland, New Zealand (pp. 59-67).

Pfleeger, S.L. (1998). *Software engineering : Theory and practice*. Upper Saddle River, NJ: Prentice Hall.

Pinter, S.S., & Golani, M. (2004). Discovering workflow models from activities' lifespans. *Computers in Industry, 53*(3), 283-296. doi: http://dx.doi.org/10.1016/j.compind.2003.10.004

Pogue, D. (2015). The upgrade game. *Scientific American, 312*(6), 29-29.

Pool, M. (2008). The easy way to writing good user stories. Retrieved from http://codesqueeze.com/the-easy-way-to-writing-good-user-stories/

Raymond, E.S. (2008). *The cathedral and the bazaar: Musings on linux and open source by an accidental revolutionary* (Kindle ed.). Sebastopol, CA: O'Reilly Media.

Rozinat, A., Mans, R.S., Song, M., & van der Aalst, W.M.P. (2009). Discovering simulation models. *Information Systems, 34*(3), 305-327. doi: 10.1016/j.is.2008.09.002

Rozinat, A., & van der Aalst, W.M.P. (2008). Conformance checking of processes based on monitoring real behavior. *Information Systems, 33*(1), 64-95. doi: 10.1016/j.is.2007.07.001

Rozinat, A., Wynn, M.T., van der Aalst, W.M.P., ter Hofstede, A.H.M., & Fidge, C.J. (2009). Workflow simulation for operational decision support. *Data & Knowledge Engineering, 68*(9), 834-850. doi: http://dx.doi.org/10.1016/j.datak.2009.02.014

Rubin, V.A., Günther, C.W., van der Aalst, W.M.P., Kindler, E., van Dongen, B.F., & Schäfer, W. (2007). Process mining framework for software processes. In Q. Wang, D. Pfahl & D. M. Raffo (Eds.), *Software process dynamics and agility* (pp. 169-181). Berlin: Springer.

Rubin, V.A., Lomazova, I., & van der Aalst, W.M.P. (2014). Agile development with software process mining. *Proceedings of the 2014 International Conference on Software and System Process*. Nanjing, China (pp. 70-74). doi: 10.1145/2600821.2600842

Rubin, V.A., Mitsyuk, A.A., Lomazova, I.A., & van der Aalst, W.M.P. (2014). Process mining can be applied to software too! *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Torino, Italy (pp. 1-8). doi: 10.1145/2652524.2652583

Sabhnani, M.R., Neill, D.B., & Moore, A.W. (2005). Detecting anomalous patterns in pharmacy retail data. *Data Mining Methods for Anomaly Detection*, 58.

Sahin, I., & Zahedi, F. (2001). Control limit policies for warranty, maintenance and upgrade of software systems. *IIE Transactions, 33*(9), 729.

Schimm, G. (2002). Process miner—a tool for mining process schemes from event-based data *Logics in artificial intelligence* (pp. 525-528): Springer.

Schimm, G. (2004). Mining exact models of concurrent workflows. *Computers in Industry, 53*(3), 265-281. doi: http://dx.doi.org/10.1016/j.compind.2003.10.003

Schwaber, K., & Beedle, M. (2002). *Agile software development with scrum*. Upper Saddle River, NJ: Prentice Hall.

Shewhart, W.A. (1931). *Economic control of quality of manufactured product*. New York: D. Van Nostrand Company.

Siau, K., Nah, F.F.-H., & Teng, L. (2002). Acceptable internet use policy. *Commun. ACM, 45*(1), 75-79. doi: 10.1145/502269.502302

Snoeck, M., Poelmans, S., & Dedene, G. (2000). A layered software specification architecture *Conceptual modeling—er 2000* (pp. 454-469): Springer.

Stanton, J.M., Stam, K.R., Mastrangelo, P., & Jolton, J. (2005). Analysis of end user security behaviors. *Computers & Security, 24*(2), 124-133. doi: http://dx.doi.org/10.1016/j.cose.2004.07.001

Summers, B. (2014). 50 tweetable quotes about ux. Retrieved from http://www.dtelepathy.com/blog/inspiration/50-shareable-ux-quotes

Sun, C., Du, J., Chen, N., Khoo, S.-C., & Yang, Y. (2013). Mining explicit rules for software process evaluation. *Proceedings of the 2013 International Conference on Software and System Process*. San Francisco, CA, USA (pp. 118-125). doi: 10.1145/2486046.2486067

Sun, C., Zhang, H., Lou, J.-G., Zhang, H., Wang, Q., Zhang, D., & Khoo, S.-C. (2014). Querying sequential software engineering data. *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. Hong Kong, China (pp. 700-710). doi: 10.1145/2635868.2635902

Tomas, B. (2003). *A case study investigating the characteristics of verification and validation activities in the software development process*.

Turner, C.J., Tiwari, A., & Mehnen, J. (2008). A genetic programming approach to business process mining. *Proceedings of the 10th annual conference on Genetic and evolutionary computation*. Atlanta, GA, USA (pp. 1307-1314). doi: 10.1145/1389095.1389345

van der Aalst, W.M.P. (1998). The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers, 8*(01), 21-66.

van der Aalst, W.M.P. (2005). Business alignment: Using process mining as a tool for delta analysis and conformance testing. *Requirements Engineering, 10*(3), 198-211. doi: 10.1007/s00766-005-0001-x

van der Aalst, W.M.P. (2011a). *Process mining : Discovery, conformance and enhancement of business processes* (Kindle ed.). New York: Springer.

van der Aalst, W.M.P. (2011b). Process mining manifesto: Toward real business intelligence. *Computing Now.* Retrieved September 27, 2014, from http://www.computer.org.dml.regis.edu/portal/web/computingnow/pmm#lm

van der Aalst, W.M.P. (2012a). Process mining: Making knowledge discovery process centric. *SIGKDD Explor. Newsl., 13*(2), 45-49. doi: 10.1145/2207243.2207251

van der Aalst, W.M.P. (2012b). Process mining: Overview and opportunities. *ACM Trans. Manage. Inf. Syst., 3*(2), 1-17. doi: 10.1145/2229156.2229157

van der Aalst, W.M.P. (2012c). What makes a good process model? *Software & Systems Modeling, 11*(4), 557-569. doi: 10.1007/s10270-012-0265-9

van der Aalst, W.M.P., Adriansyah, A., Alves de Medeiros, A.K., Arcieri, F., Baier, T., Blickle, T., . . . Wynn, M. (2012). Process mining manifesto. In F. Daniel, K. Barkaoui & S. Dustdar (Eds.), *Business process management workshops* (Vol. 99, pp. 169-194): Springer Berlin Heidelberg.

van der Aalst, W.M.P., Reijers, H.A., & Song, M. (2005). Discovering social networks from event logs. *Computer Supported Cooperative Work: The Journal of Collaborative Computing, 14*(6), 549-593. doi: 10.1007/s10606-005-9005-9

van der Aalst, W.M.P., Reijers, H.A., Weijters, A.J.M.M., van Dongen, B.F., Alves de Medeiros, A.K., Song, M., & Verbeek, H.M.W. (2007). Business process mining: An industrial application. *Information Systems, 32*(5), 713-732. doi: http://dx.doi.org/10.1016/j.is.2006.05.003

van der Aalst, W.M.P., & van Dongen, B.F. (2013). Discovering Petri nets from event logs. In K. Jensen, W. P. van der Aalst, G. Balbo, M. Koutny & K. Wolf (Eds.), *Transactions on Petri nets and other models of concurrency vii* (Vol. 7480, pp. 372-422): Springer Berlin Heidelberg.

van der Aalst, W.M.P., van Dongen, B.F., Herbst, J., Maruster, L., Schimm, G., & Weijters, A.J.M.M. (2003). Workflow mining: A survey of issues and approaches. *Data & Knowledge Engineering, 47*(2), 237-267. doi: http://dx.doi.org/10.1016/S0169-023X(03)00066-1

van der Aalst, W.M.P., & Weijters, A.J.M.M. (2004). Process mining: A research agenda. *Computers in Industry, 53*(3), 231-244. doi: http://dx.doi.org/10.1016/j.compind.2003.10.001

van der Aalst, W.M.P., Weijters, A.J.M.M., & Maruster, L. (2002). Workflow mining: Which processes can be rediscovered: Citeseer.

van der Aalst, W.M.P., Weijters, A.J.M.M., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering, 16*(9), 1128-1142. doi: 10.1109/TKDE.2004.47

van Dongen, B.F., & van der Aalst, W.M.P. (2005). *A meta model for process mining data.* Paper presented at the EMOI-INTEROP, Porto, Portugal.

van Genuchten, M., Mans, R., Reijers, H., & Wismeijer, D. (2014). Is your upgrade worth it? Process mining can tell. *IEEE Software*(September/October), 94-100.

van Solingen, R., Kusters, R.J., Trienekens, J.J.M., & van Uijtregt, A. (1999). Product-focused software process improvement (p-spi): Concepts and their application. *Quality and*

Reliability Engineering International, 15(6), 475-483. doi: 10.1002/(SICI)1099-1638(199911/12)15:6<475::AID-QRE296>3.0.CO;2-1

VersionOne. (n.d.). Feature estimation. Retrieved from https://www.versionone.com/agile-101/agile-project-management-customer-management-best-practices/agile-feature-estimation/

Vogelgesang, T., & Appelrath, H.-J. (2013). Multidimensional process mining: A flexible analysis approach for health services research. *Proceedings of the Joint EDBT/ICDT 2013 Workshops*. Genoa, Italy (pp. 17-22). doi: 10.1145/2457317.2457321

Vogels, W. (2009). Eventually consistent. *Communications of the ACM, 52*(1), 40-44.

Wallace, D.R., & Fujii, R.U. (1989). Software verification and validation: An overview. *IEEE Software, 6*(3), 10-17.

Weijters, A.J.M.M., & van der Aalst, W.M.P. (2001). *Rediscovering workflow models from event-based data.* Paper presented at the Proceedings of the 11th Dutch-Belgian Conference on Machine Learning (Benelearn 2001).

Weijters, A.J.M.M., & van der Aalst, W.M.P. (2003). Rediscovering workflow models from event-based data using Little Thumb. *Integrated Computer-Aided Engineering, 10*(2), 151-162.

Wen, L., Wang, J., van der Aalst, W.M.P., Huang, B., & Sun, J. (2010). Mining process models with prime invisible tasks. *Data & Knowledge Engineering, 69*(10), 999-1021. doi: http://dx.doi.org/10.1016/j.datak.2010.06.001

XebiaLabs. (2017). Periodic table of DevOps tools. Retrieved November 1, 2017, from https://xebialabs.com/periodic-table-of-devops-tools/

Yue, D., Wu, X., Wang, H., & Bai, J. (2011, 13-15 May 2011). *A review of process mining algorithms.* Paper presented at the Business Management and Electronic Information (BMEI), 2011 International Conference on.

Zakia, R. (1995). Quotes for teachers. Retrieved from http://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1381&context=article

Zhang, D., Dang, Y., Lou, J.-G., Han, S., Zhang, H., & Xie, T. (2011). *Software analytics as a learning case in practice: Approaches and experiences*. Paper presented at the Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering, Lawrence, Kansas, USA.

Zhao, W., Liu, J., Ye, D., & Wei, J. (2013). Mining user daily behavior patterns from access logs of massive software and websites. *Proceedings of the 5th Asia-Pacific Symposium on Internetware*. Changsha, China (pp. 1-4). doi: 10.1145/2532443.2532462

Zimmermann, T., Weisgerber, P., Diehl, S., & Zeller, A. (2004). Mining version histories to guide software changes. *Proceedings of the 26th International Conference on Software Engineering*. (pp. 563-572).

Zou, Y., & Hung, M. (2006). *An approach for extracting workflows from e-commerce applications.* Paper presented at the Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on.