Contents

1 Introduction

Software development is a process that involves creativity. Yet, practical software development projects are executed with specific requirements on time, budget and quality. In order to fulfill these requirements the development process must be monitored. For example, it is important to know what type of work is being done at a certain moment in time and by whom. This is for instance the case of large development projects where coordination mechanisms emerge spontaneously among developers who want to contribute with their code. These type of projects are difficult to control for several reasons. First, there is no clear understanding in how far a certain piece of code advances the current status of the project. Second, a piece of code written by a developer goes through different stages of code-review and it is difficult to predict whether it will ever be merged with the main source. Third, coordination of work may involve a large number of message exchanges among many developers in forums. Hence, it is not feasible to manually oversee what work is going on at a particular point in time.

Literature has tackled this problem from different angles. In the area of process mining approaches exist that exploit event logs for abstracting a process model. In the area of software engineering, software repositories have been explicitly studied. These type of works provide several metrics that help with understanding various aspects of development. However, process mining approaches only work with event logs where activities are explicitly recorded in the log. This is not the case with software development where data is rather unstructured. Likewise, software engineering approaches lack a perspective about work patterns.

This study addresses the discovery of work patterns from software development event data. To this end it defines concepts to capture work from software repositories. This allows to construct several discovery techniques that provide information about different aspects of the development process. In this sense, this work provides a bridge between process mining and software engineering. The findings of this work enable project managers as well as software developers to raise transparency about the actual development based on facts. As a result, it enables both understanding the current status and monitoring for potentially unwanted patterns of work.

The rest of this proposal is organized as follows. Section 2 describes the problem, the existing literature, and derives the solution requirements. Section 3 provides background knowledge on the related fields of process mining, text mining, and mining software repositories. Section 4 explains the research method, the dataset and preliminary results in addressing the research requirements. Section 6 presents the expected results and future steps towards the completion of this dissertation. Section 7 draws the implications for research and presents a dissemination plan.

2 Problem Definition

Software development processes are highly complex endeavors that require the coordination of multiple resources. Compared to standard business processes, they present the following

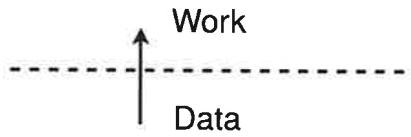

Figure 1: Illustration of the problem: work is reflected by available data

characteristics. First, they involve creativity when executing the tasks, i.e. there exists no strict process model that is followed by the developers to produce a new piece of code. Second, they are driven by methodologies, e.g., Rational Unified Process (RUP), Scrum[1], Waterfall, etc. These methodologies constitute guidelines and best practices for project success. Third, they typically make use of software tools such as Issue Tracking System (ITS), Version Control System (VCS) and Integrated Development Environment (IDE). Such tools typically record work activities into log files. Fourth, there is in general no process engine to control the execution. Rather than that, a project manager or a Scrum master has to make sure that tasks are made available to the respective resources. Fifth, it is not trivial to obtain high level information on performance measures such as the burn-down rate, which are resource bottlenecks, what are time requirements for certain tasks, what type of work is actually being done, and how productive are people working in certain tasks, etc. Hence, there is the need for algorithms and tools that help extracting this knowledge.

Software development processes fall into the category of *project-oriented business processes* [BCM+15]. That is, they are rather ad-hoc plans performed with limited resources and time, but with a clear goal: develop a software artifact. Unlike classic business processes which are best captured with notations such as Petri nets and Business Process Model and Notation (BPMN), software development processes are more akin to one-time plans, which are usually captured by models such as Gantt and PERT diagrams. The following example captures the essence of a typical software development process in practice, such as for instance in Google LLC [Hen17].

> A new software version or feature needs to be developed. Unquestionably, Google has know-how on software development. However, before running straight into the development phase, the company first makes sure to gather and properly formulate all the requirements. In a Scrum scenario these requirements would be written down as user stories. At a later stage, the project manager needs to plan the time and resources allocated to the respond to project deadlines. He can go through the list of requirements that need to be implemented and assign to an effort estimation value to each of them. With the plan done, the development phase can commence. During the development phase, resources address the task in a creative way, choosing the order of the tasks according to their own knowledge and expertise, until eventually all the tasks are terminated. Every time a resource wants to save their progress issue a so-called *commit* command which creates a new version of the modified files on the VCS. Likewise, every time a certain task (which may be worked carried out through by different commits) is completed, it is marked as done in the ITS. Both the commits and the tasks can be commented by the users, respectively to document the changes and raise a discussion for better understanding the task goals.

[1] https://www.scrum.org

Several tools are used by software project participants to support their work. Therefore, traces about the overall process are typically available under different repositories and artifacts, e.g., spreadsheets, word processor documents, programming languages files, emails, etc. This makes it cumbersome to obtain knowledge about the overall business process through manual inspection. Also, it is a challenge to automatically *extract* work information from unstructured data such as user comments when working on tasks. For instance, existing solutions (e.g., process mining algorithms) are not able to provide informative results when the data is not available as a structured single event log where activities are explicitly labeled. Likewise, other software tools (e.g. GitHub) limit themselves at providing only process-unaware statistical information (e.g., number of commits in a file).

While obtaining an overarching view on software development is challenging, there are still repositories that we can be exploited to obtain process knowledge. One tool that provides important traces of the development work is VCS. This tool is used to keep track of the different versions of files created by users and to manage their collaboration. Not only they allow to keep track of the different versions, but also allow users to associate a textual comment that describe the changes made. Therefore, a new version of a particular file is created as the result of an activity done by a user. The evolution of these versions, along with information about the users and their comments can be retrieved from these type of tools in the form of semi-structured event logs. It is then a challenge how to discover the business process from events (e.g., lines of code changed, comments, user information) present in these logs.

Figure 2 illustrates such problem scenario. Software development relies on tools like ITS (e.g., JIRA, GitHub Issues) and VCS (e.g. Subversion, Git). One the one hand, ITS is used for tracking the project management aspect. This aspect offers plan related information, such as issues, features, bugs, text, planned and executed tasks, timestamps, stories, and effort estimation (e.g., story points in JIRA). On the other hand, VCS offers information about work traces, such as versions of the artifacts, number and content of commits done by developers, artifacts evolution, users and their comments, and timestamps of each action.

Although different technologies exist in practice (e.g., Subversion, Mercurial, Git), the information contained in the logs can be roughly summarized by Table 1. The table displays an excerpt of a VCS log, i.e., a set of user commits, after having extracted and structured the data according to five attributes. The semantics of the attributes is a follows: *i) Id* is a unique identifier; *ii) Resource* is the resource that issued the commit; *iii) Date* is a timestamp associated to the time and date the commit was stored in the system; *iv) Comment* is a user comment on the changes made; *v) Diff* is low-level information on the difference between the current and the previous version, for each file. Likewise, information about issues from ITS can be extracted and presented in a tabular way. In this case, other attributes are more important. These attributes can

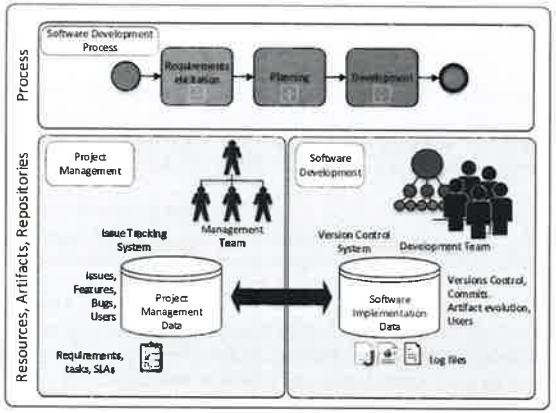

Figure 2: Software development scenario with two repositories used for project management and software development, respectively.

be, for example, the status of issues and the conversations that take place around them. Hence, studying the co-evolution of these two repositories can help extracting relevant knowledge about the software development process.

More precisely this work wants to answer to the following research question. **RQ**: *How can we make use of project event data to gather insights about the software development process that are informative to managers?* Because the project managers can benefit from a process view to better analyze hidden aspects of the software development process, this work takes a process mining stance on the problem. Therefore, the main research question is broken down into the following four subquestions. Each of them addresses **RQ** with respect to the four perspectives of a process that are useful to discover the event data [vdA16].

RQ1. Mining the time perspective. *How can we use project event data to extract information about the temporal order of the activities?* For example, an answer to this question would look like: the development activity has a duration of 2 weeks on average, the average time of task creation is 15 minutes, etc.

RQ2. Mining the case perspective. *How can we use project event data to extract information about the case perspective?* For example, an answer to this question would look like: all the bugs are solved in a 3 steps iteration, or a quality piece of code takes a conversation with 3 people and is successfully merged into the main branch after 1 week, etc.

RQ3. Mining the organizational perspective. *How can we use project event data to extract information about the organizational perspective?* For example, an answer to this question would look like: the software development is carried out by a team of 4 people, the actual user roles in the company are developer and tester, etc.

Table 1: An excerpt of a VCS log data

Id	Resource	Date	Comment	Diff
1	John	2017-01-31 12:16:30	Create readme file	diff –git a/README.md b/README.md @@ -0,0 +1 @@ +# StoryMiningSoftwareRepositories
2	Mary	2017-02-01 10:13:51	Add a license	diff –git a/README b/README @@ -1,0 +2,3 @@ +The MIT License (MIT) + +Copyright (c) 2015 Mary+
3	Paul	2017-02-02 16:10:22	Updated the requirements.	diff –git a/README.md b/README.md @@ -1,4 +1,5 @@ + # string 1, string 2, string 3 diff –git a/requirements.txt b/requirements.txt @@ -0,0 +1 @@ +The software must solve the problems
4	Paul	2017-02-02 15:00:02	Implement new requirements	diff –git a/model.java b/model.java @@ -1,9 +1,10 @@ +public static methodA(){int newVal=0; @@ -21,10 +23,11 @@ + "1/0","0/0", diff –git a/test.java b/test.java @@ -0,0 +1,2 @@ +//test method A +testMethodA()

RQ4. Mining the control-flow perspective. *How can we use project event data to extract information about the control-flow perspective?* For example, an answer to this question would look like: the testing is always done before development, or while new features are worked on, also new requirements are created, etc.

3 State of the Art

This problem has been partially addresses in three research areas: *i)* Process Mining (PM); *ii)* Mining Software Repositories (MSR); and *iii)* Text Mining (TM). The following sections introduce these fields along with their contribution to the research question and limitations.

3.1 Process Mining

3.1.1 Description of the field

what it generally does

Process mining is a discipline that emerged in the last decade. The goal of this discipline is to provide fact-based insights and support process improvement. On a broader context, process mining can be considered as the missing link between traditional model-based process analysis and data driven techniques such as data mining and machine learning [vdA16]. Compared to existing Business Intelligence (BI) technologies, process mining techniques go beyond the calculation of simple Key Performance Indicators (KPIs). Instead, by building on model-driven approaches, they provide means to gain transparency on several aspects of the end-to-end business process. More specifically process mining techniques can infer models from event logs, which inform about the diverse aspects of a business process. Main aspects or *perspectives* of a business process are the following.

- The *time perspective* aims at analyzing time and frequency of process events. The focus is to use time information for performance analysis, such as bottlenecks, resource utilization, prediction of workload, remaining running time of process instances, etc.
- The *case perspective* aims at identifying properties of process cases. A process case is one execution of a process from start to end, e.g. from the moment a customer order a product to its delivery. However, there can be other ways to determine cases, e.g., the evolution of a data artifact, such a for instance the fulfillment of monthly report of the orders.
- The *organizational perspective* aims at analyzing the event log to gain transparency on the resources involved in the process. The focus of this perspective is to help understanding which are the people and systems in the process and how are they related in terms of roles, hierarchy, handover of work, privileges, access-control, social network, etc.
- The *control-flow perspective* aims at analyzing the different variations of the process, i.e., in which order its constituting activities are carried out in real life. The focus of this perspective is on understanding the different process variants by the help of process models expressed as Petri nets, BPMN, event-driven process chain (EPC), Unified Modeling Language (UML), etc.

Figure 3 illustrates process mining research and how it relates event data from real world and business process models. There are three types of process mining, namely *i)* process discovery; *ii)* conformance checking; and *iii)* enhancement.

Process discovery. This type of process mining is concerned with the inference of process models from event logs. Process discovery algorithms are typically unsupervised techniques that produce models in various notations such as Petri nets, BPMN, EPC, etc. A example of a process discovery algorithm is the so-called α-algorithm [vdAWM04].

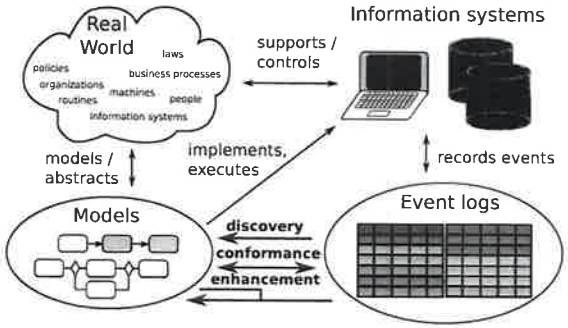

Figure 3: The process mining framework Adapted from [vdA16].

Conformance checking. This type of process mining compares the model and the log of the same process. The goal is to verify if the reality as recorded in the event log corresponds to the plan as defined by the model. Possible deviations are quantified making it possible to obtain important cues about bad performance in case the model is prescriptive or noncompliance in case the model is normative.

Enhancement. This type for process mining is focused on improving the existing process model by using information from past executions recorded in the log. Differently from conformance, the goal is to go beyond measuring the misalignment but actually correcting the process model. Main types of corrections are *repair*, where a process model is repaired to better explain the data from the log, and *extension*, where a new information is added to the process model that is found in the log, e.g., labeling activities with the resource names, labeling sequence flows with durations, and so on.

Process mining is a mature field with considerable number of algorithms from academia and a many industry tools such as Celonis[2], Disco[3], minit[4], LANA Process Mining[5] and many more. Mining algorithms are developed every year to deal to mine not only process workflow, but also other perspectives, i.e. organizational, data, etc. Nevertheless, process mining algorithms work with structured data [vDdMV+05, VBVV11].

3.1.2 Contribution to the research questions

how they have contributed to the questions

[2] https://www.celonis.com
[3] https://fluxicon.com/disco
[4] https://www.minit.io
[5] https://lana-labs.com/en

Several works in the area of PM can help to tackle the problem by transforming it into a PM problem. These works enrich VCS log data with case and activity information, and then use process mining to discover a model. In this category, Kindler et al. [KRS06a, KRS06b] can discover a Petri net from a structured and enriched version control log. This approach was further improved by Rubin et al. [RGV+07] and a ProM[6] plug-in was provided. Poncin et al. [PSvdB11] provide the FRASR framework for preprocessing software repositories such that they can be used in ProM. Song et al. [SvdA07] help addressing the time perspective by mining a dotted chart. Process mining techniques are related to all the research questions of this thesis. Especially, they help with addressing **RQ4**, i.e. finding the control flow of software development activities.

3.1.3 Limitations

what they are not able to offer

While providing interesting insights, these contributions leave out many important aspects of software development projects, such as trying to understand whether the process was done according to the organization plan. Especially, they are limited to either simply display one perspective of the process [SvdA07] or transform the events from software repositories into the standard eXtensible Event Stream (XES) format which can be mined by tools like ProM. In this case, existing methods only take into account high level information (such as the type of file) to label events accordingly. Textual information is also taken into account [RGV+07] but often limited to use of a dictionary for identifying keywords. Moreover, fine granular information on the amount of change and the comments of commit messages have not been exploited enough by existing literature.

This works aims to use the strengths of process mining techniques on discovering informative models. At the same time, it aims to go beyond the current limitations of these techniques to handle fine granular (e.g., amount of changes) and unstructured data (e.g. user comments).

3.2 Text Mining

3.2.1 Description of the field

Text mining refers to the process of deriving information from natural language text. It relates to data mining in the aspect that both strive to extract meaningful information from raw data. However, data mining is characterized as the extraction of implicit, previously unknown, and potentially useful information from data, whereas with text mining the information to be extracted is clearly and explicitly stated in the text [Wit04]. Text mining can be regarded as going beyond information access to further help users analyze and digest information and facilitate decision making. There are also many applications of text mining where the primary goal is to analyze and discover any interesting patterns, including trends and outliers in text data. Although text mining is mostly about Natural Language Processing (NLP) [JM14], it

[6] http://www.promtools.org

embraces also applications that go beyond. For example, it analyzes linkage structures such as the citations in the academic literature and hyperlinks in the Web literature, both useful sources of information that lie outside the traditional domain of NLP. The main types of text mining are the following.

Information extraction. Information extraction is used to refer to the task of filling in templates from natural language text. The goal is to extract from the documents (which may be in a variety of languages) salient facts about prespecified types of events, entities or relationships. These facts are then usually entered automatically into a database, which may then be used to analyze the data for trends, to give a natural language summary, or simply to serve for on-line access. Traditional information extraction techniques [CL96, Moo99] leverage on rule-based systems that match predefined linguistic patterns. More recently, work on named entity recognition uses statistical machine learning methods [SMR99]. A tool that uses unsupervised learning can be found in [BCS+07].

Topic detection and tracking. Topic Detection and Tracking (TDT) was a DARPA-sponsored initiative to investigate on finding and following new events in a stream of broadcast news stories. The TDT problem consists of three major tasks: (1) *segmenting* a stream of data, especially recognized speech, into distinct stories; (2) *identifying* those news stories that are the first to discuss a new event occurring in the news; and (3) given a small number of sample news stories about an event, *finding* all *following* stories in the stream. The work of [All02] has formally defined this problem and proposed the initial set of algorithms for the task. Main subtasks of TDT, as identified in [Way00] are (i) finding topically homogeneous regions (segmentation); (ii) finding additional stories about a given topic (tracking); (iii) detecting and threading together new topics (detection); (iv) Detecting new topics (first story detection); and (v) Deciding whether stories are on the same topic (linking). An example of a real-world TDT system is Google Alerts[7].

Summarization. Summarization is the task of reducing the content obtained from text documents, still keeping a brief overview on a topic that they treat. Summarization techniques generally fall into two categories [Agg15]. In extractive summarization, a summary consists of information units extracted from the original text; in contrast, in abstractive summarization, a summary may contain "synthesized" information units that may not necessarily occur in the text document. An automatic summarization process can be divided into three steps [GL09]: (1) the *preprocessing* step where a structured representation of the original text is obtained; (2) the *processing* step where an algorithm must transform the text structure into a summary structure; and (3) the *generation* step where the final summary is obtained from the summary structure. A plethora of text summarization tools can be found online. A few examples are the open source libraries such Open Text

[7] https://www.google.com/alerts

Summarizer[8], Sumplify[9] and Online summarize tool[10].

Categorization. Categorization aims at identifying the main themes of a document. This translates into assigning natural language documents to predefined categories according to their content [Seb02]. Categorization often relies on a thesaurus for which topics are predefined, and relationships are identified by looking for broad terms, narrower terms, synonyms, and related terms. Support Vector Machines (SVMs) are used to automatically learn text classifiers from examples [Joa98]. Categorization tools usually rank documents according to how much of their content fits in a particular topic.

Clustering. Clustering is a technique used to group similar documents, but it differs from categorization in that documents are clustered on the fly instead of through predefined topics. Documents can also appear in multiple subtopics, ensuring that useful documents are not omitted from the search results. A basic clustering algorithm creates a vector of topics for each document and measures the weights of how the document fits into each cluster. A survey of clustering algorithms can be found in [Agg15].

Concept Linkage. Concept-linkage tools connect related documents by identifying their shared concepts, helping users find information they perhaps would not have found through traditional search methods [GL09]. It promotes browsing for information rather than searching for it. For example, a text mining software solution may easily identify a transitive closure in a set of topics {X, Y, Z}, i.e., a link between X and Y, a link between Y and Z, and a link between X and Z. With large sets of data, such a relationship could be disregarded by humans.

Information Visualization. Information visualization aims at visualizing large textual sources in such a way that the content can be displayed in a hierarchy or map and provides browsing features, in addition to simple search. For instance, governments or police can identify terrorist networks in a map and identify crimes that were previously unconnected [GL09].

Question Answering. Another application area of developed text-mining technologies, along with the natural language processing, is natural language features Question Answering (QA). QA is concerned with building systems that automatically answer questions posed by humans in a natural language. One of the first Query Answering tools was START[11], developed in the work of [Kat97]. Question answering has been extensively used in Biomedical domain for aiding researchers and health care professionals in managing the continuous growth of information.

Association Rule Mining. The focus of association rules mining is to study the relationships and implications among topics, or descriptive concepts, that are used to characterize

[8] https://www.splitbrain.org/services/ots
[9] http://sumplify.com/
[10] http://www.tools4noobs.com/summarize/
[11] http://start.csail.mit.edu/index.php

a corpus. The work of [ASO94] shows how it is possible to discover association rules from massive data from databases, referred to as *basket* data. The same approach can be followed by constructing a database of rules using information extraction methods and subsequently applying techniques, e.g. [HZY+10], to uncover hidden associations in the database. For instance, a rule might be that 98% of customers that purchase tires and auto accessories also get automotive services done. This suggests that association rules can be captured as if/then patterns. Criteria such as support and confidence [AIS93] are used to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements has been found to be true.

3.2.2 Contribution to the research questions

My research involves unstructured data in the form of word processor documents, emails, user forums, and commit messages from VCS. Text mining algorithms can be combined to quantitative data to gather information about project work. For example, topics models can be combined to time-clustered messages from pull request comments. In this way it is possible order to discover "hot topics" (e.g. deliverable, milestone, meeting, etc) during project development. More in general, text mining techniques help with preprocessing the data and retrieving relevant information. Successively, this information can be brought together to create a structured event log in the XES format. Therefore, text mining research serves and a preprocessing technique to facilitate process extraction. Especially, it helps as follows. *i)* **RQ2** – extracting case characteristics from conversations in user forums; *ii)* **RQ3** – extracting resource roles from user comments in commit messages; *iii)* **RQ4** – extracting activities out of coordination message exchanges.

3.2.3 Limitations

Text mining approaches focus on obtaining structured information from unstructured textual data, and mainly uses NLP [WBMT99]. Works that use NLP can be found in both PM [vdA16] and MSR [CTH16]. In the Business Process Management (BPM) [DLMR13] area, NLP techniques have been used to understand process activities [Leo13, MLP14] and analyze software processes under a knowledge-intensive perspective [dSB11, RdBS17]. Likewise, in the MSR area, NLP has been used as an information extraction tool to obtain informative metrics from a software engineering perspective [THB14, CTH16].

3.3 Mining Software Repositories

3.3.1 Description of the field

Mining software repositories (MSR) research has emerged in the last decade from the software engineering field. It focuses on analyzing patterns and trends in software. Main topics in MSR are about software quality and metrics, visual software analytics, bug analysis, communication among project members, defect tracking, predictions about software development, and software

evolution. Works in MSR often use techniques from data mining, text mining and statistics. Typically, these methods focus on analyzing dependencies in the structure of the software artifacts. There are some existing works that can be taken into account as a starting point for understanding the development process. These works focus principally on the users and the artifacts, mining co-evolution or co-change of project parts [ZRDvD08, DLL09] and network analysis of file dependency graph based on commit distance [ZN08, ABDZ09, YSX13]. Hidden work dependencies are mentioned as *logical dependencies* [OSGdS11]. Also techniques for trend analysis [RHL15] and inter-dependencies between developers [LBGL16] are proposed. All these techniques allows to obtain several metrics and models from software repository data.

3.3.2 Contribution to the research questions

MSR focuses on software engineering aspects, like code quality metrics, modularization, code complexity, user networks, and other important metric. In general, methods from MSR provide useful dependency analyses of the repository structure. Contributions in this area focus on the users, the artifacts and the repository evolution [ZRDvD08, DLL09], and network analysis of file dependency graph based on commit distance [ABDZ09, WSX13]. Also techniques for trend analysis [RHL15] and inter-dependencies between developers [LBGL16] are proposed.

All these techniques inform this research on relevant metrics about software development. MSR works can be used as state of the art measures that are interesting to be discovered from event data. Moreover, the aim is to include these measures for enriching event logs. As a consequence, the manager is enabled to have additional information on the development process. Approaches from MSR help with the software engineering part of this thesis. They are relevant to the following research questions. *i)* **RQ1** – through gathering times and durations of development activities; *ii)* **RQ2** – through helping the extraction of artifact evolution; *iii)* **RQ3** – though helping extraction of social networks of users.

3.3.3 Limitations

Limitations of MSR works derive from its software engineering orientation. In fact, none of these works aims at extracting knowledge about the actual process. Extraction is typically limited to engineering metrics and rarely goes beyond mining the bug lifecycle [PSvdB11] when it comes to process discovery. This works assumes that both the communities of MSR and PM could benefit from research on mining process models out of software development activities. The former would gain better perspectives and easier understanding of the engineering activities. The latter would extend its range towards a new domain. In this sense, this thesis serves as an attempt to bridge the above-mentioned fields.

3.4 Summary

This research combines ideas from the above mentioned areas to devise algorithms for mining the software development process. This section defines the contribution of the related fields to

the four research question. Table 2 lists summarizes the most relevant works in the literature and classifies them in relation the addressed research questions.

Table 2: Classification of existing literature addressing the various aspects of mining the development process

	R1	R2	R3	R4
PM	Dotted Chart [SvdA07]	Decision mining [RvdA06]	Visualization techniques [BS13] Organizational mining [SvdA08] [SCJM15]	Bug fixing [PSvdB11], Workflow fragments [KRS06a, KRS06b]
TM	Information extraction [CL96]	Topic models [CTH16] Theory-generating case studies [LBGL16]	Social network [BGD+06] Survey [BKZ10] [dASB10]	Speech acts [DM13] [CRBS18] Natural language processing [FMP11]
MSR	Time series [RHL15] [HMCX14], Statistical analyses [OSGdS11]	Network analysis [DLL09], [ZN08], Language Models [AS13]	Email analysis [BGD+06], Role identification [YR07]	Exploratory studies [GPvD14]

4 Research Design and Methodology

This section presents the research method. It also describes the dataset and outlines the expected research outcome.

4.1 Research Method

Information systems research is an interdisciplinary field of study that uses theories from social sciences, economics, and computer science. The field can be divided into two complementary paradigms: *behavioral science* and *design science*. Behavioral science aims at developing and justifying theories in order to explain or predict information systems phenomena [Gre06]. Design science focuses on the creation and evaluation of innovative design *artifacts* [HMPR04]. Figure 4 illustrates the design science approach as a process [PTRC08].

This proposal foresees the development of four different artifacts. These artifacts are interconnected to one another in that each of them tackles a different perspective of the overarching software development process. More specifically, each artifact is devised following the Design

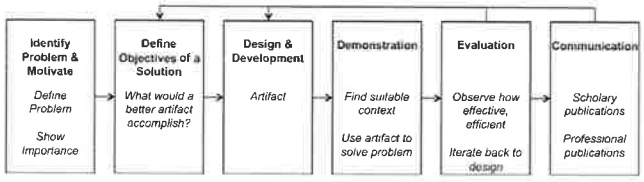

Figure 4: The Design Science Research process, adapted from [PTRC08]

Science Research (DSR) approach. These artifacts are useful useful to inform the mentioned approach to theory building [BSS18].

Research rigor is ensured by evaluating the artifacts with the Framework for Evaluation in Design Science Research (FEDS) [VPHB16]. FEDS provides strategies for evaluating DSR artifacts. More specifically, it takes into account two dimensions *i)* the functional purpose of the evaluation (formative or summative); and *ii)* the paradigm of the evaluation (artificial or naturalistic). Moreover, it provides four steps for choosing an appropriate evaluation strategy *i)* explicate the goals of the evaluation; *ii)* choose the evaluation strategy or strategies *iii)* determine the properties to evaluate; and *iv)* design the individual evaluation episode(s).

Next, the evaluation for each designed artifact is provided.

Artifact A1.

Structure is good. Make it text instead of bullets

Design objective: has to be applicable, usable, simple and provide overview as well as detailed view

Features:

- Time information
- Duration of activities
- Structure of the project
- Roll-up and drill-down on granularity of events

Evaluation goals:

- Rigor: assured by the design science approach
- Uncertainty and risk reduction: artifacts is technically infeasible, data is unavailable
- Ethics: "big-brother is watching you" effect on people
- Efficiency: find all time patterns, algorithm runs in a reasonably short time

Evaluation strategy: Quick & Simple, i.e., the technique is directly evaluated with real data

Iteration episodes:

1. Initial design of Gantt chart (formative)
2. Evaluation of successful identification events, their duration and project structure
3. Run of the tool on real projects and provide a visualization (summative)

Artifact A2.

Design objective: has to be applicable, simple and provide comparison of different cases

Features:

- Difference between work instances
- Measure of work: amount of changes
- Handle unstructured data from comments
- Evolutionary analysis of files
- Uncover work dependencies

Evaluation goals:

- Rigor: assured by the iterations
- Uncertainty and risk reduction: technically infeasibility risk is addressed through development by smaller iterations; data unavailability risk is addressed by local replication of data; simplicity of results is addressed by exploiting user comments and derive informative process labels
- Ethics: data is already public on GitHub
- Efficiency: find all work dependencies in a reasonable time

Evaluation strategy: Technical risk & efficacy: the technique is firstly evaluated with a toy example, then revised and applied to a large number of software development projects

Iteration episodes:

1. Initial design of artifact and results visualizations(formative)
2. Revision and choice of final visualization
3. Evaluation of efficacy on retrieving features
4. Run the technique on toy example (artificial evaluation). Validate efficacy and usefulness. Revise artifact.
5. Run the technique on real data sets from GitHub projects (summative and naturalistic evaluation)

Artifact A3.

Design objective: has to applicable and provide correct classification of roles

Features:

- determine roles of users
- handle user comments
- automatically classify resources

Evaluation goals:

- Rigor: assured by the iterations
- Uncertainty and risk reduction: incorrectly resource classification risk is reduced by using different training set data from industry partners and obtaining feedback
- Ethics: there is a risk of obtaining information about people's work. This is mitigated by NDAs signed by the parties involved.
- Efficiency: evaluated in terms of precision and recall

Evaluation strategy: Quick & simple: the technique is evaluated with real data from industry partners

Iteration episodes:

1. Initial design of the artifact voted to a simple and structured representation of the data (formative)
2. Exploration and selection of the best features and classifiers.
3. Evaluation of results with real data from partners and feedback (summative + naturalistic)

Artifact A4.

Design objective: has to applicable to pull requests and provide informative and reliable process models about specific work patterns

Features:

- handle comments from forum conversations
- discover a process model

Evaluation goals:

- Rigor: assured by the iterations
- Uncertainty and risk reduction: risks are: 1) activities of business processes are not mapped correctly – mitigated by manual annotation 2) the resulting model is not informative – mitigated by statistical techniques voted to cluster traces into significant groups
- Ethics: n/a. Only data from open source repositories will be used
- Efficiency: the artifact can deal with complex projects quickly

Evaluation strategy: Technical risk and efficacy: the technique is firstly evaluated with an initial well-known dataset, then revised and evaluated with other projects

Iteration episodes:

1. Initial exploratory analysis on the applicability of process mining techniques on pull requests (formative)
2. Evaluation of process mining methods and assessment of statistical significance of results
3. Mine process models that are statistically significant and analyze the idea generation patterns (summative + naturalistic)

4.2 Data Collection

This thesis includes the collection of and generation of datasets from real life software projects. These project data were collected both from industrial partners and from Open-source Software (OSS). Table 3 summarizes the available data. Logs from industrial partner are concerned with specific development activities (e.g., building railway interlocking-system software) and include a sufficient number of events that cover one software release. Logs from OSS were manually extracted by GitHub repositories. A tool has then been devised to parse these data and use them to populate a database. An original dataset is represented by Github pull requests. This dataset contains a list of manually annotated pull requests using the coding scheme by [MM16]. An additional output of this thesis will be the creation of a larger dataset using a machine learning approach that is able to automatically categorize pull requests in the classes given by [MM16].

4.3 Expected Outcome

In addition to an original dataset, this doctoral thesis is expected to produce outcomes that help the project manager analyze his software development processes. To this end new approaches in the form of artifacts [PTRC08] will be devised. These artifacts will serve as proofs-of-concept for the applicability of the research. In particular, the artifacts must address each of the four process aspects. Therefore, the outcome is categorized according to the research questions derived in Section 2 as follows.

Table 3: Available dataset

Log	Description
SHAPE	Industry VCS from software development in the railway domain
Github	Logs from real world OSS development
Github pull requests	Manually annotated pull requests from one real life open source project
Jira	Log data from ITS from industry partner
Asana	Generated dataset from project management tool of industrial partner
Kibana	Event traces from distributed process-agnostic enviroment used to handle communication among several processes

- **Artifact A1: Mining the Time Perspective (RQ1).** Novel technique that allows for gaining transparency on the time perspective of software development work. For instance, and artifact that explicits what are the durations of tasks and when are actual tasks executed.
- **Artifact A2: Mining the Case Perspective (RQ2).** Novel technique that allows for gaining transparency on the case perspective of software development work. For instance, an artifact that explicits how the different cases are handled and whether there is any dependency of work that might influence the case execution.
- **Artifact A3: Mining the Organizational Perspective (RQ3).** Novel technique that allows for gaining transparency on the organizations aspect of software development work. For instance, an artifact that explicits which are the actual roles software developers are covering and which is the organizational network.
- **Artifact A4: Mining the Control-Flow Perspective (RQ4).** Novel technique that allows for gaining transparency on the control-flow perspective of software development work. For instance, an artifact that allows for abstracting from data which is order of activities that are executed to accomplish a certain task or goal in software development.

Figure 5 shows the overall information coming from combining the four perspectives into a Gantt chart that is more informative to managers.

5 Preliminary Results

Next, I present efforts done towards addressing the requirements presented in Section 2.

5.1 A1. Mining the Time Perspective (RQ1)

Mining the time perspective from a software repository is defined as extracting temporal process knowledge. This problem relates to monitoring whether the process was executed respecting a

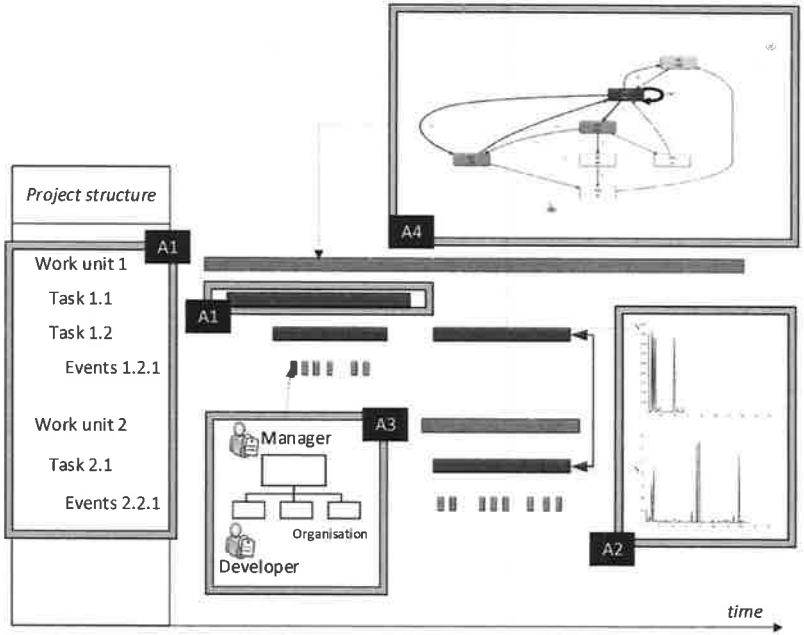

Figure 5: The empirical Gantt chart of software development. The four process perspectives combined.

predefined plan. Typically the process involves various actors which track their work through the use of VCS. Often, the plans are not represented in standard process notation. Rather, Gantt or PERT charts are used.

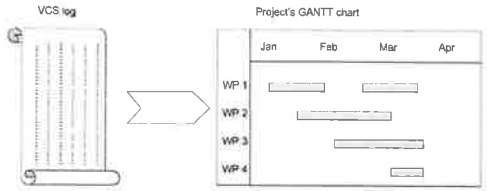

Figure 6: Mining the time perspective of software development into a Gantt chart

Challenges are *i) time approximation* (i.e., when the activity really started, compared to when it was registered in the log); *ii) granularity* (i.e., be able to switch from a detailed view of the single events and a coarse-grained view of the overall project); and *iii) coverage* (i.e., how much work effort was put within the duration of the activity). Figure 6 depicts the idea of mining a Gantt chart from VCS logs. The output is presented in a way that is informative to project managers. Details about the technique can be found in [BCM+15].

5.2 Mining the Case Perspective (RQ2)

The identification of process cases is not an easy task when it comes to software development data. However, a highly informative case candidate can be considered the data artifact itself. According to [vdA16], cases can also be characterized by the values of data elements. In line with this, it is possible to devise techniques that study the *artifact evolution*. Although it can be input to process mining techniques, this evolution can also be analyzed as a time series. An example of such evolution is given in Figure 7, which shows the lines of code (LOC) changes over time.

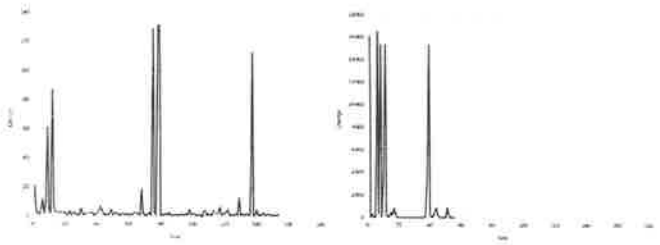

Figure 7: Evolution of files in a version control systems: LOC changes over time

Challenges related to the artifact evolution pertain *prediction* of plateaus and *pattern recognition*. The results reflect work patterns over the artifact. For example, when a data file is not being modified anymore, its time series has a plateau and it can be interpreted that the document is now ready for release. A relevant problem in software development is bad modularization. Especially, the dependency of two artifacts on one another is considered a bad practice. Two time series can be compared together and file dependencies can be found that reflect work coupling. This is equivalent to finding dependence between two processes which might have been designed for different purposes.

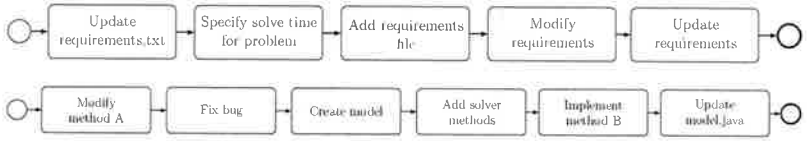

Figure 8: Processes of two work-dependent files. They may seem different but they co-evolve in time and space, i.e., they describe the same *case*.

We have devised a proof-of-concept in [BRd+17]. Figure 8 shows a possible outcome of the technique. The result is obtained by identifying couples of files with similar evolution and mining a business process from the comments. As the result is aggregated, several comments have been combined together and mined as a story. Among other things, the technique allows to profile existing software development projects.

5.3 Mining the Organizational Perspective (RQ3)

Software development projects are knowledge intensive, thus involve creativity and flexibility. Project participants are free to tackle development tasks according to their expertise and ideas to solve ad-hoc problems. De facto, members may behave according several roles in the organization. Therefore, the discovery of *de facto* roles within a software project may give important insights on the actual project. Role discovery can help with monitoring existing projects in two ways. First, emerging roles can be analyzed in order to have a better skills profiling of the resources. Second, emerging roles can be used to check whether resources are breaking contractual agreements or norms.

We have developed an approach for tackling the problem of mining the organizational perspective in software development projects. The approach provides information about the actors involved in the business process and their relations. In substance, it provides automatic classification of resources into company roles (e.g., developer, tester, etc) based on the comments of project participants. It works on VCS logs and provides information about the people, their roles, other systems involved, the organization hierarchy, the social network, and resource profiling. Figure 9 shows an example of resource profiles automatically inferred from VCS

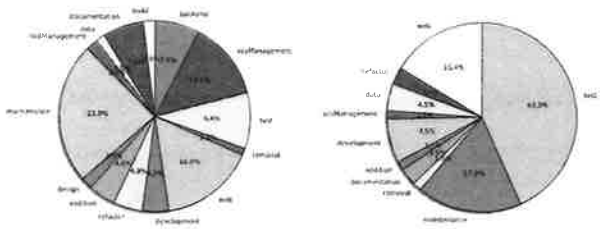

Figure 9: Profiles of a developer and a tester, from left to right.

comments.

Challenges of using NLP concern the *jargon* used by software developers. They often use technical terms, very short phrases, or a number of codes and hyperlinks. These makes NLP techniques score low results even on simple tasks like sentence parsing, or name-entity recognition. Therefore, approaches should take into account several factors when mining for the organizational perspective. The result allows the manager to have measurable information about the resources. For example, do two people work better together and how can we build the best team? Details of the approach to extract resource profiles can be found in [AAT+16].

6 Next Steps and Finalization of the Thesis

6.1 Mining the Control-Flow Perspective (RQ4)

Knowledge of the control flow of a software development process can be extracted from VCS and ITS. One important mechanism widely used in software development for collaboration are *pull requests*. A pull request is issued every time a developer wants to contribute to the main source with a new piece of code. Pull requests trigger communication among different people and may help shedding light into problems, learn new things and obtain new ideas on the existing software. It is interesting for project managers and developers to better understand factors and behavior that makes a pull request successful.

In ongoing work, we are investigating the relation of the pull request process and surrounding factors such as the generation of new ideas or change of behavior. We have obtained a data set of pull requests spanning over two years from an real world repository. User conversations have been manually annotated with codes that classify them into categories of comments. We are current able to discriminate whether a comment is a new idea, an assumption, a merge that closes a pull request, etc. Considering the pull request identifiers as cases and the codes as activities, we are able to extract the process of the pull requests. It is interesting to compare how the conversation (i.e., behavioral perspective) unfolds before and after a pull request is merged. We plan to explain these differences by also taking into account further information from text

such as emotions.

The challenge of mining the control-flow usually pertains the completeness of data and the case and activity identification. While PM contribution have been focusing mostly on the control-flow perspective, techniques from MSR hardly go beyond mining the bug lifecycle [Akb18]. The nature of software development process makes it difficult to recognize recurrent activities when only VCS are analyzed.

6.2 Finalization of the Thesis

The expected result for the above-mentioned work is an artifact that is able to explicitly show what are process patterns that lead to the generation of an idea. This will serve as a proof-of-concept that event data from projects can be used to increase understanding of process flow perspective. The overall thesis will be finalized by evaluating the results both against the real world dataset and project managers. There is ongoing work in which a questionnaire has been answered by industry partners. Questions pertains activities, tools and habits within a software development company. These information can be used to evaluate the artifacts developed by this thesis.

7 Dissertation Relevance and Publication Plan

7.1 Implications for Research

This proposal presents research on mining the software development process. Approaches towards discovering the perspectives of time, resources and cases have been developed and the results suggest that obtaining process knowledge from software repositories brings new useful insights for the project managers. Ongoing work aims to complete the research by tackling the problem of discovering the flow-perspective of the software development process. Usefulness will be evaluated through user studies. This research extends the process mining field towards its adoption on software repositories. Project managers would benefit from this research by having a process perspective on their ongoing projects through visual models and diagrams, thus overcoming existing flat approaches based on simple indicators such as burndown charts or change plots offered by existing tools.

Change?!

7.2 Work Plan and Dissemination

Figure 10 summarizes the work plan based on the artifacts to be build for addressing the four research questions. In other words, the thesis work is currently in its final stage, which involves the creation of artifact **A4** to address the corresponding research question **RQ4**.

Target venues for communicating the results are BPM and software engineering journals and conferences. So far, the following publications have been achieved that directly address the research questions concerning time, organization, case and control-flow, as defined in Section 2.