Saimir Bala

Mining Software Development Projects

Using Trace Data to gather Insights on Time, Case, Organizational and Control-Flow Perspectives of Software Processes

January 26, 2022

Abstract

Software development is a highly flexible endeavor. Yet, software development projects have specific requirements about quality, costs and time to deliver. Therefore, there is need for monitoring and controlling the software development process. This problem is tackled separately by the disciplines of software engineering and business process management. In the software engineering area, especially under the mining software repositories conference series, the focus is to develop new techniques and algorithms that exploit data science, artificial intelligence and machine learning to gather new information that helps to improve software engineering practices. In the business process management area, especially through process mining techniques, the focus is to gather insights about the exiting process of an organization with the goal of improving such process. While both disciplines focus on improvement, there are currently not many works that exploit this synergy. On the one hand, software engineering rarely adopts a process point of view. Hence, they still fail at delivering knowledge about software development that can be easily understood by managers. On the other hand, process mining approaches from business process management, cannot be readily applied to trace data from software repositories. Therefore, they still fail at including trace data about these types of processes.

This dissertation develops novel methods and algorithms that can be used to extract process knowledge from software development trace data. This knowledge is provided under four perspectives, namely time, case, organizational, and control-flow. Equipped with such multi-perspective knowledge of the process, a manager can obtain factual insights into projects. This dissertation can be positioned as a bridge between the process mining discipline and the area of software engineering. From a research point of view, the outcomes of this dissertation are a building block to further cross-fertilize research streams on process mining and software engineering. From a practical point of view, the outcomes of this dissertation provides familiar means of analysis on software development to the project manager who wants to gather insights on the underlying process.

Contents

List of Figures

List of Tables

Acronyms

List of abbreviations used in this thesis.

BI	Business Intelligence
BPM	Business Process Management
BPMN	Business Process Model and Notation
BPMS	Business Process Management System
DBMS	Database Management System
DSR	Design Science Research
DT	Decision Tree
ER	Entity-Relationship
ETL	Extract, Transform and Load
FEDS	Framework for Evaluation in Design Science Research
GT	Grounded Theory
GUI	Graphical User Interface
IDE	Integrated Development Environment
ITS	Issue Tracking System
KPI	Key Performance Indicator
LCA	Least Common Ancestor
LOC	Lines of Code
MSR	Mining Software Repositories
NAP	Number of Authors in Project
NLP	Natural Language Processing
NTP	Number of Types of work in Project
OSS	Open-source Software
PAIS	Process-Aware Information Systems
PIS	Specialization of Author in each Activity Type
PM	Process mining
PW	Project Workload
PWS	Specialization of Project Workload
RAM	Reliability, Availability and Maintainability
RBAC	Role-Based Access Control

RPIS Specialization of Relative Author in each Activity Type
RPWS Specialization of Relative Project Workload
RUP Rational Unified Process
SCM Software Configuration Management
SDM Software Development Methodology
SVN Subversion
TW Type of change Workload
UTI User Activity Involvement
UTW User Activity Workload
VCS Version Control System
XES eXtensible Event Stream
YAWL Yet Another Workflow Language

Chapter 1
Introduction

This chapter provides an introduction to this doctoral thesis. Section 1.1 motivates the research. Section 1.2 defines the problem. Section 1.3 gives an overview of the research methodology and defines the artifacts generated as outcome. Section 1.4 briefly discusses the data created and used in this thesis. Section 1.5 classifies the contributions of this thesis. Section 1.6 provides an outlook of the structure, and Section 1.7 lists the publications that have resulted from work on this thesis.

1.1 Motivation

Software development is a process that involves creativity (Aldave et al., 2019; Dingsøyr et al., 2012). 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 for 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 the a major interest. Contributions from this area 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 perspectives of the development process. More specifically we categorize them into *time*, *case*, *organizational* and *control-flow*. The time perspective informs about temporal aspects of the process, such as when did an activity happen and for how long. The case perspective informs about characteristic of the different cases, such as the number of lines of code being produced at during the creation of each version of an artifact. The organizational perspective informs about the roles of the people in the context of the organization, such as developers, testers, etc. The control-flow perspective informs about the logical succession of activities, such as an artifact must first be implemented and then tested.

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.

1.2 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 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* (Bala et al., 2015). That is, they are rather ad-hoc plans performed with

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

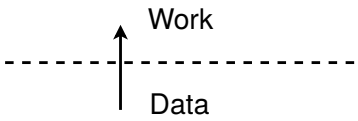


Fig. 1.1: Automatically derive work from trace data

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 (Henderson, 2017).

> A new software version or feature needs to be developed. Unquestionably, Google has know-how on software development. However, before starting the implementation phase, precise requirements for tasks must be formulated. These tasks are stored in ITS and referred to as *issues*. Usually effort estimation values are assigned to each of them. Subsequently time and resources are allocated to the tasks. This starts the implementation phase. During this phase, resources work on tasks in a creative way, choosing among available tasks according to their own skills and expertise. In order to save the progress, developers 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 carried out through by different commits) is completed, the corresponding issue is marked as done. Both commits and issues can be commented by the users, respectively to document the changes and raise a discussion for increasing understanding the problem.

Several tools are used by software project participants to support their work. Therefore, traces about the overall process are typically available in 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 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 supports it keeping track of file versions, but also allows 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.

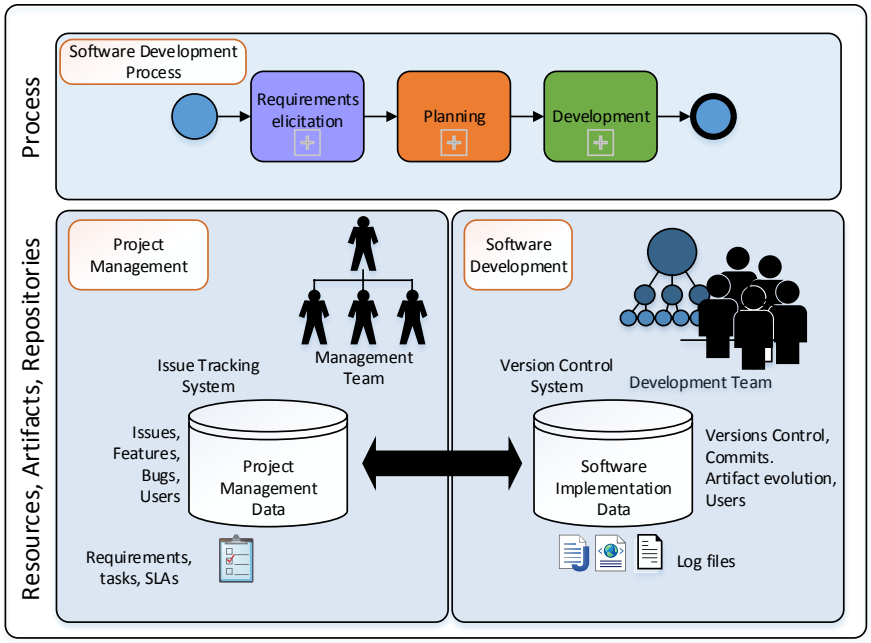


Fig. 1.2: Software development scenario with two repositories used for project management and software development, respectively.

Figure 1.2 illustrates such problem scenario. Software development relies on tools like ITS (e.g., JIRA, GitHub Issues) and VCS (e.g. Subversion, Git). 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). 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.

Maybe I can change this table with a more generic one that contains all information?

Although different technologies exist in practice (e.g., Subversion, Mercurial, Git), the information contained in the logs can be roughly summarized by Table 4.1. 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

Table 1.1: 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()

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 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 seeks 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 (van der Aalst, 2016).

RQ1. *How can we use project event data to extract information about the **temporal** perspective of activities?*

RQ2. *How can we use project event data to extract information about the **case** perspective?*

RQ3. *How can we use project event data to extract information about the* ***organizational*** *perspective?*

RQ4. *How can we use project event data to extract information about the* ***control-flow*** *perspective?*

1.3 Research Methodology and Generated Artifacts

This thesis is positioned as design science (Hevner et al., 2004). 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 (Gregor, 2006). Design science focuses on the creation and evaluation of innovative design *artifacts* (Hevner et al., 2004). Figure 1.3 illustrates the design science approach as a process (Peffers et al., 2008).

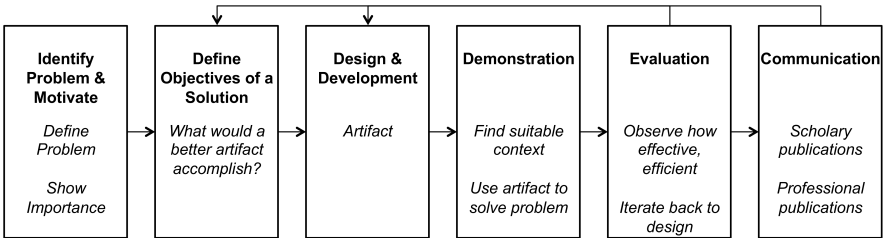


Fig. 1.3: The Design Science Research process, adapted from (Peffers et al., 2008)

This thesis develops four artifacts **A1**, **A2**, **A3**, **A4** respectively addressing the four research questions **RQ1**, **RQ2**, **RQ3**, **RQ4**. Each artifact is devised following the Design Science Research (DSR) approach (Peffers et al., 2008). These artifacts are interconnected to one another in that each of them tackles a different perspective of the overarching software development process. Their combinations enables a view on the *real* Gantt chart of the project, as illustrated in Figure 1.4. Such techniques are also useful to inform computationally-driven theory building (Berente et al., 2019).

Research rigor and validity are ensured by evaluating the artifacts with the Framework for Evaluation in Design Science Research (FEDS) (Venable et al., 2016). 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 chinfinicaoosing 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

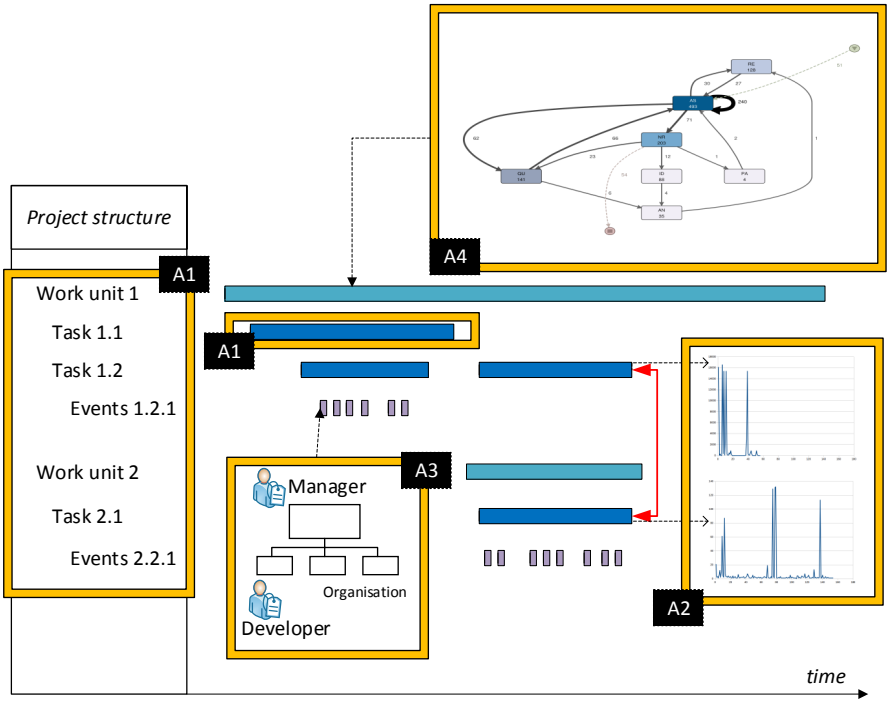


Fig. 1.4: The empirical Gantt chart of software development. The four process perspectives combined.

evaluation episode(s). In the following, we describe each designed artifact and their evaluation.

Gregor et al. (2020) define a schema for defining *design principles* so that they are understandable and useful in real world context.

Shoud I define design principles according to their schema? This may be a good idea!

1.3.1 Discovering the Temporal Perspective of the Software Development Process (Artifact A1)

Design objectives of **A1** aure *(i)* has to be applicable, *(ii)* usable, *(iii)* simple and, *(iv)* provide overview as well as detailed view. The following features are included: *(i)* time information, *(ii)* duration of activities, *(iii)* structure of the project, and *(iv)* roll-up and drill-down on granularity of events. Evaluation goals are *(i)* rigor, *(ii)* uncertainty and risk reduction, *(iii)* ethics, and *(iv)* efficiency. They are respectively addressed as follows. Rigor is assured by the design science approach. Uncertainty involves technical infeasibility and unavailuability of the data. Infeasibility risk is reduced by starting from the design of small artifacts with basic features and

incrementally improve. Unavailability risk is reduced by replication of the dataset locally. An ethics problem may regard the "big-brother is watching you" effect on people. This will be solved considering only data from partners and public repositories. Efficiency is evaluated by measuring whether all time patterns are found in a reasonably short time.

This artifact is evaluated through a Quick & Simple strategy (Venable et al., 2016), i.e., the technique is directly evaluated with real data. This will include three iteration episodes *(i)* initial design of Gantt chart (formative) *(ii)* evaluation of successful identification events, their duration and project structure; and *(iii)* run of the tool on real projects and provide a visualization (summative).

1.3.2 Discovering the Case Perspective of the Software Development Process (Artifact A2)

Design objectives of **A2** include applicability, simplicity and providing comparison of different cases. The following features are designed: *(i)* difference between work instances; *(ii)* measure of work: amount of changes; *(iii)* handle unstructured data from comments; *(iv)* evolutionary analysis of files; and *(v)* uncover work dependencies. Evaluation goals are as follows. Rigor is assured by the iterations. Uncertainty and risk reduction consist in: technical infeasibility – addressed through development by smaller iterations, data unavailability – addressed by local replication of data, and simplicity of results – addressed by exploiting user comments and derive informative process labels. Ethics problems do not arise as the data for this artifact is already public on GitHub. Efficiency is evaluated by finding all work dependencies in a reasonable time.

This artifact is evaluated through a Technical risk & Efficacy strategy (see Venable et al. (2016)), i.e., the technique is firstly evaluated with a toy example, then revised and applied to a large number of software development projects. Five iteration episodes are included: *(i)* initial design of artifact and results visualizations(formative); *(ii)* revision and choice of final visualization; *(iii)* evaluation of efficacy on retrieving features; *(iv)* run the technique on toy example (artificial evaluation), validate efficacy and usefulness, revise artifact; and *(v)* run the technique on real data sets from GitHub projects (summative and naturalistic evaluation).

1.3.3 Discovering the Organizational Perspective of the Software Development Process (Artifact A3)

Design objectives of **A3** include applicability and correct classification of roles. The following features are designed *(i)* determine roles of users; *(ii)* handle user comments; and *(iii)* automatically classify resources. Evaluation goals are as follows. Rigor is assured by the iterations. Uncertainty and risk reduction regard classification.

Incorrect resource classification risk is reduced by using different training set data from industry partners and obtaining feedback. An ethics risk is about obtaining information about people's work. This is mitigated by NDAs signed by the parties involved. Efficiency is evaluated in terms of precision and recall.

This artifact is evaluated through a Quick & Simple strategy (Venable et al. (2016)), i.e., the technique is evaluated with real data from industry partners. Three iteration episodes are included: *(i)* initial design of the artifact voted to a simple and structured representation of the data (formative) *(ii)* exploration and selection of the best features and classifiers. *(iii)* evaluation of results with real data from partners and feedback (summative + naturalistic).

1.3.4 Discovering the Control-Flow Perspective of the Software Development Process (Artifact A4)

Design objectives of **A4** are its applicability to pull requests and its functionality to provide informative and reliable process models about specific work patterns. The following features are designed. *(i)* handle comments from forum conversations, and *(ii)* discover a process model. Evaluation goals are considered as follows. Rigor is assured by the iterations. Uncertainty and risk reduction the following *(i)* activities of business processes are not mapped correctly – mitigated by manual annotation; and *(ii)* the resulting model is not informative – mitigated by statistical techniques voted to cluster traces into significant groups. Ethics problems do not arise because only data from open source repositories will be used. Efficiency is measured by the extent to which the artifact can deal with complex projects quickly.

This artifact is evaluated through a Technical risk & Efficacy strategy (see Venable et al. (2016)), i.e., the technique is firstly evaluated with an initial well-known dataset, then revised and evaluated with other projects. Three iteration episodes are included: *(i)* initial exploratory analysis on the applicability of Process mining (PM) techniques on pull requests (formative); *(ii)* evaluation of PM methods and assessment of statistical significance of results; and *(iii)* mine process models that are statistically significant and analyze the idea generation patterns (summative + naturalistic).

1.4 Generated Datasets

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 1.2 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 (Majchrzak and Malhotra, 2016). 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 (Majchrzak and Malhotra, 2016).

Table 1.2: Available datasets

Event Logs	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

1.5 Research Contributions

This doctoral thesis aims at bridging the gap between automatic analysis of software development data and process mining. It provides analyses techniques that help learning specific characteristics of the software process from its trace data. Beyond the publications, this thesis makes the following contributions.

- **Conceptualization of project-oriented business processes.** Processes have been seen so far as repeatable endeavors. However, there are processes which are not repeated exactly in the same way twice. This is the case with project-oriented business processes. Such processes, are conducted as projects, in that they are usually planned a priori and run under limited resources.
- **Visualization.** Existing techniques in the software engineering area are able to provide several possible views on the software development. Major visualization techniques can be divided into graph-based (Germán and Hindle, 2006; Greevy et al., 2006), notation-based (McNair et al., 2007), matrix-based (Gîrba et al., 2004) and metaphor-based (Wettel and Lanza, 2007). These visualization techniques help in multiple ways, such as understanding code smells, refactoring, evolution, distribution of work etc. However, most of the works in software engineering area are not process-aware. Thus, this thesis aims at researching the most suitable visualization techniques that help understanding the work process that lays behind software development.
- **Extension of the scope of process mining.** Process mining techniques rely on structured data. These data are typically captured by the standard XES

(Xes-standard.org, 2015) which defines precisely the input of process mining techniques. This work extends process mining towards mining processes also from data that are not directly represented in the XES standard.
- **Datasets for further research.** During the course of this research a number of datasets have been generated. These datasets are available for further use by researchers who want to replicate these studies or pursue different goals using the same concepts that this thesis provides.

1.6 Thesis Structure

This thesis is structured as follows.

- **Chapter 1: Introduction** motivates research on the topic of mining processes from software repositories. This chapter also scopes the problem addressed by this dissertation and provides an overall guide to the content of this manuscript.
- **Chapter 2: Research Background** presents the body of knowledge which inform this thesis. This chapter also elaborates on existing solutions and techniques that are present in the literature which are used as a base to develop new artifacts for mining the software process.
- **Chapter 3: Discovering the Temporal Perspective** tackles the problem of discovering the temporal aspect (RQ1) of the software development process. A visualization technique is presented that simplifies the managers' understanding of such perspective.
- **Chapter 4: Discovering the Case Perspective** presents an approach that helps understanding how different artifacts co-evolve in software development. With this approach, it is possible to compare how similarly different artifacts evolve and consequently cluster them into similar development cases (RQ2).
- **Chapter 5: Discovering the Organizational Perspective** tackles the problem of automatically identifying emerging roles in software development. Thus, a technique for classifying human-resources (RQ3) based on factual data is presented.
- **Chapter 6: Discovering the Control-Flow Perspective** complements the techniques presented in previous chapter with the information about which activities were executed by resources. These activity labels along with the temporal help understanding the control-flow (RQ4) of the development process.
- **Chapter 7: Further Discovery Approaches** presents two more approaches which use trace data to gather insights on the process. The first approach provides a showcase where trace data analyses of software repositories is combined with subjective information from questionnaire. Mixing these methods helps to better identify issues in and suggest interventions based on facts. The second approach leverages on speech act theory to better understand the work done in by software developers. Unstructured data from conversations is used to analyze collaboration and derive a process (RQ4).

- **Chapter 8: Conclusion** summarizes the main findings. Further more, it discusses their implications along with future research.

1.7 Publications

This dissertation has led to the publications listed below.

Mining the Time Perspective(**RQ1**): (Bala et al., 2015)

- **Bala, S.**, Cabanillas, C., Mendling, J., Rogge-Solti, A., and Polleres, A.: *Mining Project-Oriented Business Processes*. In Hamid Reza Motahari-Nezhad, Jan Recker, and Matthias Weidlich, editors, BPM 2015, Innsbruck, Austria, volume 9253 of Lecture Notes in Computer Science, pages 425–440. Springer, 2015.

Mining the Case Perspective (**RQ2**): (Bala et al., 2017b).

- **Bala, S.**, Revoredo, K., de A.R. Gonçalves, J.C., Baião, F., Mendling, J., Santoro, F.: *Uncovering the Hidden Co-evolution in the Work History of Software Projects*. In: Carmona, J., Engels, G., and Kumar, A. (eds.) Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings. pp. 164–180. Springer (2017).

Mining the Organizational Perspective (**RQ3**): (Agrawal et al., 2016).

- Agrawal, K., Aschauer, M., Thonhofer, T., **Bala, S.**, Rogge-Solti, A., Tomsich, N.: *Resource Classification from Version Control System Logs*. In: Proceedings - IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshop. pp. 249–258 (2016).

Mining the Control-Flow Perspective (**RQ4**): (Bala et al., 2020)

- **Bala, S.**, Kneringer, P., & Mendling, J.: *Discovering Activities in Software Development Processes*. In: CEUR Workshop Proceedings. (2020)

The following publications are related to the requirements.

- **Bala, S.**, Cabanillas, C., Haselböck, A., Havur, G., Mendling, J., Polleres, A., Sperl, S., Steyskal, S.: *A Framework for Safety-Critical Process Management in Engineering Projects*. In: Ceravolo, P. and Rinderle-Ma, S. (eds.) Data-Driven Process Discovery and Analysis- SIMPDA. pp. 1–27. Springer (2015). (Bala et al., 2017a) – Related to **RQ1** & **RQ3**.
- **Bala, S.**, Havur, G., Sperl, S., Steyskal, S., Haselböck, A., Mendling, J., Polleres, A.: *SHAPEworks: A BPMS extension for complex process management*. In: CEUR Workshop Proceedings. pp. 50–55 (2016). (Bala et al., 2016) – Related to **RQ1** & **RQ3**.

Early stage concepts of this research (research proposal) have been presented in doctoral consortium and conference.

- **Bala, S.**: *Mining projects from structured and unstructured data*. In: CEUR Workshop Proceedings (2017). (Bala, 2017)
- **Bala, S.**, Mendling, J.: *Monitoring the Software Development Process with Process Mining*. In Boris Shishkov, editor, Business Modeling and Software Design, volume 319 of Lecture Notes in Business Information Processing, 2018 (Bala and Mendling, 2018)

Further work published in the Business Process Management (BPM) area is the following.

- Woliński, B., **Bala, S.**: *Comprehensive Business Process Management at Siemens: Implementing Business Process Excellence*. In: vom Brocke, J. and Mendling, J. (eds.) Business Process Management Cases: Digital Innovation and Business Transformation in Practice. pp. 111–124. Springer International Publishing, Cham (2018). (Wolinski and Bala, 2018)
- **Bala, S.**, Mendling, J., Schimak, M. and Queteschiner, P., 2018, October. *Case and activity identification for mining process models from middleware*. In IFIP Working Conference on The Practice of Enterprise Modeling (pp. 86-102). Springer, Cham. (Bala et al., 2018)
- Azemi, E., **Bala, S.**: *Exploring BPM adoption and strategic alignment of processes at Raiffeisen Bank Kosovo*. In: BPM (Industry Forum). pp. 37–48. CEUR-WS.org (2019). (Azemi and Bala, 2019)
- Vidgof, M., Djurica, D., **Bala, S.**, and Mendling, J. (2020). *Cherry-picking from spaghetti: Multi-range filtering of event logs*. In BPMDS/EMMSAD@CAiSE, volume 387 of Lecture Notes in Business Information Processing, pages 135–149. Springer. (Vidgof et al., 2020)
- Vidgof, M., Djurica, D., **Bala, S.**, Mendling, J.: *Interactive Log-Delta-Analysis using Multi-Range Filtering*. Software and Systems Modeling, (2021). (Vidgof et al., 2021)
- Azemi, E., **Bala, S.**: *Exploring BPM adoption and strategic alignment of processes at Raiffeisen Bank Kosovo*. In: vom Brocke, J., Mendling, J., Rosemann, M. (eds.) Business Process Management Cases. Springer (2021). (Azemi and Bala, 2021)
- Waibel, P., Novak, C., **Bala, S.**, Revoredo, K., Mendling, J.: *Analysis of Business Process Batching Using Causal Event Models*. In: Leemans, S.J.J. and Leopold, H. (eds.) ICPM Workshops. pp. 1–13. Springer Nature Switzerland AG (2021). (Waibel et al., 2021)

Chapter 2
Research Background

This dissertation is in the intersection between process mining and mining software repositories, respectively belonging to the broader contexts of BPM and software development process. This chapter, hierarchically delves into the details of these fields as follows. Section 2.1 gives an introduction on the main concepts of the software development process, elaborating on methodologies and repositories in which the data is stored. Section 2.2.1 provides an overview on BPM and introduces the BPM lifecycle. Section 2.2.2 describes the specific area of process mining along with its requirements and limitations when analyzing software. Section 2.1.4 describes related work on mining data from these repositories.

2.1 Software Development Process

The software development process is a specific kind of process. Differently from standard business processes, it is run without referring to a process model (e.g., BPMN, Petri net). A common trait of both software development and business is the fact that they both describe and endeavor to organize work in such a way that should be measured and hence monitored and improved. Therefore, both areas rely on methods for managing their processes. Popular methodologies in software development are Waterfall, Spiral, and Agile. Agile methods further refine into branches such as Scrum and Kanban. In order to measure the effectiveness of these methodologies, we can analyze the underlying trace data that is generated in the various steps. In the following, we further elaborate on software development methodologies and the software repositories used to keep track of the development process.

2.1.1 Overview on Software Development

How does software development work

The software development process is an practice of diving the creation of an artifact into smaller steps. The final goal is to create or improve an existing piece of software. Each step typically defines specifications about the input and output. The totality of the steps is also called software development life cycle. There are many existing methodologies, each with an specific life-cycle. Popular methodologies are agile, waterfall, spiral, rapid application development and extreme programming.

Picture:
software process methodology (guidance over the process) data genereated

2.1.2 Software Development Methodologies

One paragraph describing what SDM is

Existing research shows that explicit knowledge defined by Software Development Methodologies (SDMs) improves productivity of software development enterprises and quality of the developed software. This is achieved by increasing enterprises' ability to transfer knowledge between employees, systematically manage software development process, etc. (Avison and Fitzgerald, 2003; Fitzgerald, 1998; Hovelja et al., 2015; Riemenschneider et al., 2002). However, it is not enough that an enterprise only describes its SDM in a document; the developers need to use it consistently in their everyday work. The use of SDM was often a topic of research in the last decades, since SDM adoption among software developers was relatively low and the developers often preferred different ad hoc approaches (Aaen, 2003; Fitzgerald, 1996; Huisman and Iivari, 2006).

The use of SDMs in enterprises can be analyzed with the help of different existing approaches (Aboelmaged, 2010a; Venkatesh and Davis, 2000; Wang et al., 2013). One of the grounding theories in the field of SDM evaluation is diffusion of innovations theory (Rogers, 2003) according to which SDM is considered as innovation that developers adopt (Gallivan, 2003; Green et al., 2005; Iivari and Huisman, 2001). To obtain information about the studied SDM and/or its elements the aforementioned studies focused on perceptions of different stakeholders, namely developers, managers, users, etc. to measure characteristics like level of use, assimilation, social and technical suitability, developer satisfaction and impact on performance (Atkinson, 1999; Cooper and Zmud, 1990; Rogers, 2003; Vavpotic and Bajec, 2009; Vavpotic and Hovelja, 2012). Even better insight into SDM and/or its elements can be gained by comparing the perceptions of different stakeholders regarding the same SDM and/or its element (Hovelja et al., 2015).

Another important theoretical development in the field was that SDM should be studied on the level of its constituent elements like activities, tools, roles, produced documents, etc. and not only as a whole. This improved the understanding of suitability of studied SDM elements for a certain development team and enabled comparison between the studied SDM elements. Thus, allowing enterprises to better pinpoint problematic elements of SDM, prepare focused improvements and examine the link between a specific SDM element and overall project success (Atkinson, 1999; Hovelja

et al., 2015). Such development is in line with findings in the field of situational method engineering (Karlsson and Ågerfalk, 2009; Ralyté et al., 2003) that constructs a custom SDM from those SDM elements that fit with characteristics of certain development team and other situational factors (internal and external enterprise's environment).

Another potential source of information regarding development process emerged in recent years with widespread use of tools that support software development activities like issue tracking, requirements management, test management etc. Such tools store valuable data about actual execution of the development process and give us additional information about SDM and its elements thus importantly complementing stakeholder perceptions (Choetkiertikul et al., 2017; Mäntylä et al., 2016; Ortu et al., 2016, 2015).

2.1.3 Software Repositories

what research has been doing around sw. repos. e.g, coordination,

A software repository is a location that stores data about software. Generally software repositories serve the purposes of promoting and managing collaboration among developers. Data stored in a repository include source code, documentation, libraries, models and other digital artifacts that support a specific software.

Repository data can be structured and unstructured (Bacchelli, 2016; Bavota, 2016). Data generated by systems are usually structured according to a preset schema. For example, the system may always record the number of changes to a file along with the version number and the user who last changed it. The purpose of these kind of data is to be processed by machines. Conversely, unstructured data are pieces of information that are used by developers to communicate. Examples of unstructured data are posts developer forums and emails.

Not all the data from software is stored in one single repository (Kalliamvakou et al., 2016). Thus, in order to gain a more complete understanding, integration of more repositories is considered. This increases the chance of dealing with large volume of data (Dyer et al., 2015; Boettiger, 2018; Boldi et al., 2020).

Software repositories are typically managed by Software Configuration Management (SCM), a process to systematically organize and control the changes during the software development life-cycle (Estublier, 2000). SCM relies on software such a VCS to keep track of the evolution of files. To manage the tasks of developers, software repositories may adopt an Issue Tracking System (ITS) (Bertram et al., 2010). Popular ITS include Jira[1], Bugzilla[2], Micosoft Dynamics CRM[3], Trello[4], etc.

[1] https://www.atlassian.com/software/jira

[2] https://www.bugzilla.org

[3] http://dynamics.microsoft.com

[4] http://trello.com

Version control systems are a part of SCM that are used for keeping track of the different versions of a file. Tracking is required not only explicitly as part of some activities but also to comply with norms and regulations that may require some evidence of the actions being performed in the organization. Documents are usually free of format or contain tables, at best. The unstructuredness of data makes it difficult to monitor processes and check rules on them. A starting point for analysis of project-oriented processes can be data logs that are stored in Software Configuration Management (SCM) systems that help tracking the evolution of data and restore information if needed (Voinea and Telea, 2006a). However, hundreds of versions of thousands of files are common in a single project (Voinea and Telea, 2006b), which makes it impractical to browse this data manually.

Search: (Hu et al., 2020) provide an algorithm for code retrieval from the StackOverflow[5] forum.

(Feiner and Andrews, 2018) develop a system that allows full text search and visualization of over a software repository. The tool supports

Let us see an example inspired by a real scenario of a process to write a project proposal that uses a Version Control System (VCS) to store the data. The project history, and hence, the data produced, starts when people begin to work on the proposal, which involves a description of the project goals and milestones, a division of tasks into work packages, an estimation of cost and resources required, etcetera. This information is spread in the repository over several folders containing different documents, which are later merged into a single file. If the proposal is accepted, the first step is to organize a kickoff meeting and assign specific resources to the work packages. A hierarchical set of folders is then created in the repository in order to store the information generated for each work package. As the project evolves over time, resources contribute by adding, removing or modifying information to the VCS repository. Project evolution is guided by specific norms that impose the execution of predefined steps. For instance, the European norm EN5016 requires a preliminary Reliability, Availability and Maintainability (RAM) analysis to support targets.

Table 4.1 depicts an excerpt of the log data generated, where the first column (on the left hand side) indicates the commit identifier. The second column indicates the person who committed changes. The third column indicates the commit date. The fourth column shows a comment from the developer that describes the changes performed. The fifth column indicates the files affected and the type of action performed in terms of differences between the previous and current version. For the sake of simplicity, the table shows the log data of a specific time period and the actions related to a specific task, namely, *Define example*. That task was assigned to resource *John* and was worked on by resource *Mary* and, later on, also by resource *Paul*.

Existing frameworks, such as Subversion or Git, allow to access their logs in different ways. However, the covered information is limited to (roughly) that depicted in Table 4.1. Especially for big projects that are frequently updated over a large period of time, these logs are complex to analyze. Therefore, the problem to address is how to analyze and visualize the information produced in project-oriented business

[5] stackoverflow.com

Table 2.1: An excerpt of a VCS log data

Id	Project Participant	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()

processes such that it can be represented in an understandable and manageable way by project experts and enable, a.o., the automation of mechanisms for compliance checking. The following properties of project-oriented process logs must be taken into account to achieve this goal: (i) VCS repositories consist of a hierarchy of folders and files which are logically organized such that work is grouped in a specific way; (ii) process activities are not registered in VCS log entries. Therefore, such information must be inferred by reasoning on the repository structure and/or the content of the log entries; (iii) the granularity of the events is unknown a priori and it needs to be defined before analyzing the data.

2.1.4 Mining Software Repositories

Which sort of analyses techniques were used

Mining software repositories applies machine learning algorithms to data from software, with the goal of extracting knowledge. This area is used as the base for providing techniques and algorithms as solutions to engineering problems. These problems revolve around questions aimed at understanding various performance and functionality issues with the software at several stages. Examples of knowledge are related to the problems "where-to-log", "who-did-what", "what-does-this-change-mean", "who can work on this bug"

Data used for analyses come from a variety of systems such as version control, issue tracking, emails, log data from IDE, and project management.

For the analyses a variety of machine learning algorithms are used (**?**). When the data is unstructured, Natural Language Processing (NLP) is used for preprocessing. Also, visualization is a common technique to show results (**?**) (e.g., software as cities).

Most of the methods are quantitative. However there are also a number of empirical studies (Kalliamvakou et al., 2016; Dehghan et al., 2017)

2.2 Business Process Analytics

2.2.1 Business Process Management

Business Process Management (BPM) is the discipline that oversees how work is performed in an organization to ensure consistent outcomes and exploit improvement opportunities (Dumas et al., 2018). Rather than focusing on specific activities, events or decisions, BPM focuses on how their interplay adds value to the organization and its customers. These chains of events are named *processes*. More formally, a business process is set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal (Weske, 2019).

Business process management aims at improving business processes following an iterative methodology which is refereed to as the *BPM lifecycle* (Dumas et al., 2018). It consists of six phases: *i)* process identification, *ii)* process discovery, *iii)* process analysis, *iv)* process redesign, *v)* process implementation, and *vi)* process monitoring.

In the *process identification* phase a business problem is posed. The main goal of this phase is to define, enumerate and put into relation existing processes in the company. An artifact produced by this phase is a *process architecture*. Such provides a guiding map for selecting which processes to further manage in the following BPM phases. Typically, the processes that are most relevant to the posed business problem will be highlighted for further management. In the *process discovery* phase, processes are documented. Typically this is done via modeling the process in some modeling language (e.g., BPMN[6], Petri net (Lohmann et al., 2009), EPC (Scheer et al., 2005), Yet Another Workflow Language (YAWL) (van der Aalst and ter Hofstede, 2005)). The outcome of this phase is a set of so-called *as-is* process models. In the *process analysis* phase, the as-is processes are analyzed via qualitative and quantitative methods. The output of this phase is a collection of weaknesses in the as-is process along with their impact and estimated effort to address them. In the *process redesign* phase, the goal is to identify those changes to the existing process that address the previously identified issues and contribute to solving the posed business problem. Redesign changes are usually backed up by the analyses results Performance indicators are also used to measure performance of the processes. of the previous phase. An output of this phase is a redesigned process, which typically is also modeled into a *to-be* process model. In the *process implementation* phase, the previously identified changes are implemented. This phase covers both changing

[6] https://www.omg.org/bpmn

the way work is performed in the organization and the set up and development of the IT systems required for the realization of the *to-be* process. Special focus is here dedicated to the technical challenges of automating the new process with the goal of running it on a Business Process Management System (BPMS). Thus, the output of this phase consists of a set of executable process model. In the *process monitoring* phase, various measures of the running process are collected. They provide the basis for analyzing the success of redesign, collecting new insights that emerged from the execution context and determining how well the new process is performing against defined quality/performance indicators. Deviations, issues and bottlenecks are here collected. They provide the starting point of a new iteration of the BPM lifecycle. Figure 2.1 summarizes the phases of the lifecycle and their interplay.

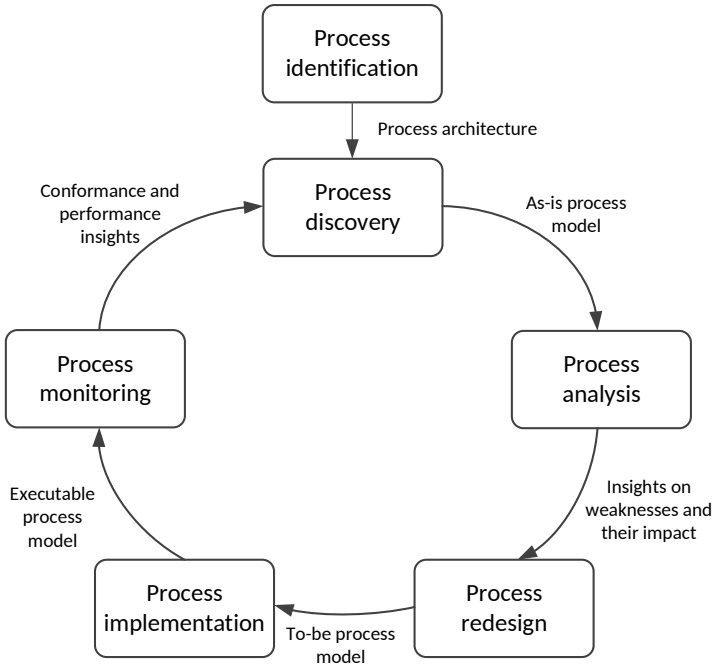


Fig. 2.1: The BPM lifecycle. Adapted from (Dumas et al., 2018)

Coordination of work in business processes can be viewed from different perspectives. When analyzing business processes, a good characterization can be achieved with the following four perspectives (van der Aalst, 2016). First, the *control-flow perspective* aims at capturing the all the possible paths in which activities may follow one-another in a process. Second, the *organizational perspective* captures which actors (e.g., people, organizational units, roles) participate in the business process and how they interact with each other. Third, the *case-perspective* captures the properties of specific process instances such as the evolution of certain data-objects, originators

of the case, specific values of process variables, etc. Fourth, the time-perspective captures temporal information about the process, like when and how often certain activities occurred and what was their duration.

2.2.2 Process Mining

Process mining (PM) is a sub-field of BPM that emerged in the last decade, with main focus on the monitoring and discovery phases of the lifecycle. The goal is to provide fact-based insights and support process improvement. On a broader context, PM can be considered as the missing link between traditional model-based process analysis and data driven techniques such as data mining and machine learning (van der Aalst, 2016). Compared to existing Business Intelligence (BI) technologies, PM techniques go beyond the calculation of simple 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 PM techniques can infer models from event logs, which inform about the diverse aspects of a business process. As defined in (van der Aalst, 2016), main *perspectives*[7] of a business process are the four. First, the *time perspective* aims at analyzing time and frequency of process events. Second, the *case perspective* aims at identifying properties of process cases. Third, the *organizational perspective* aims at analyzing the event log to gain transparency on the resources involved in the process. Fourth, 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.

There are three types of PM, namely *i)* process discovery; *ii)* conformance checking; and *iii)* enhancement. Process mining is becoming widely adopted with considerable number of algorithms from academia and several industry tools such as Celonis[8], Disco[9], minit[10], LANA Process Mining[11] and more[12]. This proposal focuses more on the discovery part, i.e., take an event log as an input and abstract process patterns from it.

In the context of mining the software development process, several works in the area of PM have tackled the problem by transforming it into a PM problem. Consequently, approaches have been developed to preprocess VCS data such that PM techniques can be applied, and hence, a business process can be derived from the log data. In this group, (Kindler et al., 2006b,a) developed an algorithm for extracting software processes that are mapped to Petri Nets. Activities, which are not explicit in the logs, are discovered from their input and output artifacts. However,

---

[7] Note that the different perspectives are partially overlapping. Yet, they are widely used in the BPM community.

[8] https://www.celonis.com

[9] https://fluxicon.com/disco

[10] https://www.minit.io

[11] https://lana-labs.com/en

[12] https://www.processmining-software.com/tools

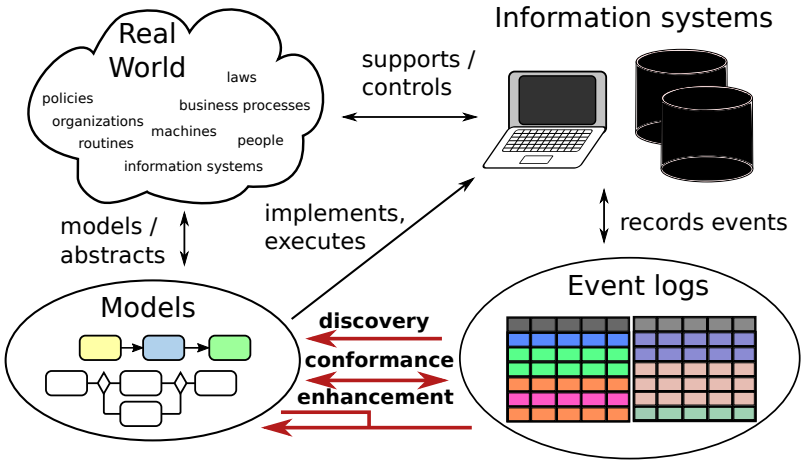


Fig. 2.2: The process mining framework. Adapted from (van der Aalst, 2016).

strong assumptions are made on the filenames as well as on the software process lifecycle. Rubin et al. (2007) addressed the problem of engineering processes that are not well documented and are usually unstructured. They provided a bridge from Kindler et al.'s approach to ProM (van Dongen et al., 2005) in order to mine different process perspectives, such as performance social network analyses. Rubin et al. (2014) applied PM to the touristic industry and obtained user processes from web client logs pursuing the goal of improving the software system by analyzing the underlying process. Poncin et al. (2011a) developed the FRASR framework for preprocessing software repositories to transform the VCS data to logs that conform to the PM event log meta model (van Dongen and der Aalst, 2005) as utilized in ProM (van Dongen et al., 2005). However, these approaches disregard the single-instance nature of project-oriented business processes and treat them as procedures that can be repeated over time. Process mining techniques are related to all the research questions of this thesis. Especially, process discovery helps with addressing **RQ4**.

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 (Song and van der Aalst, 2007) 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 (Rubin et al., 2007) 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.

To sum up, the analyses of software would benefit from a processes mining techniques, especially when it comes to uncovering behavioral patterns that are

linked to the work of the developers. However, process mining techniques have not sufficiently been exploited with software development data, due to the fact that such data is not readily consumable by process mining techniques. Moreover, useful process mining techniques should provide insights that can be understood and enacted by software domain experts. Therefore, such techniques should not only look into the process but also be aware of the existing methods and performance indicators for software.

2.3 Summary

Write summary

Chapter 3
Article 1: Mining Project-Oriented Business Processes

Authors:

Saimir Bala, Cristina Cabanillas, Jan Mendling, Andreas Rogge-Solti, and Axel Polleres

Published in:

Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 – September 3, 2015, Proceedings

Abstract:

Large engineering processes need to be monitored in detail regarding when what was done in order to prove compliance with rules and regulations. A typical problem of these processes is the lack of control that a central process engine provides, such that it is difficult to track the actual course of work even if data is stored in version control systems (VCS). In this paper, we address this problem by defining a mining technique that helps to generate models that visualize the work history as GANTT charts. To this end, we formally define the notion of a project-oriented business process and a corresponding mining algorithm. Our evaluation based on a prototypical implementation demonstrates the benefits in comparison to existing process mining approaches for this specific class of processes.

Make cover page. Example: This chapter presents the central research results, which relate to methods and techniques for quantifying inconsistency in business rule bases, pin-pointing elements responsible for the overall inconsistency, and resolving inconsistency. The results and their interrelations are also summarized in a concluding discussion. As stated in the disclaimer, the following results have already been published in various works. Subsequently, the following section includes many materials of these works, which are cited accordingly. All original material was written by myself, except if explicitly stated otherwise.

3.1 Introduction

Business process management plays an important role for improving the performance and compliance of various types of processes. In practice, many processes are executed with clear guidelines and regulatory rules, but without an explicit centralized control imposed by a process engine. In particular, it is often important to exactly know when which work was done. This is, for instance, the case for complex engineering processes in which different parties are involved. We refer to this class of processes as project-oriented business processes.

Such project-oriented business processes are difficult to control due to the lack of a centralized process engine. However, there are various unstructured pieces of information available to analyze and monitor their progress. One type of data that are often available these processes is event data from version control systems (VCS). While process mining techniques provide a useful perspective on how such event data can be analyzed, they do not produce output that is readily organized according to the project orientation of these processes.

In this paper, we define formal concepts for capturing project-oriented processes. These concepts provide the foundation for us to develop an automatic discovery technique which we refer to as *project mining*. The output of our project mining algorithm is organized according to the specific structure typically encountered in project-oriented business processes. With this work, we extend the field of process mining towards the coverage of this specific type of business process.

The paper is structured as follows. Section 6.2 describes the research problem and summarizes insights from prior research upon which our project mining approach is built. Section 5.3 defines the preliminaries of our work and presents an algorithm to mine project-oriented business processes. Section 4.4 describes the implementation of this algorithm and discusses the results from its application to VCS logs from a real-world engineering project. Section 3.5 highlights the implications of this work before Section 3.6 concludes.

3.2 Background

Here, we describe the addressed problem and related work.

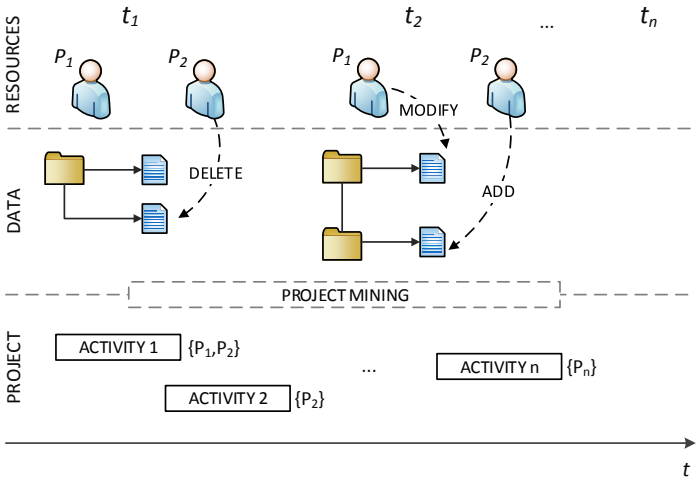


Fig. 3.1: Problem illustration

3.2.1 Problem Description

The class of processes that we discuss in this paper are long-term engineering projects. These processes have specific requirements for monitoring. First, they are executed only once according to the specific needs of a particular project, and only partially according to recurring process descriptions. Second, they involve various actors that typically document their work in a semi-structured way using text and tables. Third, work in the project is usually subject to constraints regarding the start and end and the temporal order. Fourth, there is typically no process engine controlling the execution. Fifth, even though these limitations in terms of traceability exist, there are usually strong requirements in terms of tracking when which work was conducted.

In line with these observations, a *project-oriented business process* can be defined as an ad-hoc plan that specifies the tasks to be performed within a limited period of time and with a limited set of resources for achieving a specific goal. Unlike repetitive business processes for which notations such as BPMN OMG (2011) or EPC van der Aalst (1999) are commonly used, project-oriented business processes may be properly represented with PERT or GANTT models. The concept is illustrated in Fig. 3.1.

Documentation is required not only explicitly as part of some activities but also to comply with norms and regulations that may require some evidence of the actions being performed in the organization. Documents are usually free of format or contain tables, at best. The unstructuredness of data makes it difficult to monitor processes and check rules on them. A starting point for analysis of project-oriented processes can be data logs that are stored in Software Configuration Management (SCM) systems that help tracking the evolution of data and restore information if needed Voinea and Telea (2006a). However, hundreds of versions of thousands of files are common in a

Table 3.1: Excerpt from VCS log data for the referenced time period

CID	Resource	Date	List of changes
1	Y	2014-11-12 11:57:46	A /example A /example/SHAPE/ToyStationExample.docx
...	...	...	...
3	X	2014-11-14 16:34:07	M /example/ToyStation.bpmn M /example/ToyStation.png
4	W	2014-12-15 13:49:11	D /example/Download
5	W	2015-01-08 16:06:41	A /example/Download2
6	X	2015-01-13 11:47:09	M /example/ToyStation_0Loop.bpmn M /example/ToyStation_nLoop.bpmn
7	Z	2015-01-16 16:50:29	A /example/ToyStation_0Loop.pdf A /example/ToyStation-feedbackZ.pdf

single project Voinea and Telea (2006b), which makes it impractical to browse this data manually.

Let us see an example inspired by a real scenario of a process to write a project proposal that uses a Version Control System (VCS) to store the data. The project history, and hence, the data produced, starts when people begin to work on the proposal, which involves a description of the project goals and milestones, a division of tasks into work packages, an estimation of cost and resources required, etcetera. This information is spread in the repository over several folders containing different documents, which are later merged into a single file. If the proposal is accepted, the first step is to organize a kickoff meeting and assign specific resources to the work packages. A hierarchical set of folders is then created in the repository in order to store the information generated for each work package. As the project evolves over time, resources contribute by adding, removing or modifying information to the VCS repository. Project evolution is guided by specific norms that impose the execution of predefined steps. For instance, the European norm EN5016 requires a preliminary RAM analysis to support targets.

Table 3.1 depicts an excerpt of the log data generated, where the first column (on the left hand side) indicates the commit identifier, the second column indicates the person who committed changes, the third column indicates the commit date, and the fourth column indicates the files affected and the type of action performed among added (A), modified (M) and deleted (D). For the sake of simplicity, the table shows the log data of a specific time period and the actions related to a specific task, namely, *Define example*. That task was assigned to resource X and was supervised by resource Y and, later on, also by resource Z.

Existing frameworks, such as Subversion or Git, allow to access their logs in different ways. However, the covered information is limited to (roughly) that depicted in Table 3.1. Especially for big projects that are frequently updated over a large period of time, these logs are complex to analyze. Therefore, the problem to address is

how to analyze and visualize the information produced in project-oriented business processes such that it can be represented in an understandable and manageable way by project experts and enable, a.o., the automation of mechanisms for compliance checking. The following properties of project-oriented process logs must be taken into account to achieve this goal: (i) VCS repositories consist of a hierarchy of folders and files which are logically organized such that work is grouped in a specific way; (ii) process activities are not registered in VCS log entries. Therefore, such information must be inferred by reasoning on the repository structure and/or the content of the log entries; (iii) the granularity of the events is unknown a priori and it needs to be defined before analyzing the data.

3.2.2 Related Work

The problem described has been addressed in the literature from different perspectives. The first category of related work tackles the problem by transforming it into a process mining problem. Consequently, approaches have been developed to preprocess VCS data such that process mining techniques can be applied, and hence, a business process can be derived from the log data. In this group, Kindler et al. (2006b,a) developed an algorithm for extracting software processes that are mapped to Petri Nets. Activities, which are not explicit in the logs, are discovered from their input and output artifacts. However, strong assumptions are made on the filenames as well as on the software process lifecycle. Rubin et al. (2007) addressed the problem of engineering processes that are not well documented and are usually unstructured. They provided a bridge from Kindler et al.'s approach to ProM van Dongen et al. (2005) in order to mine different process perspectives, such as performance social network analyses. Rubin et al. (2014) applied process mining to the touristic industry and obtained user processes from web client logs pursuing the goal of improving the software system by analyzing the underlying process. Poncin et al. (2011a) developed the FRASR framework for preprocessing software repositories to transform the VCS data to logs that conform to the process mining event log meta model van Dongen and der Aalst (2005) as utilized in ProM van Dongen et al. (2005). However, these approaches disregard the single-instance nature of project-oriented business processes and treat them as procedures that can be repeated over time.

The second category of related work focuses on the visualization of VCS data for different purposes. Several approaches study the interaction among developers over time from a visualization point of view. For instance, Ogawa and Ma (2010) drew storyline pathways to show the story of each developer's contribution. Other approaches analyze and visualize VCS data at file level in order to discover file version evolution. Voinea and Telea (2006b) introduced an interactive navigation method to surf file version evolution as well as two methods to cluster versions of the same file in an abstraction layer. Wu et al. Wu et al. (2004) also visualized the evolutions of entire projects at file level, emphasizing the evolution moments. Finally, several approaches study change prediction with the aim of discovering prediction patterns

that can help in the process of software development Zimmermann et al. (2005); Ying et al. (2004). The approaches mentioned in this category as well as others that apply similar techniques Feldt et al. (2013); Kagdi et al. (2006); D'Ambros and Lanza (2008) focus on studying software evolution from different standpoints. However, the goal pursued differs in all cases from our goal in that they are not interested in discovering projects tasks out of the log data, and hence, they lack an explicit notion of work structure that we need to consider for our purpose.

Our approach combines ideas from both areas, as we aim at identifying tasks like in the approaches that rely on process mining, but we must cluster the data in an appropriate way, for which techniques developed in the approaches that pursue visualization may be adapted or extended.

3.3 Mining VCS Event Data

Here, we first formalize the notions encountered in the project mining setting. Then we develop an approach to acquire a hierarchical overview on the project from a repository perspective.

3.3.1 Preliminaries

Version Control System (VCS) are used in projects to ensure reliable collaboration. We build our approach on VCS. Typically, the workflow in VCS is that people work on files (e.g., text, source code, spread sheets) and commit them to the central repository. Project participants comment on their commits so that other participants can better understand the nature of the changes to the files.

Let F be the universe of files. Files are organized in a file tree. Therefore, each file $f \in F$ has one parent file. The only file without a parent file is the root file. We capture this information in the parent relation $Parent : F \times F$. For example, let $f_p \in F$ be the parent of file $f_c \in F$, then $(f_p, f_c) \in Parent$. The transitive closure on the parent files is given by the function $ancestor : F \rightarrow 2^F$ that returns the set of files along the path to the root.

When project members did a certain amount of work and want to save their current progress, they commit the changes to the VCS. We define changes on files as the events of interest on the lowest granularity.

Definition 3.1 (Event) Let E be the set of events. An event $e \in E$ is a four-tuple (f, o, ts, k), where

- $f \in F$ is the affected file of the event.
- $o \in O = \{added, modified, deleted\}$ is the change operation on the file with obvious meaning.
- $ts \in TS = \mathbb{N}_0$ represents a unix time stamp marking the time of the event occurrence.

- $k \in \Sigma^*$ is a comment in natural language text.

For events $e = (f, o, ts, k)$ we overload f, o, ts, and k to be used as accessor functions. For example, f is the function $f : E \rightarrow F$ mapping an event to its affected file.

Project participants can commit a number of changes to different files at one step. Therefore, we define the notion of commits as follows.

Definition 3.2 (Commit) A commit C is a set of events sharing the same time stamp and comment, i.e., $\forall e, e' \in C : ts(e) = ts(e') \wedge k(e) = k(e')$. Additionally, each event in a commit affects different files, i.e., $\forall e, e' \in C : e \neq e' \rightarrow f(e) \neq f(e')$.

Usually, it is in the hands of project participants, when they decide to commit changes to the VCS. In the extreme case, there could be only a single commit made in a project that adds all files to the repository. Note that this extreme practice would render the use of a VCS obsolete. On the contrary, it is common practice to regularly perform commits in order to securely store work progress and to reduce the chance of conflicts (Pilato et al., 2008; Hou et al., 2014). Conflicts occur, when another participant committed changes to a file that is being committed and can cause extra work. Based on these insights, we make the assumption that commits are regularly made during work.

Projects are decomposed into work packages. We assume a hierarchical work package structure of a project, such that a work package can have sub work packages. Further, the amount of work in a single work package need not be done in one single time span, but it can be split into several activities. Activities have a start and end time, and subsequent activities can have idle periods in between. Thus, we define projects as follows.

Definition 3.3 (Project) A project P is a tuple $(W, S, A, \alpha, \omega, \beta)$, where

- W is the set of work packages in the project.
- $S \subseteq W \times W$ is the relation that hierarchically decomposes work packages into a tree structure.
- A is the set of activities that are conducted in the work packages.
- $\alpha : A \rightarrow TS$ is the function that assigns a start time to activities. Activities are ordered by their start times.
- $\omega : A \rightarrow TS$ is the function that assigns an end time to activities.
- $\beta : A \rightarrow W$ is the mapping function that maps activities to their corresponding work packages.

Note that this definition reflects an activity centric view on projects. The definition deliberately omits further dimensions, e.g., costs, resources, risks. The idea is not to capture projects in every detail, but to focus on the work packages of a project to obtain an overview of the work that is being done. We are interested in when work has been started in a work package, and when work packages have been done. This information can be derived from the activities associated to the workpackages. An obvious assumption is that the work package starts with its first activity, and ends when its last activity is completed.

Based on these notions, we can define the task of *project discovery* as reconstructing the project P from a set of low level event data E. In the following, we present an approach to this problem.

3.3.2 Project Discovery Technique

For project discovery from the VCS commit history, we need to identify activities that are performed, associate the activities to work packages and recreate the work package structure of the project. Our aim is to create a hierarchical model that provides an overview of the project work. Therefore, we have to identify the start and end times of activities and of work packages before we can visualize the project work. The input to the technique is the log that is stored in the VCS. The challenge is that the raw log only records commits on the file system level and information on activity level is missing. However, we can deduce activity information from events based on the following assumptions.

A1: Meaningful file tree structure. The file tree structure in a project represents its work package structure. That is, the knowledge workers organize their work in a file hierarchy that reflects the project structure.
A2: Local changes. Activities in a work package affect only files of the work package folder, or in the corresponding sub-tree in the file tree structure.
A3: Frequent commits. Commits to the VCS are regularly performed, when conducting work in an activity.

Note that assumption A1 can be seen as a strong assumption on the file tree structure. Nevertheless, we argue that even if A1 is not entirely met, the aggregation of work information on the file tree hierarchy provides a valuable view on the project. Figure 3.2 shows the different steps of the technique. We describe each of them in detail.

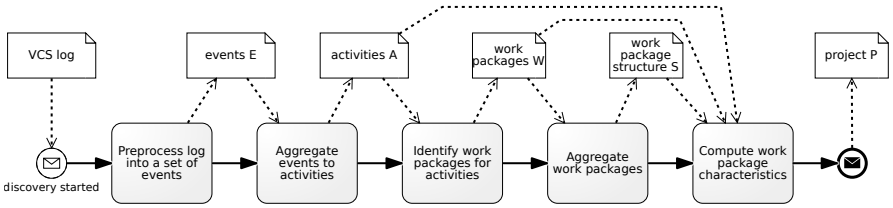


Fig. 3.2: Project discovery technique overview as BPMN process model.

3.3.2.1 Step 1: Preprocessing.

The first step is to transform raw logs of version control systems (which might be grouped by commits) into a list of events as specified in Definition 3.1. This step is easily done by replicating the information on commit level to be contained in the events. The output is a set of events E.

3.3.2.2 Step 2: Aggregating events to activities.

Given the set of events E that we gathered from a version control system, the next step is to identify the activities to which the events belong. Note that we do not know the activities of the project in advance, but need to infer them based on the events. Each event affects a single file in the file hierarchy.

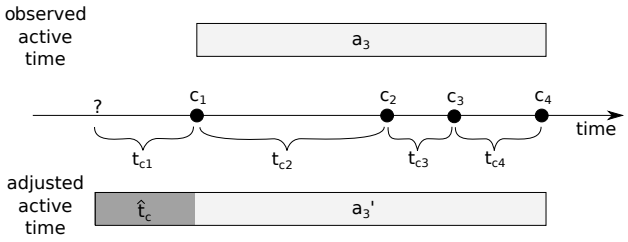


Fig. 3.3: Adjustment of activity start time α.

Based on assumption A2, we are interested in activities conducted in a work package, that is, we filter for the events that are contained in the given file or its children. For every file f of interest, we select the set of events affecting the file or its children as $E_f = \{e \in E \mid f = f(e) \vee f \in ancestor(f(e))\}$. The task is then to find the activities which emitted the set of events E_f. We rely on assumption A3, which states that during an activity, we expect multiple commits. Assumption A3 allows us to conclude that if we do not observe commits for a longer period of time, there is no activity being performed in the work package.

To this end, we adopt the abstraction technique by Baier et al. (2014) and allow the domain expert to formulate rules for aggregating events to activities based on boundary conditions. Assuming that people frequently commit their progress (A3), we can specify a boundary condition based on the temporal distance to previous events. For example, we can specify that a time period of seven days without a commit is a boundary condition. As the result, we obtain the mapping from events to these activities, which we call $\gamma_f : E_f \rightarrow A_f$ in the remainder of the paper. The set of discovered activities identified for the work package based on given boundary conditions is then $A_f = \{a \mid e \in E_f, \gamma_f(e) = a\}$. We also define the inverse mapping, that is, the mapping from an activity to its events as $\gamma_f^{-1} : A_f \rightarrow 2^{E_f}$.

With the events mapped to activities, we need to find the temporal boundaries of the target activities. That is, we define the functions α and ω for each activity. The challenge here is that we do not know when an activity actually started, because the start of the activity is not recorded in the VCS. We can only observe the time of the first commit in that activity, but commits usually mark progress of an already running activity.

To address the challenge of missing start times, we impute the missing start time by prepending the expected active time $\hat{t}_c$ before a commit, as illustrated by Figure 3.3. This notion assumes that project participants commit their work progress after a certain amount of time. However, we cannot compute $\hat{t}_c$ by looking at the average commit rate in a work package, because this average is based on busy periods and idle periods. We need to factor out the idle periods in the computation of this measure. We know the end time of the activities, as the last commit marks the completion of work. Therefore, each activity a based on given boundary conditions has the associated end time $\omega(a) = \max(\{ts(e) \mid e \in \gamma_f^{-1}(a)\})$. Further, we write the first event's timestamp of an activity as the function $\alpha'(a) = \min(\{ts(e) \mid e \in \gamma_f^{-1}(a)\})$. Then, we define $c : A_f \rightarrow \mathbb{N}^+$ as the number of commits in one activity, formally $c(a) = |\{C \mid e \in C \wedge \gamma_f(e) = a\}|$.

With this information the expected active time between commits $\hat{t}_c$ is given as follows.

$$\hat{t}_c = \frac{\sum_{a \in A_f} (\omega(a) - \alpha'(a))}{\sum_{a \in A_f} (c(a) - 1)} \tag{3.1}$$

We assume that there is at least one activity spanning over at least two commits, i.e., $\exists a \in A_f \mid c(a) > 1$. Translated to our boundary condition, this assumption is that there is at least one week in each work package, in which there were at least two commits made. Otherwise, we set $\hat{t}_c$ to 0 for the current file f due to lack of information.

Given the expected active time between commits $\hat{t}_c$, we can finally adjust the start time of each activity. Therefore, we set the associated start time for each activity as $\alpha(a) = \alpha'(a) - \hat{t}_c$. That is, we subtract the expected active time from the first commit's timestamp.

We apply Step 2 to all files f in the file tree to get A_f. For the remainder of this paper, we define the function $\psi : A \rightarrow F$ that contains the mapping information of the discovered activities to their originating files. Finally, we set the activities A in the project to be the union of the activity sets per file $\bigcup_{f \in F} A_f$.

3.3.2.3 Steps 3 and 4: Mapping activities to work packages and aggregating.

Once activities have been identified, we want to climb to the next abstraction layer: the work packages. Assumption A1 allows us to specify a one-to-one mapping $\kappa : F \rightarrow W$ between files in the file tree structure and work packages. More precisely,

we construct the set of work packages W isomorphic to the set of files F, such that the *Parent* relation is preserved in the work package structure S relationship.

The mapping β of activities to work packages is simply $\beta(a) = \kappa(\psi(a))$. That is, the corresponding work package of the activity that was discovered for a file. In this way, we provide an activity based view on work packages, and we can aggregate on each level in the file system to see active periods of the corresponding hierarchy level.

3.3.2.4 Step 5: Computing work package characteristics.

In this final step, we compute measures of interest for the discovered work packages. First, we obtain the temporal boundaries of a work package by the functions α and ω of the associated activities.

Let $\beta^{-1} : W \to 2^A$ be the inverse of the mapping function β of the project. The start and end time of a work package (α_W and ω_W) are functions from work packages to timestamps. The start time is defined as $\alpha_W(w) = \min(\{\alpha(a) \mid a \in \beta^{-1}(w)\})$, and the end time function of work packages ω_W is analogously defined using the maximum of the end times $\omega(a)$ of the activities. We call the duration of a work package τ that is the difference between ω_W and α_W.

Moreover, we are interested in the ratio of active working periods (i.e., the time spans of activities) to the total work package duration. This quantity helps to estimate the average work intensity in a work package.

Definition 3.4 (Coverage) The coverage χ of work packages by activities is a function $\chi : W \to [0, 1]$ and is defined as follows.

$$\chi(w) = \frac{\sum_{a \in \beta^{-1}(w)} (\omega(a) - \alpha(a))}{\tau(w)} \tag{3.2}$$

With this final step, we lifted the information hidden in low level events to a high-level Gantt chart perspective, with which project managers are familiar. In the following, we compare our technique to existing process mining approaches.

3.4 Evaluation

In this section we evaluate our solution to the project mining problem, and show results for the example presented in Section 6.2.

3.4.1 Experimental setup

We evaluate our technique by a visual perspective and by comparison to possible different approaches. To this end we implemented our technique as a prototype. We

used JAVA as a programming language to code the logic of our technique. For the visualization part we made use of custom SWT widgets provided by the Nebula Project[1]. Our program can deal with logs from Subversion (SVN) (Pilato et al., 2008) and Git (Torvalds and Hamano, 2010), but it can be extended to other version control systems by providing an implementation of the preprocessing step discussed in Section 3.3.2. We ran the software in an Intel®Core ™ i5-4570 CPU @ 3.20 GHz x 4 machine with 15.6 GiB of RAM and Linux kernel 3.13.0-46-generic 64-bit version.

3.4.2 Input data description

We tested our prototype with real-world log data taken from the SHAPE project. Logs were exported from the SVN and Git repositories of different projects. They come from the railway domain and describe engineering processes. Documentation stored in the repositories consists of manually produced text files, diagrams, and files coming from proprietary tools that are typically used in the domain.

We will display results for the SVN log that describes the process oriented project for SHAPE. Data span over one year, going from January 2014 to January 2015. This time window covers the phases of project definition and planning, and a part of the project execution. In the first phase, feasibility of the project was studied and budget, schedule and resources were determined. Proposal submission marked the end of this phase. The second phase started with a kickoff meeting in October 2014 and is still ongoing.

The total number of participants who actively contributed to the work packages stored in the SVN repository was 8 people in the beginning, with new resources joining the project after the kickoff date. The total number of files and directories counts up to 156 objects and 226 overall commit events. The total number of extracted change events after preprocessing (i.e. atomic changes on all the files) was 453.

The last part of the log data contains the task *Define example*, introduced in Section 6.2.1. For our showcase we assume that this task is contained in a work package named *example*.

3.4.3 Output data

To monitor the project execution, we visualize the work progress that was done for each work package. Monitoring is performed by managers who want to have an overview on the project (which work packages are done, when and for how long, and where idleness or congestion occurs). Gantt charts offer a graphical representation for displaying schedules and jobs that were done on the various work packages Wilson (2003) in a way that can easily be communicated to managers.

[1] https://www.eclipse.org/nebula/

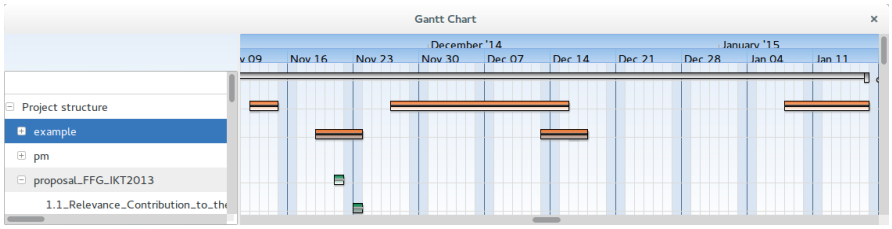


Fig. 3.4: Data representation from our tool. Atomic events are drawn as dot with a minimal duration and different color per commit.

Figure 3.4 is a screenshot of how our tool presents the data. The tree structure on the left represents the *Parent* relation in the file tree. Events belonging to the same commit have the same color. On the top part of the chart we can see the result of merging events to activities with our aggregation method. Here we have merged the events of the example scenario on their highest abstraction level. The chart shows the three main activities and the idle times between them. On the other hand, in correspondence to expanded directories we show only their status before the aggregation. That is, every time a directory is fully expanded we apply a disaggregation into the corresponding activities. In this way, we can also show the finest granularity of work, i.e. the atomic events.

3.4.4 Project Analysis

Next, we apply our algorithm to the example case from Table 3.1 and check how it helps to identify work packages. The data is aggregated according to our threshold of seven days. We can observe three groups of events being temporally close to each other according to our threshold. That is, we expect the event data to be grouped into three activities.

The second step of our algorithm takes care of adjusting the starting time of the activities. Furthermore, we vertically order the events and activities in the Gantt chart according to the directory structure to show the mapping from the objects on the Gantt chart to each work package in the tree structure. The last step, computing work package characteristics is done automatically when we collapse a node on of tree.

Figure 3.5 shows a comparison of the case when we do not implement the activity adjustment to the case which adjusts it. In the upper part, activity boundaries are based only on the first commit time that we see in the data. In the lower part, we observe that the start times were adjusted by approximately one day. The tool automatically adjusted the start time of the activities. As a consequence, the coverage factor increases because we expect that there was more work than what we observe by only considering the first commit time.

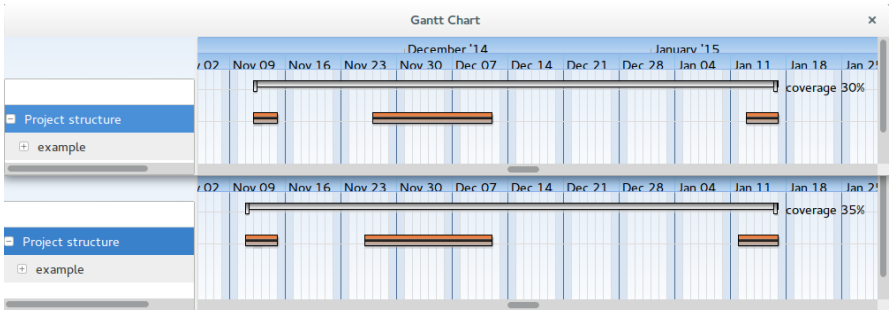


Fig. 3.5: Before and after the prepending the expected time before commit. Coverage factor increases when we adjust the starting times of the activities.

3.4.5 Coverage tests on available open projects

Finally, we apply our approach on different input data from open source projects. We are interested in exploring how the coverage factor varies in different existing projects. Hence, we take the work package w as our controlled variable and set it to the highest level of aggregation. Then, we analyze each project of the data set and observe the dependent variable $\chi(w)$. Another variable of interest is the $\hat{t}_c$ since it gives an idea of the average work speed (commit frequency) during active times.

Table 3.2: Coverage results for different open source projects

Log File name	**Duration** Days	**Idle periods** Number	**Files** Number	**Commits** Number	$\hat{t}_c$ Hours	$\chi(w)$ %
MiningCVS	24	0	89	63	9	100
Whitehall	1279	6	6539	15566	2	95
Petitions	834	17	1562	914	13	59
Study	624	13	7501	736	11	58
The Guardian	1667	59	12889	621	30	44
Book	414	15	154	592	5	32
Papers	1859	55	1791	649	20	30
Requirements	771	22	505	231	17	21
Yelp	206	6	24	54	20	20
Adobe	1076	13	356	237	24	15

The data we used stems from the following projects. *MiningVCS* is our tool. It consists of daily commits and was developed over 24 days. *Whitehall* is the code name for the Inside Government project, which aims to bring Government departments online in a consistent and user-friendly manner. *Petitions* is a Drupal 7 code base used to build an application on "We The People", the platform to create and sign petitions of the White House. *Study* is an SVN log about Healthcare domain, taken

from SHAPE. *The guardian* is the log data from the Git repository of the well-known British national daily newspaper. *Book* is the log data that describes the writing of the book Crypto 101 by Laurens Van Houtven, taken from Git. *Papers* is taken from SHAPE project for building a paper archive. *Requirements* log data is taken from the the Git repository of OpenETCS and belongs to the railway domain. *Yelp* is the main Github page of Yelp were they showcase all their projects. *Adobe* is the Adobe Github Homepage v2.0, which is a central hub for Adobe Open sources projects.

Table 3.2 shows our experiments on the above-mentioned logs and the corresponding coverage factors. Projects that score a high coverage factor are characterized by continuous work. This can be further seen by looking at their average idle times $\bar{t}_{Idle}$. Let n_c be the number of commits per work package. We compute the average idle time as follows.

$$\bar{t}_{Idle} = \frac{\tau - n_c \cdot \hat{t}_c}{n} \ , \ n > 0 \tag{3.3}$$

where n is the number of idle times in the work package. If $n = 0$, then we trivially assign $\bar{t}_{Idle} = 0$, because there were no break periods over time.

Applying the formula to the above projects, we can observe how projects with a higher coverage factor have actually low values of $\bar{t}_{Idle}$. For instance, *Whitehall* scores a $\bar{t}_{Idle}$ of 11 days, whereas *Adobe* scores a $\bar{t}_{Idle}$ of 36 days. This supports the usage of the coverage factor χ as an indicator for work package time utilization.

3.5 Discussion

In this section we compare our method to other alternatives for mining data out of logs and interpret our results.

Well known tools that are used in academia and practice include ProM van Dongen et al. (2005) and Disco[2]. Both tools require input data to be in the XES (Verbeek et al., 2010) format. Thus, we convert our data from the *Define example* case into XES. To show events per objects of the project structure, we choose the file path as the *caseId*. To flatten the logs we extract all the file paths and build a mapping from each file to the set of changes done to it.

Figure 3.6 depicts the results of the Dotted chart plugin of ProM applied to our log data. Also here, we observe different changes of each file of the repository. While the files and their corresponding events are shown, the plugin does not allow to rearrange the data in order to understand the file structure, nor does it allow to perform any kind of aggregation or connection between data, to observe them from a higher level perspective.

Figure 3.7 shows the results from mining our log data with the Disco tool. Here we can see a plot that displays the events that happen over time. The plot has some peeks in correspondence to active times of the *example* work package. They can be grouped in three clusters: an initial cluster with a few amount work, an intermediate

[2] http://fluxicon.com/disco/

cluster with the most significant part of the work, and a final cluster that again is not very active. In this way, clusters can be associated to activities. As a drawback, when the number of work packages and activities increase, the number of peeks grows and generate identifying clusters of activities by look at active (or idle) times becomes unworkable.

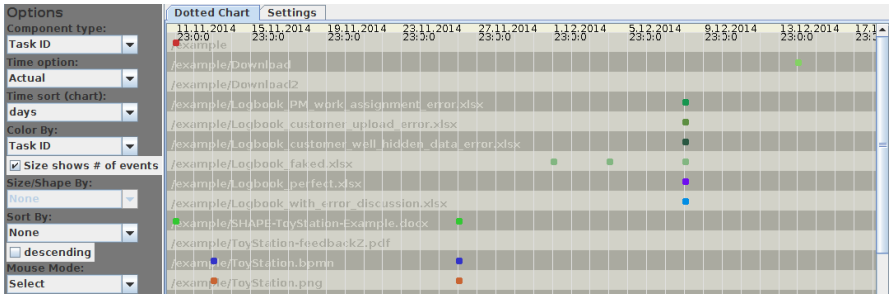


Fig. 3.6: Dotted chart from ProM

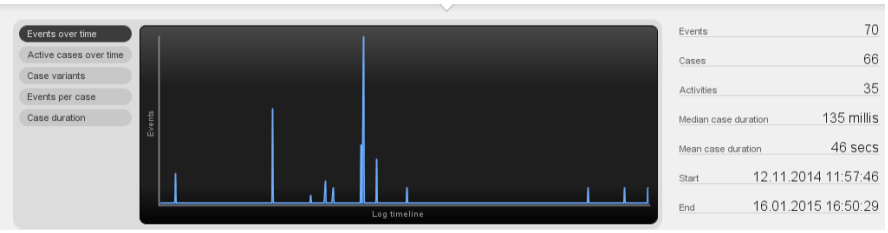


Fig. 3.7: Chart from Disco plotting the events over time.

Our approach to mining the work progress of project-oriented business processes complements these techniques with metrics and a corresponding visualization that is informative to managers.

3.6 Conclusion

In this paper we addressed the problem of mining and visualizing project-oriented business processes in a way that is informative to managers. We define an approach that takes VCS logs as input to generate Gantt charts. Our algorithm works under the assumptions that repositories reflect the hierarchical structure of the project, each work package is contained in a corresponding directory and project members commit

their work regularly during active working times. The approach was implemented as a prototype and evaluated based on real-world data from open source projects.

In future work, we aim to extract further details of the VCS logs in order to calculate metrics that approximate the work effort. We plan to investigate on how the project mining approach is affected by project characteristics. Furthermore, we want to utilize statistical methods to better estimate the boundaries of the activities and work packages. Finally, we have already incorporated feedback from managers and plan to extend these to full user studies.

Chapter 4
Article 2: Uncovering the Hidden Co-Evolution in the Work History of Software Projects

Authors:

Saimir Bala, Kate Revoredo, João Carlos de A. R. Gonçalves, Fernanda Baião, Jan Mendling, and Flavia Santoro

Published in:

Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings

Abstract:

The monitoring of project-oriented business processes is difficult because their state is fragmented and represented by the progress of different documents and artifacts being worked on. This observation holds in particular for software development projects in which various developers work on different parts of the software concurrently. Prior contributions in this area have proposed a plethora of techniques to analyze and visualize the current state of the software artifact as a product. It is surprising that these techniques are missing to provide insights into what types of work are conducted at different stages of the project and how they are dependent upon another. In this paper, we address this research gap and present a technique for mining the software process including dependencies between artifacts. Our evaluation of various open-source projects demonstrates the applicability of our technique.

4.1 Introduction

Project-oriented business processes play an important role in various industries like engineering, health care or software development Bala et al. (2015). Such processes are characterized by the fact that work towards a predefined outcome involves complex tasks executed by different parties. Typically, these processes are not supported by a process engine, but their status is fragmented over different documents and artifacts. This is especially the case for software development processes: the expected outcome is the release of a new software version, but the different project members collaborate with tools like version control systems that are only partially aware of the work process.

A key challenge for project-oriented business processes like software development is gaining transparency of the overall project status and work history. Literature has recognized that analyzing the evolution of business process artifacts in projects can help obtaining important clues about the project performance in terms of time (Beheshti et al., 2016), cost (Voinea and Telea, 2007) and quality (Lindberg et al., 2016). This is addressed by functionality of version control systems (VCS) to track versions and changes of informational artifacts like source code and configuration files. While prior research has presented various perspectives for analyzing software artifacts, e.g. Bani-Salameh et al. (2016); Robles et al. (2014); Mittal and Sureka (2014); Weicheng et al. (2013), there is a notable gap on the discovery of dependencies in the work history. For these reasons, project managers often lack insights into side effects of changes in large software processes.

In this paper, we address this research gap by building on partial solutions from the separate fields of mining software repositories and process mining. More specifically, we develop a technique that uncovers non-hierarchical work dependencies which we call *hidden co-evolution*. This technique extracts the labeled work history from VCS repositories and identifies dependencies beyond simple hierarchical containment. In this way, we help the project manager to spot dependencies in the co-evolution of work histories of different information artifacts. Our technique has been implemented and evaluated using data from a diverse set of open source projects.

The paper is structured as follows. Section 6.2 describes the research problem along with its requirements and summarizes insights from prior research. Section 6.3 presents our approach in detail. Section 4.4 shows a prototypical implementation and evaluates its applicability both in a use case scenario and on real world projects from GitHub. Section 5.6 concludes the paper.

4.2 Related Work

This thesis follows the DSR paradigm (Peffers et al., 2008). In this section, we describe the research problem in more detail and define requirements for a solution. Against these requirements, we analyze related work.

4.2.1 Problem Description

In this paper, we focus on a specific class of project-oriented business processes, namely software development processes. These processes share some common characteristics. First, they involve various resources with different roles. In the simplest case, we can distinguish *project managers* and *project participants*. Project managers are responsible for managing the development process and supervising the work of the project participants, who in turn are responsible for specific work tasks. Second, such processes are usually subject to constraints in terms of cost, time and quality, which is mostly associated with the performance of each of the work tasks. Third, the project participants work on a plethora of artifacts, which are logically organized in a hierarchical structure, with complex interdependencies among them. Given these characteristics, it is the goal of the project manager to organize the software development process in such a way that the work on different files and tasks reflects the complex interdependencies, the constraints and the available participants. Therefore, it is important for the manager to understand the *work history* of the process in order to monitor the progress systematically.

Table 4.1: 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()

Software tools like VCSs do not provide direct support for monitoring work histories, but they provide a good starting point by continuously collecting event data on successive versions of artifacts. Table 4.1 shows an excerpt of log data, where the columns, from left to right, indicate the commit identifier, the project participant who committed the changes, the commit date, the comment written by the project participant and the files affected and the change performed[1]. In order to understand the work history and dependencies based upon such data, we identify three major requirements:

R1 (Extract the work history): Discover the process of how artifacts evolve in the project as a *labeled* set of steps. This requirement is difficult because the version changes of a commit in relation to a single file do not directly reveal which type of work has been done. Both commit messages and edit characteristics might inform the labeling.

R2 (Uncover Work-Related Dependencies): Identify that certain work in one part of the project is connected with work in another part. This requirement is difficult because such dependencies might not only exist between files that reside in the same directory. For example, a change in a source code file might have the side effect of triggering work on a configuration file. We refer to this as *co-evolution* of these files.

R3 (Measure Dependencies): Determine how strong the co-evolution of different artifacts is. This requirement is difficult because measures of *strength* of dependencies and on the *distance* of dependent artifacts have to be devised.

4.3 Conceptual Approach

We propose a technique to extract and represent the work history and the dependencies among artifacts of a project-oriented business process. The technique takes as input a VCS log and produces analysis data that describe the evolution of the artifacts, along with metrics about their distance and their similarity in terms of work. The process is depicted in Figure 4.1 and consists of three successive steps towards extracting hidden work dependencies from VCS event data. The method works under three main assumptions. First, we assume a *meaningful tree structure*, i.e. the project participants organize the files in a representative hierarchy (e.g., spatially separating documentation from testing into different folders). Second, project participants perform *regular commits* in the VCS. Third, project participants write *descriptive comments* that allow other members to understand the changes.

The first step of the technique is the preprocessing of the VCS log received as input. The main goal of this phase is to generate a set of events and store them into a database. Second, we obtain different views on the stored events. In particular, we are interested in observing *i)* all the commits that affected the files over time; *ii)* the amount of change brought by the commits to the files; and *iii)* the users who

[1] cf. unified diff format https://git-scm.com/docs/git-diff

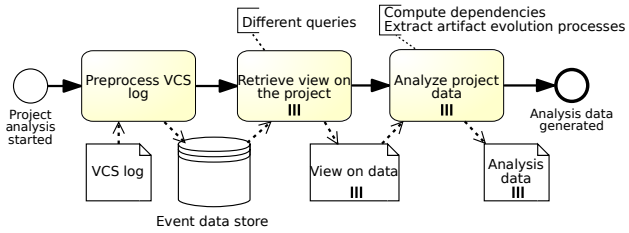


Fig. 4.1: Approach for generating analysis data from VCS logs

issued such commits. The third phase is responsible for considering the different perspectives defined by the project manager and through the generated views extract the necessary knowledge. In the following, we detail the formal concepts and the algorithm of our technique.

4.3.1 Preliminaries

As the objective of our technique is to uncover hidden work dependencies, we define the fundamental concepts required to capture them. Work is reflected by *artifacts*, e.g., word documents, spreadsheets, code, etc. Artifacts are leaves in the file tree hierarchy (with directories being special type of non-leaf files). Artifacts evolve over time, while project participants contribute their changes. Each change is an *event* that happens to an artifact in a single point in time. Events can be abstracted into *aggregated events* that allow a coarser grained view on the history. The history of the changes of an artifact over a time interval at a given level of abstraction is referred to as *artifact evolution*. Similar artifact co-evolution establishes a *dependency* between two artifacts.

A software product is subdivided into files and directories. In this work, we consider directories as special type of files which are parents of other files. Formally, let F be the universe of files in a software development project. Files are organized in a file tree. Therefore, each file $f \in F$ has one parent file. The only file without a parent file is the *root* file. We capture this information in the parent relation $Parent : F \times F$. For example, let $f_p \in F$ be the parent of file $f_c \in F$, then $(f_p, f_c) \in Parent$. An *artifact* is a file that is not a parent file, i.e. a file f_a is an artifact if $\forall_{f \in F}(f_a, f) \notin Parent$.

When project participants do a certain amount of work and want to save their current progress, they commit the changes to the VCS. We define changes on artifacts as the *events* of interest on the lowest granularity.

Definition 4.1 (Event) Let E be the set of events. An *event* $e \in E$ is a five-tuple (f, ac, ts, k, u), where

- $f \in F$ is the affected artifact of the event.
- $ac \in AC = \mathbb{N}$ is the amount of change done in the artifact.

- $ts \in TS = \mathbb{N}$ represents a unix time stamp marking the time of the event occurrence.
- $k \in \Sigma^*$ is a comment in natural language text.
- $u \in U$ is the project participant responsible for the change.

For event $e = (f, ac, ts, k, u)$ we overload f, ac, ts, k and u to be used as accessor functions. For example, f is the function $f : E \rightarrow F$ mapping an event to its affected artifact.

In some situations, it can be interesting to have a higher level overview of the changes done to a particular artifact. In this case, an aggregation of events related to this artifact in an interval of time can be performed. The time window for the aggregation, henceforth denoted as tw_{agg}, must be defined, i.e. the size of the time interval. For instance, a time window for aggregation can be a day. Thus, all events occurring for an artifact in the same day will be aggregated. An *aggregated event* is defined as follows:

Definition 4.2 (Aggregated Event) An *Aggregated Event* for tw_{agg} ($AE_{tw_{agg}}$) is a five-tuple (f, aac, ats, ak, au), where

- $f \in F$ is the affected artifact in the set of events being aggregated.
- $aac \in AAC = \mathbb{N}$ is the aggregate amount of change done in the artifact for tw_{agg}. It is calculated by summing the amount of changes done in each of the time aggregated.
- $ats \in ATS = \mathbb{N}$ represents an aggregate time of the unix time stamp of the events being aggregated.
- $ak \in \Sigma^*$ is the concatenation of the comments presented in the events being aggregated.
- $au \subseteq U$ are the project participants responsible for the changes in tw_{agg} being aggregated.

The set of aggregated events for a particular artifact defines how this artifact evolves over time. Considering an interval of analysis, henceforth denoted as ia, we define artifact evolution as follows.

Definition 4.3 (Artifact Evolution) *Artifact evolution* is the process describing how the file f changed over an interval of time ia, i.e., a set of labeled tuples $A_{evo}(f) = \{(t, a, l) | e \in AE_{ia}, f = f(e), t = ats(e), a = aac, l = ak(e)\}$ chronologically ordered.

Note that artifact evolution represents the changes that happened to a file over time. Thus, we can build the time series of a file f as the vectors of changes $\mathbf{X_f} = (a_1, ..., a_n)$ in the time window $tw_{agg} = [t_1, t_n]$, with a_i being the sum of the changes of f in of the aggregated intervals t_i of the time window tw_{agg}.

We measure the dependency between two files f_a and f_b in terms of their *degree of co-evolution* as follows.

Definition 4.4 (Degree of Co-Evolution) Given two files f_a and f_b, the *degree of co-evolution* $\chi : F \times F \rightarrow [0, 1]$ is a similarity function of the respective time series.

In this paper, we fix $\chi(f_a, f_b) = |\sigma(\mathbf{X_{f_a}}, \mathbf{X_{f_b}})|$, where σ is the correlation function of the two vectors $\mathbf{X_{f_a}}$ and $\mathbf{X_{f_b}}$.

The way files are kept in the directory structure establishes an inherent relationship among files being stored close to each other in the hierarchy. For instance, files serving the same purpose are stored close to each other in the file system. Hidden work dependencies are expected to happen between artifacts that are distant in the file structure. We measure this distance as the length of the shortest route connecting two files in the file tree. We adapt the notion of path from Gubichev et al. (2010) to our file tree. Given a file f, the path to the root node can be obtained by navigating the $Parent$ relationship up to the root file. The path p from f_a to the root f_r is the set of parent files encountered along such route. i.e. $p(f_1, f_r) = \{(f_1, ..., f_k, f_{k+1}, ..., f_r)\}$ such that for any k, $(f_{k+1}, f_k) \in Parent$. The length of the path is the cardinality $|p|$ of the set. The shortest path between two files f_a, f_b in a tree passes through the Least Common Ancestor (LCA) Bender and Farach-Colton (2000). This is equivalent to considering the paths from the single files to the root node $p_a = p(f_a, f_r)$ and $p_b = p(f_b, f_r)$ minus their intersection $I_{p_a,p_b} = \{p(f_a, f_r) \cap p(f_b, f_r)\}$. Thus, we define the *file distance* as the length of the shortest path between two files f_a and f_b as follows.

Definition 4.5 (File Distance) The distance $d : F \times F \rightarrow \mathbb{N}$ between two files belonging to the same directory structure is defined as the number of nodes in the minimum path connecting the two files in the project file tree: $d(f_a, f_b) = |p_a| + |p_b| - 2 * (|I_{p_a,p_b}|)$.

4.3.2 Hidden Dependencies Discovery Algorithm

We are focused on finding interesting hidden work dependencies. These dependencies are typically reflected by changes that happen to couples of allegedly unrelated files during their evolution. This section details the procedure that implements the technique outlined in Figure 4.1.

Algorithm 1 presents the steps required to explicate such hidden dependencies. The procedure `PreprocessLog`($\mathcal{L}$) in line 2 takes as input a VCS log $\mathcal{L}$ structured as in Table 4.1 and parses out work events at the granularity of line changes. These events are then stored into an event data storage. Events parsed from VCS logs contain rich information about multiple aspects of the work they reflect. In order to represent all these different aspects, we devised the entity-relationship data model. Hence, we are able to store all the information that is possible to obtain after parsing the VCS log. Furthermore, this step allows the user to obtain simple information, such as statistics on the project, already at an early stage of the procedure. The output of the `PreprocessLog`($\mathcal{L}$) step results in the storage of all the events E into a database.

Next, the iterative call of the procedure `RetrieveView`(E, query) in line 3 performs several querying the data storage containing the set E. For example, a possible query can obtain all the comments associated to each change of a specific file. To obtain information on the evolution of files, we query the database for the

changes of all the files within a user defined time interval tw_{agg}. In general several time frames can be chosen, each of them producing a *view* V on the data, i.e., a set of aggregated events chronologically sorted within tw_{agg}. For example, users may be interested in artifact-views aggregated by day, by month, etc. Multiple *views* are possible by defining them in the queries parameter. We collect these views into a set $\mathcal{V} = \bigcup_{\text{queries}} V$.

Algorithm 1: Generate project analysis data

Input : A VCS log $\mathcal{L}$

Output : A set of triples $\{(Dist, Stories, D_{co-evo})\}$, artifact evolutions, and dependencies

Data : E event set, $\mathcal{V}$ views set, $AnalysisData = \{(Dist, Stories, D_{co-evo})\}$, degree of co-evolution threshold γ, file distance threshold δ, user defined queries queries

1 $Files \leftarrow \emptyset$, $Stories \leftarrow \emptyset$, $TimeSeries \leftarrow \emptyset$, $AnalysisData \leftarrow \emptyset$, $\mathcal{V} \leftarrow \emptyset$, $A_{evo}(f) \leftarrow \emptyset$;
/* Preprocess VCS log */
2 $E \leftarrow$ PreprocessLog($\mathcal{L}$);
/* Retrieve views on the project */
3 **for** *i from* 1 **to** |queries| **do** $\mathcal{V} \leftarrow \mathcal{V} \cup$ RetrieveView(E, queries[i]);
/* Analyze project data */
4 **foreach** *view* $V \in \mathcal{V}$ **do**
5 **foreach** *aggreagated event* $ae \in V$ **do**
6 **foreach** $f = f(ae), t = ats(ae), a = aac(ae), l = aak(ae) \in ae$ **do**
 /* Construct the artifact evolution set for the file */
7 $A_{evo}(f) \leftarrow A_{evo}(f) \cup \{(t, a, l)\}$;
 /* Construct the process using story mining */
8 $Stories \leftarrow Stories \cup (f,$ StoryMining(l)$)$;
 /* Collect files and time series */
9 $Files \leftarrow Files \cup \{f\}$;
10 $TimeSeries(f) \leftarrow$ construct time series from $A_{evo}(f)$;
11 **end**
12 **end**
13 **foreach** *pair of files* $i, j \in Files$ **do**
 /* Compute degree of co-evolution */
14 $coEvoDegree \leftarrow \chi(TimesSeries(i), TimeSeries(j))$;
 /* Compute file distances */
15 $distance \leftarrow d(i, j)$;
 /* Select based on user defined thresholds */
16 **if** $coEvoDegree > \gamma$ **then** $D_{co-evo} \leftarrow D_{co-evo} \cup \{coEvoDegree\}$;
17 **if** $distance > \delta$ **then** $Dist \leftarrow Dist \cup \{distance\}$;
18 **end**
19 $AnalysisData \leftarrow AnalysisData \cup \{Dist, Stories, D_{co-evo}\}$;
20 **end**
21 **return** $AnalysisData$;

The step in line 4 starts an iteration over the views set $\mathcal{V}$. Here is where we collect the analysis data that are returned by the algorithm. For each of the aggregated artifacts contained in a view V, we retrieve the information necessary to compute the *degree of co-evolution* between pairs of files and their *file distance*. First, we

construct the artifact evolution of all the artifacts present in $ae \in V$. Note that an aggregated event $ae \in V$ is a record obtained from a view on the project which is composed, among other attributes (e.g., file, time, amount of change), by the comment associated to the specific change. Comments describe multiple changes executed on the file, i.e. they describe a *story* of the artifact. Stories associated to each file are collected and the corresponding labels are chronologically ordered. These file stories are then input to the StoryMining technique Gonçalves et al. (2011). Story Mining was designed to receive as input a story freely written by the participants, describing their work in a particular business process. As an output, the *actors* and the process *activities* executed by them are extracted. Our technique is concerned with the stories of the files. Therefore, they are the actors of the story mining, and the resulting business process consists of the steps describing their evolution process. We collect the resulting processes in the step in line 8. The step in line 10 is concerned with the construction of a time series from the set of artifact evolutions A_{evo} computed in line 7. Specifically, this step gathers the values of the changes of each of the artifact f in A_{evo} and records them in $TimeSeries(f)$.

After all the aggregated events ae have been explored, the algorithm moves on to computing the metrics (lines 13–18). In this loop, the algorithm iterates through all the pairs of files. For each pair, the *degree of co-evolution* and *artifact-distance* metrics are computed according the Definition 4.4 and Definition 4.5, respectively. These two measures are collected only if their values are above the user defined thresholds γ and δ. After the loop is over, the two measurements and the stories mined with the StoryMiner are stored in $AnalysisData$.

Finally, after iterating over all the user defined views, the algorithm returns the $AnalysisData$ collection which can now be further inspected and analyzed in more detail, as we show next with an example.

4.3.3 Proof of Concept

Let us consider the following example of a software development process. It contains 10 files arranged hierarchically as depicted by the file tree in Figure 4.2. At the first level of the file tree there is the README.md file which describes the project. The software product in our case is called *running example* and is contained under the f_3 directory. The product consists of an example for software developers who want to organize their projects according to a predefined structure. The project has 21 commits over 10 days.

An excerpt of the VCS log for this project was illustrated in Table 4.1 above. The project managers are interested in understanding the work process done by project participants in each of the files and whether there is some hidden work dependency. We show how our technique meets the requirements by applying each step to this project and discussing the outcomes.

Let us suppose we have preprocessed our data and have the events set E already stored in a database. Then $\mathcal{V}$ is obtained by querying the data and aggre-

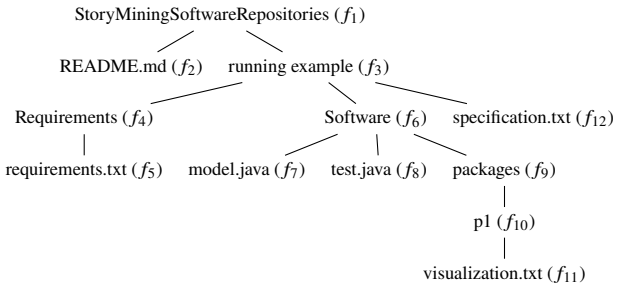


Fig. 4.2: File tree describing the file structure in our scenario of use.

gating them by day. Then, the *parent* relation is $Parent = \{(f_1, f_2), (f_1, f_3), (f_3, f_4), (f_3, f_6), (f_3, f_{12}), (f_4, f_5), (f_6, f_7), (f_6, f_8), (f_6, f_9), (f_9, f_{10}), (f_{10}, f_{11})\}$. Next, we compute the artifact evolution of for each artifact. For example, the artifact evolution of file REAMDE.md (f_2) limited on the information from Table 4.1 is A_{evo} = { (2017-01-31, 1, *Create readme file*), (2017-02-01, 3, *Add a license*), (2017-02-02, 1, *Updated the requirements*)}. The resulting process from the story mining algorithm is shown in Figure 4.3.

Fig. 4.3: Example of business process showing the artifact evolution

Next, we calculate the metrics. The dependencies are computed in the steps enclosed in lines 13–18 of Algorithm 1. E.g., the artifacts `README.md` (f_2) and `test.java` (f_7) appear in the $TimeSeries$ collection as the vectors $\mathbf{X_{f_2}} = (1, 3, 1, 0)$ and $\mathbf{X_{f_7}} = (0, 0, 0, 2)$. We use the Pearson correlation between the to vectors $\sigma(\mathbf{X_{f_2}}, \mathbf{X_{f_7}} = -0.66)$ and take its absolute value as degree of co-evolution $\chi = |\sigma|$. Therefore, the *degree of co-evolution* between the considered artifacts is $\chi = 0.66$. The *file distance* is the length of the route from f_2 to f_7, i.e $d(f_2, f_7) = \{(f_2, f_1), (f_1, f_3), (f_3, f_6), (f_6, f_7)\}$. Therefore, the file distance between `README.md` and `test.java` is $d(f_2, f_7) = 4$.

4.4 Evaluation

In this section, we show the applicability of our technique to project-oriented business processes and its effectiveness in uncovering work dependencies. With respect to the requirements formulated in Section 6.2, we evaluate against requirements R2 and R3 in Section 4.4.1 and against requirement R1 in Section 4.5.

Table 4.2: Evaluation of real world projects. Respectively the thresholds are: χ^L if $\chi < 0.3$, χ^H if $\chi > 0.7$ low and high degree of co-evolution; d^L if $d \leq 2$, d^H if $d > 2$ respectively low and high distance.

Project	Commits	Files	χ^H	χ^L	(d^L, χ^L)	(d^L, χ^H)	$(d^H, \chi^L,)$	(d^H, χ^H)	$\overline{\lvert p_f \rvert}$	$max(\lvert p_f \rvert)$	$\lvert A_{evo} \rvert$	$\overline{d}$	$max(d)$
mwaligner	21	9	37	7	6	30	1	7	1.11	2	2.40	0.94	3
Biglist	202	15	22	90	31	18	59	4	1.47	3	2.76	1.20	5
camundaRD	11	15	74	26	0	25	26	49	2.18	4	2.05	2.03	7
graphql	256	30	89	357	121	89	236	0	1.40	2	3.18	1.11	4
jgitcookbook	135	89	773	2866	505	289	2361	484	6.93	8	1.33	2.68	14
mysqlpython	749	168	2288	11571	742	591	10829	1697	2.59	7	1.65	2.52	11
gantt	23	228	7006	14343	386	3480	13957	3526	3.30	4	1.71	2.16	7
facebookjavasdk	38	293	16478	26092	2017	16311	24075	167	6.21	8	4.78	5.58	13
caret	864	432	15366	60874	9538	14785	51336	581	3.01	4	3.15	1.60	7
operationcode	1114	1053	84024	444605	2291	5537	442314	78487	4.27	8	2.01	4.85	15

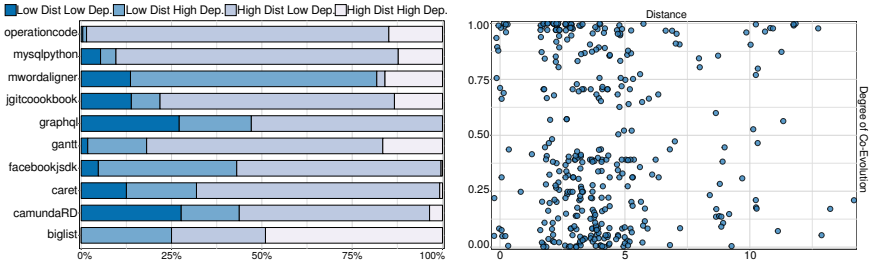
We implemented our techniques as a prototype[2] and used it on 10 real world software projects with different sizes. The input of our program is a VCS log and the output is a set of analysis data with information about the evolution of the artifacts and their dependencies. We report the results in Table 4.2. The results are listed in increasing order of project size. The parameters χ and d are the metrics of *degree of co-evolution* and *distance*, respectively. In this example, $\chi > 0.7$ signifies that the co-evolution is high (χ^H) and $\chi < 0.3$ that the co-evolution is low (χ^L). As previously mentioned, this is a user customizable threshold that can be set by the domain expert. Likewise, the distance is considered low (d^L) when $d <= 2$ and high (d^H) when $d > 2$. The parameter $\overline{|p_f|}$ and $max(|p_f|)$ are respectively the average and the maximum lengths of the path to the root (i.e. average tree depth of the files). The column $|A_{evo}|$ shows the average number of activities in the process representing the artifact evolution. Lastly, the columns $\overline{d}$ and $max(d)$ report the average and maximum file distance, respectively. Next, we use these data for a quantitative evaluation of the projects.

4.4.1 Quantitative Evaluation

Here we address requirements R2 and R3. First, we compute project profiles. These profiles show the distribution of work-related dependencies in a project. Second, we evaluate whether the work on files can be predicted.

Before assessing project profiles, we make the following consideration. Our metrics define four classes: *i)* low distance low co-evolution; *ii)* high distance low co-evolution; *iii)* low distance high co-evolution; *iv)* high distance high co-evolution. Figure 4.4b helps clarifying these four classes. In fact, except for values of distance

[2] The source code is available at https://github.com/s41m1r/MiningVCS



(a) Evaluation on real projects

(b) Distribution of pairs on real projects

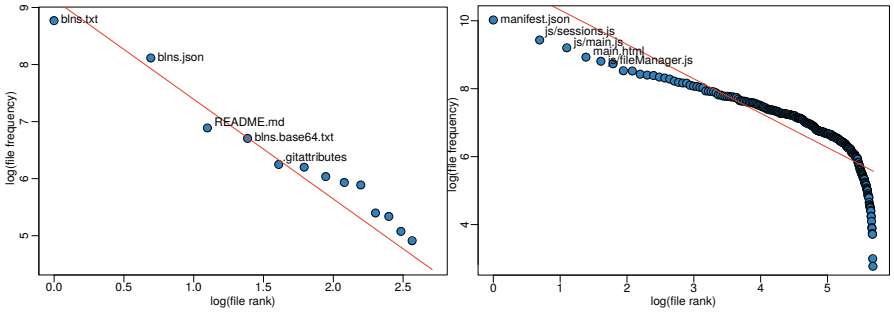
Fig. 4.4: Characterization of the evaluated software projects

equal to 0, it is possible to see how the density of file pairs is higher when the distance is low. This is a normal situation in project where highly related files are stored closely to each other in the file system. Conversely, the dots on the top right of the plot mark files which are very distant to each other but still highly correlated. These can be, for instance, logical dependencies that can happen because of bad modularization of the project.

Hidden work dependencies belong to the last mentioned case, i.e. files are distant in the file tree but they have similar time series. According to this consideration we computed the project profiles in Figure 4.4a. We observe three types of processes. First, several projects have hardly any hidden work dependencies. Second, several have a moderate degree between 10% and 20%. Third, the project *Biglist* has a high share of hidden dependencies. This hints at the possibility for better organizing the project according to good modularization best practices. That means, the project can be restructured in a way to reduce the unwanted side-effect the work on one file produces on other files.

Next, we evaluate whether the work on files can be predicted. Zipf's law is typically used in corpus analysis and states that the *frequency* of usage of any word is inversely proportional to its *rank* in the frequency table. This approach has already been applied to software projects for understanding whether the assignment of developers to tasks in a software project could be predicted Canfora and Cerulo (2006). Here, we focus on understanding whether the Zipf's law holds true also for work dependencies within a project.

To this end, we selected one big and one small project from Table 4.2, namely *Biglist* and *Caret*. Biglist is a small project on a list of strings which are known to cause issues when used as user-input data. Caret is a big project consisting in the development of a sublime text editor for Chrome OS. We collected how frequently were the artifacts worked on to generate a ranking. Figure 4.5 depicts the corresponding charts and the fitted Zipf distribution. We notice that both projects present a similar distribution of values. This holds also for the other projects analyzed. In particular, Zipf's law is valid for the most frequently changed files. Afterwards, the distribution drops because of files not being worked anymore but still being part of the project.



(a) Biglist project (b) Caret project

Fig. 4.5: Zipf distribution of the worked files

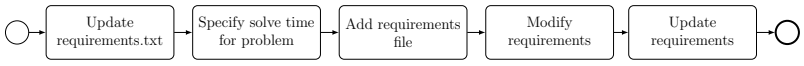
4.5 Discussion

In this section, we address requirement R1 by showing insights on the work history of files that are related. To this end we focused on the project *smsr*, which has 21 commits over a time span of ten days.

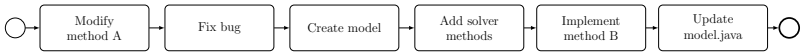
Let us consider an example where our technique proves helpful. Our technique finds 6 highly related pairs, as shown in Table 4.2. We excluded files that have a functional dependencies, e.g. interface-class relations, where a change in the interface trivially brings change in the class. Thus, we were able to select the files `smsr/running example/Requirements/requirements.txt` and `smsr/running example/Software/model.java`, having $\chi = 0.7$ and $d = 4$. Moreover, by observing the content we verified that they do not have functional dependencies. Therefore, these two files are work dependent. Figure 4.6 shows the extracted processes after mining their stories. Interestingly, the two processes do not share any activity because they were never changed together in the same commit.

Our technique can fail under some circumstances. Consider the example above. We know that the files `requirements.txt` and `model.java` are work dependent. Let us now assume that the assumption of *regular commits* in the VCS does not hold. Nevertheless, we know that there is the following work pattern: *at irregular times, one change in the requirements produces 2 changes of work that must be implemented in model in the next day*. In a short time window of 4 days, the time series would be $X_{req} = (1, 0, 1, 0), X_{model} = (0, 2, 0, 2)$ and their correlation is $\sigma(f_{req}, f_{model}) = -1$. Hence, they would score a high degree of co-evolution $\chi = 1$. However, if we double the time window and observe only another pattern the correlation would change. We get $X_{req} = (1, 0, 1, 0, 0, 1, 0, 0)$, $X_{model} = (0, 2, 0, 2, 0, 0, 2, 0)$ which score a $\sigma(f_{req}, f_{model}) = -0.66$, $\chi = 0.66$ and therefore not a high value of correlation.

These results show that our technique helps uncovering work dependencies that are not captured by existing approaches in literature which leverage on social network analysis Zimmermann and Nagappan (2008); Weicheng et al. (2013). On the other



(a) Evolution of file `requirements.txt`



(b) Evolution of file `model.java`

Fig. 4.6: Processes of two work-dependent files

hand, our technique is currently not yet able to retrieve dependencies with delay. We plan to address this challenge by using moving-average time series models.

4.6 Conclusion

In this paper, we addressed the problem of uncovering hidden work dependencies from VCS logs. The main goal was to provide project managers with knowledge about the artifacts co-evolution in the project. Three perspectives of analysis were considered, evolution of the artifacts over time, dependencies among them and structural organization of the project.

Our approach works under the assumptions that repositories reflect the hierarchical structure of the project, project participants commit their work regularly during active working times and they provide informative comments for the changes done. The approach was implemented as a prototype. A scenario of use was provided showing how the approach can be applied and providing some discussions. We also evaluated our approach in real-world data from open source projects showing the potential of the approach.

In future work, we will improve our evaluation varying for instance the time window, the dependency threshold and consider a study case with project managers. We plan to investigate other types of dependencies between artifacts. Specifically, we are interested in a semantic analysis of the work performed in both artifacts, considering for instance some similarity measures. We also aim to improve the visualization to consider other knowledge extracted, for instance the type of change performed in the aggregate events could be shown associated to the activities in the artifact process.

Chapter 5
Article 3: Resource Classification from Version Control System Logs

Authors:

Kushal Agrawal, Michael Aschauer, Thomas Thonhofer, Nico Tomsich, Saimir Bala, and Andreas Rogge-Solti

Published in:

20th IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2016, Vienna, Austria, September 5-9, 2016

Abstract:

Collaboration in business processes and projects requires a division of responsibilities among the participants. Version control systems allow us to collect profiles of the participants that hint at participants' roles in the collaborative work. The goal of this paper is to automatically classify participants into the roles they fulfill in the collaboration. Two approaches are proposed and compared in this paper. The first approach finds classes of users by applying k-means clustering to users based on attributes calculated for them. The classes identified by the clustering are then used to build a decision tree classification model. The second approach classifies individual commits based on commit messages and file types. The distribution of commit types is used for creating a decision tree classification model. The two approaches are implemented and tested against three real datasets, one from academia and two from industry. Our classification covers 86% percent of the total commits. The results are evaluated with actual role information that was manually collected from the teams responsible for the analyzed repositories.

5.1 Introduction

In the field of software engineering, VCS such as Git or Subversion (SVN) have become an indispensable tool and are used for the majority of collaborative development projects. The repositories of these systems store data about projects and their contributors, beyond the raw source code they committed Yu and Ramaswamy (2007). From a management perspective, it is interesting to analyze repository data for compliance purposes. From an academic perspective, repositories hold valuable information about the way people work and collaborate.

In this paper the focus lies on mining and analyzing properties of the users. More specifically, it aims at classifying the users which appear in the logs. This idea is based on the assumption that different types of users utilize the system in different ways and that these differences are reflected in their commits and subsequently in the logs. More precisely, we are interested in the following research questions:

1. Is it possible to classify resources from VCS log data?
2. To what extent can resources be assigned to classes?
3. What are the main differences among these classes?

The main objective is to find a way of classifying users automatically, using an algorithm built on common data mining and machine learning methods. The devised algorithm answers these research questions.

This paper is structured as follows: Section 2 provides the basis for this paper, links it to related work and formulates the research questions. Section 3 presents two approaches on how users can be classified in version control systems. In Section 4 the implementation of two algorithmic solutions is described, followed by an analysis of the results. Section 5 concludes the paper.

5.2 The Problem of Organizational Perspective Discovery

The problem we address in this section is the discovery of the roles that members of a software project may actually play in a collaborative setting. Each member is assigned to at least one role in the project. Roles are decided in the project planning phase. This phase also involves the definition of project tasks and the assignment of suitable tasks to the various roles.

Projects follow clear guidelines. Guidelines may come from internal policies or from rules and regulations of the working domain. For example, software systems in the railway domain must make sure that their development process and resources comply with safety requirements imposed by the European standard EN50128. This is why project managers need to track how they distribute work to different people and whether the project members effectively contribute according to their role and their task.

SCM systems are a valuable source of information to investigate on the behavior of project members. These systems are used for tracking and controlling changes in the

software. If a change produces a wrong or undesired outcome, it is always possible to revert to an older configuration of the system. Artifacts' versions are automatically managed by VCSs. It is always possible to follow the evolution of each artifact, along with information about the resources who changed it and their comments, by looking into the VCS logs.

Figure 5.1 illustrates how people work in a project. People are assigned to defined roles and contribute to shared tasks or perform tasks alone. Their contributions are part of generated artifacts. As the project is enacted, the number and the content of the artifacts in a project grows. Changes in the artifacts themselves and in the structure of the project reflect the development process that project workers follow. The evolution of the repository mirrors the progress of the project.

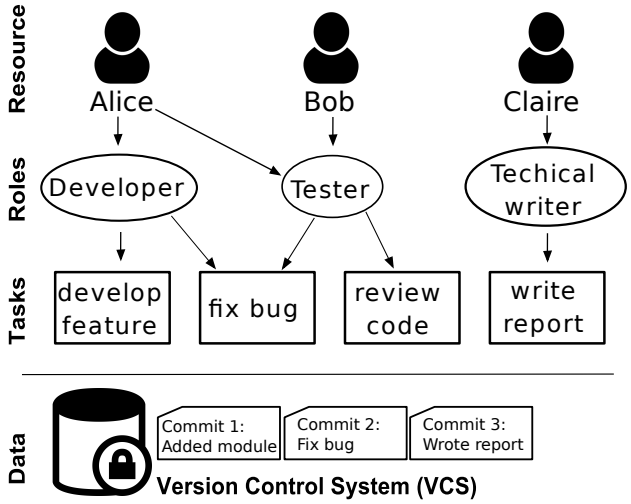


Fig. 5.1: Software project and resources

VCS logs provide rich and fine grained information about the changes in the project. A change may consist of a file being modified or new files being added to or removed from the repository. When changes are complete, it is possible to store them in the repository through a commit. Each commit contains a unique revision number, the identity of the resource who issued the commit, a timestamp, statistical information about the changes for each affected file, and a comment from the person who committed.

Let us consider the following example of how people use a VCS to collaborate in a project. Alice is a software engineer at Abc. ltd, and works on a project with her colleagues Bob and Claire. Alice is mainly assigned to the role *developer*, but she can also be a *tester*. Her tasks include the development of new features and fixing of related bugs. When a feature is ready she performs a commit and adds a new message where she describes her work, e.g. “added new module to demo". At the same time she updates a file named “rule”. This is reflected in the VCS log, like in

the first row of Table 5.1, identified by commit id 1. Bob is a *tester* whose task is to ensure that the code submitted by Alice works properly. He discovers a bug in the "setup" interface file. Bob fixes those minor bugs and informs Alice of further work needed on the feature. Meanwhile, he commits his changes with commit id 2 and comments "Modified the setup interface". Consequently, Alice reworks her code and commits a new version as reported in row 3 of Table 5.1, commenting her work with "Update application interface". As a *technical writer*, Claire takes over and starts to work on the documentation. She commits part of her work as in row 4. As the project continues, the work is accordingly stored in the log as in Table 5.1.

Table 5.1: Example of a VCS log.

Commit ID	User ID	TimeStamp	Updated Files	Comment
1	Alice	2014-10-12 13:29:09	Demo.java rule.txt	Added new module to demo and updated rules
2	Bob	2014-11-01 18:16:52	Setup.exe	Modified the setup interface
3	Alice	2015-06-14 09:13:14	Demo.java	Update the application interface
4	Claire	2015-07-12 15:05:43	graph.svg todo.doc	Define initial process diagram & listed remaining tasks
...	...	...	...	...

As we can see from the example, user work is tracked in a fine granular way by the VCS. The challenge is to use log information for getting insights into the roles of project members. The following section discusses related literature that has addressed role discovery.

5.2.1 Research Questions

As mentioned in the previous section, the focus of this paper lies on mining and analyzing properties of the users of VCS. Users belong to preset classes and they commit message styles. Based on this assumption, the following research questions are defined.

RQ1 *Is it possible to classify resources from VCS log data?* We want to scrutinize whether a classification of the users of a VCS is possible by using the information contained in the log files. Comment messages and other data must be investigated for useful features. The classification must separate the users into clusters, and the most expressive features and their combinations must be identified. These clusters can then be further analyzed by means of the next research question.

RQ2 *To what extent can resources be assigned to classes?* Based on the created clusters, meaningful classes should be derived. The chosen features influence the types of classes that can be created. These classes must present an intelligible

distinction among each other. This research question also inspects how detailed this distinction can become. Moreover, it aims at creating profiles for the class members reflecting behavior, based on the analyzed features.

RQ3 *What are the main differences among these classes?* The last research question examines the differences among the created classes in detail. It compares the results from different log files and identifies the reasons for dissimilarities. Therefore, it evaluates the quality of the classification and its applicability for different version control systems.

5.2.2 Related Work

Role discovery has been addressed by literature in different settings and from several points of view. Here we highlight existing efforts from a data perspective.

5.2.2.1 Structured data approaches

This class of methods includes algorithms that make use of quantifiable data. We divide them into: *a)* Mining Software Repositories (MSR) approaches; and *b)* PM approaches.

Mining software repositories approaches

In the area of MSR, Yu and Ramaswamy (2007) use a hierarchical clustering based on user interactions to identify two categories of users: *core member* and *associate member*. Core members are those users whose interaction frequency is higher than a given threshold. Associate members are instead users whose interaction frequency is below the threshold. Alonso et al. (2008) use a rule-based classifier that maps file types onto categories and hence each author who modified a file is linked to the files' category. Gousios et al. (2008) classify developers contribution based on Lines of Code (LOC) changes and infer activities from them. Begel et al. (2010) developed the Codebook software tool a utility for finding experts. They use a social network approach that combines sources from people, artifacts, and textual references to other people. Ying and Robillard (2014) study developer profiles in terms of their interaction with the software artifacts to understand how they modify files and to further recommend changes based on history from VCS logs. Füller et al. (2014) investigate user roles in innovation-contest communities. They use quantitative methods to analyze user activity logs and interpretative to categorize qualitative comments into classes.

Process mining approaches

Efforts have been done to analyze software repositories with process mining techniques. Rubin et al. (2007) implement a multi-perspective incremental mining that is able to continuously integrate sources of evidence and improve the software engineering process as the user interacts with the documents in the repository. Their approach allows for mining other perspectives, such as roles, by applying social network analysis. However, only statistical methods can be applied to their output, since it lacks the comments that are associated to file changes. In the same setting, Poncin et al. (2011b) developed FRASR, a framework for analyzing software repositories. FRASR can be used in order to transform VCS logs data into the XES (Verbeek et al., 2010) data format that can be further analyzed with process mining tools like ProM[1]. Song and van der Aalst (2008) focus on three types of organizational mining *i)* organizational model mining, *ii)* social network analysis, and *iii)* information flows between organizational entities. Schönig et al. (2015) propose a mining technique to discover resource-aware declarative processes.

5.2.2.2 Unstructured data

Maalej and Happel (2010) use NLP for automating descriptions of work sessions by analyzing developers' informal text notes about their tasks. Developers are then classified into two classes based on their behavior: developers who use problem information to refer to their current activity and developers who refer to task and requirements. Kouters et al. (2012) developed an identity merging algorithm based on Latent Semantic Analysis (LSA) to disambiguate user emails. Licorish and MacDonell (2014) mined developer comments to understand their attitudes.

5.2.2.3 Other related work

The term *role mining* often points to role mining algorithms based on Role-Based Access Control (RBAC) systems. These algorithms take as input predefined roles that are given as a matrix, where each user is assigned to access permissions. A number of algorithms have been developed to mine roles from RBAC systems alone (Lu et al., 2015; Frank et al., 2013) or combining their data with process history logs, as in Baumgrass et al. (2012). A survey of existing techniques and algorithms can be found in Mitra et al. (2016). Our work is disjoint from this class of algorithms as VCS does not contain access control information. Bhattacharya et al. (2014) propose a contributor graph-based model. By constructing a source-based profile, and a bug-based profile, they are able to identify seven roles: *patch tester*, *assist*, *triager*, *bug analyst*, *core developer*, *bug fixer*, and *patch-quality improver*. Hoda et al. (2013)

[1] http://www.promtools.org/doku.php

use a Grounded Theory (GT) approach to study agile teams. Their work distinguishes the roles of *mentor*, *coordinator*, *translator*, *champion*, *promoter*, and *terminator*.

This work builds upon existing literature in that it strives to get insights on organizational level like in Rubin et al. (2007) and Song and van der Aalst (2008), but it takes into account unstructured data. Differently from the literature that works with unstructured data, we explicitly consider the problem of role discovery, i.e. label resources with roles. Lastly, this approach differs from Bhattacharya et al. (2014) and Hoda et al. (2013) since we further adopt NLP techniques.

5.3 Solution Concept

Commits contain fine grained information about many aspects of a work unit. We are interested in three aspects. First we consider distinct identities and timestamps of commits. This makes it possible to compute how frequent and how distant in time commits appear in a project. Second, we obtain information about the authors. Authors are explicitly stated in the commit. Third, we consider their comments. Comments are written in natural language and need to be parsed and preprocessed before we can use them.

Once data is preprocessed, the analysis can start with a direct approach of connecting one commit message to one user and to the corresponding role. In this approach user roles are linked to the authors of commits. However, roles are not always available a priori. In these cases, a commit-based approach can be adopted. This method operates on the commits and assigns types to them first. Further analysis then maps these types to possible user roles. This approach is not a direct mapping. Instead, it allows for more flexible role discovery. We use this approach to also predict roles, based on user profiles that we create from the commit types.

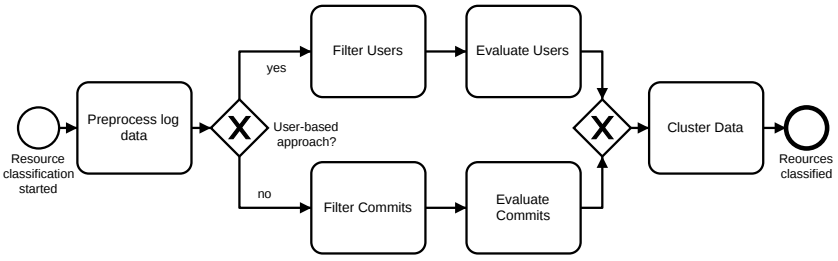


Fig. 5.2: Approach to role classification from VCS logs

Figure 5.2 illustrates the steps of our approach through a BPMN diagram. In the first step we preprocess the data and parse the SVN log file. Then we account for two different types of classification: user-based and commit-based. These approaches require a prior step where we filter the data according to users or to commits. Both

approaches include an evaluation step where we map keywords and information on file types to users or commits, respectively. The last step is the data classification. Section 5.4 describes both approaches in detail.

5.4 Implementation

We implemented the outlined solution concept in a Python script. This script automatically fetches, processes and analyses the log files and creates a classification model that is based on the extracted information. The following tools are used for implementation:

Technology. Python is a general-purpose, high-level programming language. It is open source, easy to use and offers various third party modules[2].

Machine Learning. The Scikit Learn module is used for machine learning[3]. We used Decision Trees (DTs) for classification and regression. DTs are a supervised learning method whose goal is to create a model that predicts a target feature of variable by inferring existing rules from the data.

Natural Language Processing. The Natural Language Toolkit (NTLK) offers methods to extract additional information from everyday communication. Among others, we rely on the *bag of words* technique that analyzes word occurrences[4].

Next, we discuss the data selection criteria. While there are many accessible code repositories, picking a dataset of appropriate size and quality proves to be rather challenging. We need to consider:

Diversity. Classification on a single repository might not produce a generally applicable result, whereas using more repositories increases complexity.

Size. Individual repositories' size is determined by the number of commits.

Roles. The real roles of the contributors of a repository is required for certain classification steps and the verification of results.

Type. Differences in organizational aspects of the projects: the type of the development team (e.g. professional or hobby), type of software (e.g. proprietary or open-source), and type of platform (e.g. private server or public repository hosting service) can influence the way repositories are used.

Backend. There exists a plethora of version control systems, each with their unique characteristics and formats. The approach should be able to handle popular systems including Subversion (SVN), Git, and Mercurial.

Based on these criteria, we selected three repositories for analysis, each with thousands of commits:

---

[2] https://www.python.org/

[3] http://scikit-learn.org/stable/

[4] http://www.nltk.org/book/ch00.html

1. Main code repository of the company Infinica. Proprietary software. VCS: Mercurial
2. ProM Sourceforge project repository. Open source software. VCS: SVN
3. Camunda GitHub project repository. Open source software. VCS: Git

These repositories cover a broad range of the above mentioned differences. The necessary role information was reviewed by interviews with the main contributors of these projects. While two repositories were openly available, the third was granted a direct access from within the company.

An initial screening of commit messages in the repositories revealed that there is useful information within most of the messages, but the variance regarding the style and content of the messages is high between commits and users. From the messages, certain words and phrases can be extracted, which can be linked to a specific class. In a next step, the log files were preprocessed removing commits and/or users with faulty information, merging users with several accounts and anonymizing the data if required.

To unify the different systems, we create a shared object model to store the relevant information. This model consists of a commit object for each commit in the log, consisting of an id, an author, a message, a timestamp and lists of all added, modified and deleted files. From this model it is possible to derive a second model consisting of the users, by aggregating the information for each author name occurring in the commits.

In the following, we present two ways to tackle the problem of classification of users into their respective roles. First, the *user-based* approach focuses on the users at an aggregated level. Second, the *commit-based* approach classifies individual commits to allow for a finer grained classification.

5.4.1 User-based Approach

The main idea of this approach is to use clustering in order to find potential user classes and build a classification model based on those clusters, which represent a comprehensible, existing class of users. As a clustering method we used k-means.

The first step is feature selection. We identified the following features per user.

- Total number of commits.
- Timeframe: the time between the first and last commit of the user (approximates the time a user has been working on a project).
- Commit frequency: total number of commits divided by the time frame. Represents the number of commits a user makes within a certain period of time (e.g. day, month).
- Commit message length: average length of commit messages in number of words.
- Keyword occurrence: how often a certain word (e.g. "test", "fix") is used relative to the total number of words.
- Number of added/modified/deleted files.

- Affected file types: how often a file with a certain format (e.g. .java, .html) are modified by a user, relative to the total number of modified files.

From the clusters generated with this method, classification models are built using the decision tree method. This method uses tree graphs for representing the model. Each branching in the tree represents one decision based on the characteristics of a certain data point. Each leaf of the tree stands for a class. The classification is done by going through the tree for each data point and assigning it the class of the leaf it reaches. The models are trained for the three data sets individually. For verification of their quality, the models are cross validated with the other data sets respectively.

5.4.2 Commit-based Approach

There are multiple reasons for exploring a second approach in addition to the user clustering. As already mentioned above, the research led to the conclusion that in many cases a user can have multiple roles or execute many tasks not belonging to her primary role. This kind of use case is hard to cover using only the simple clustering method. Another reason is that a lot of information is lost when aggregating the commit information for users. This leads to the idea of classifying individual commits.

The algorithm iterates over commits and tries to assign types to them. The types are assigned based on the analysis of the commit message, and on the file extensions. The commit message is searched for certain keywords and phrases which are connected to types. The file extensions are searched for known file types fitting to a commit type. There are certain overlaps between commit types, for example the type addition can be a development or test commit. In those cases where one identified type is a more specific description for another, only the more specific one is included into the further analysis. The identified commits and the related keywords and file types are listed in Table 5.2.

After assigning the commit types, the commits can be aggregated for the individual users resulting in the absolute occurrence numbers of each type for each user. Dividing each of these values by the total number of commits of the user, we get percentages for each type. These percentages tell how the work of a user is distributed among the different kinds of tasks which appear in a VCS. These user profiles are a form of classification, which is not as simple and concise as assigning one definite class for each user, but has the advantage of covering secondary roles and minor tasks. They can be useful for analyzing smaller project teams with multiple roles for one user.

The classification task is done on user profiles. A table of user profiles is created and a role for each user manually inserted, based on the role information of the data sets. For this part the Infinica data set is used, because it is the one with the most extensive information base and it has the most diverse and comprehensive set of roles among the shosen repositories. Four roles are assigned to the Infinica users: Web developers, other developers, testers and support. The last one is an aggregation of users who have different official roles but contribute in the VCS mainly in form of minor, supportive tasks.

Table 5.2: Keyword lemmas used for classification

Class	Keywords
Test	test testcase test case unittest unit test integrationtest fitnesstest fitness test
Development	implement improve update script
Web	web spring http http rest html css servlet
Backend	engine
Maintenance	bugfix fix patch cleanup clean up clean
Refactor	refactor rename move revert
Documentation	document javadoc readme userguide guide tutorial faq translate doc i18n *.txt *.doc *.docx *.text *.tex *.pdf
Design	style icon font layout *.png *.svg *.jpg
Build	build compile release
Data	data database sql postgre postgresql
Tool	tool library framework depend upgrade
Addition	add new create
Removal	delete remove
vcsManagement	svn git mercurial
Automated	automated release, automated nightly
Merge	merge

The classification task was done in two ways: 1. Manual analysis of the table and derivation of rules by looking for similarities between users with the same role. 2. Automated classification in line with our original concept. For the latter the decision tree method is used. In this case the commit type percentages are used as features and the manually assigned roles as classes. The resulting decision tree model was validated against the ProM and Camunda data sets.

The final algorithm including retrieval, processing and analysis of data as well as classification and verification, consists of roughly 700 lines of code. The results for the two approaches are discussed below.

5.4.3 Results

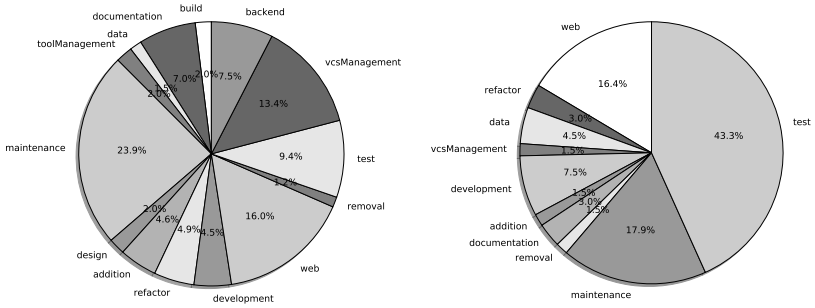
In the first step of the user-based approach, the most expressive features of the Infinica data set were identified. The solution was tested on the Infinica data set due to the immediate availability of detailed information about the log file. The best results were achieved with the combination of the commit frequency and the occurrence numbers of test related keywords.

The commit frequency is a strong indicator of developers. Moreover, it also differentiates between the working preferences of the developers. The group with the lower frequencies tends to work locally on their machines and pushes their work when it is finished. Whereas, the developer group frequently pushes incremental changes to the repository, so that everybody is updated and works with the latest code.

Test-related keywords can not only identify testers, but also differentiate between their expertise. The sum of the words "test", "tested", "testing", "tests" suffices to make a distinction between testers and developers. Additionally, manual testers use these words less often than their technical counterparts, who are more focused on automation.

The first part of the commit-based algorithm implementation, i.e., the commit classification, was successfully tested on all three data sets combined. For all three sets we achieved a similar coverage (percentage of commits which could be assigned some type). For Infinica the coverage was 88,94%, for Camunda it was 90,25% and for ProM 86,65%.

While the user profiles were originally intended as an intermediary step and a means to verify and improve the quality of our approach, they proved to be a useful perspective on user roles, beyond the simple categorisation using a single class. Especially when using a graphical representation such as the pie charts which can be seen in Figure 5.3a and Figure 5.3b, the commit distribution provides an interesting insight in the actual roles and tasks of users.



(a) User profile for a backend developer

(b) User profile for a tester

Fig. 5.3: Commit distributions of the Infinica dataset

The Infinica dataset is divided successfully into expressive classes with k-means clustering with a k=4 shown in Figure 5.4. The developers (light grey circle and white square) are split off based on their higher frequency in the first step of the decision tree and the following classes are derived:

- **Developers - frequent committers:** The square cluster is comprised of developers which push every change to the repository.
- **Developers - heavy commits:** The circle cluster contains developers who work on their local machine and push less frequently. However, they have bigger commits containing more changes.
- **Testers - technical:** Testers focused on automating the testing process are in the diamond cluster. They are solely senior developers but not all of them are situated in this group.

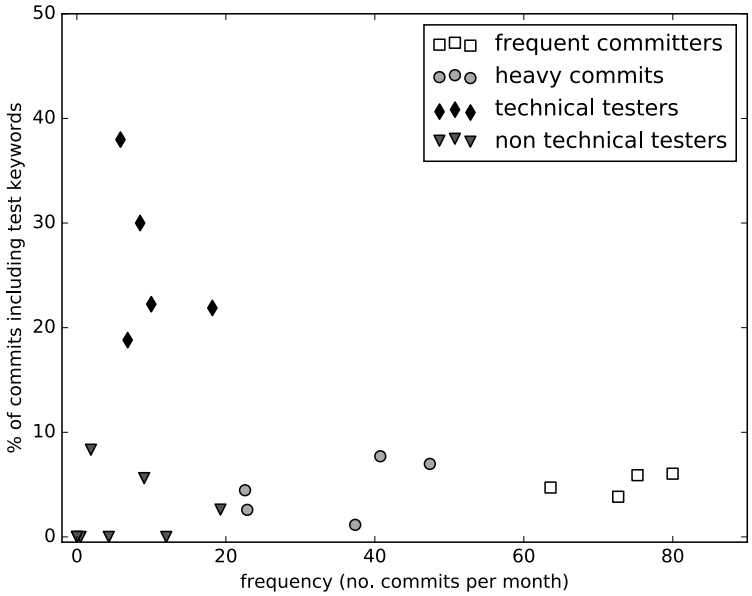


Fig. 5.4: Scatterplot with highlighted clusters (k=4) based on the Infinica data set.

- **Testers - non technical:** The triangle cluster is comprised of less technical testers which tend to do more manual testing. Additionally, it includes the professional services team.

Table 5.3 shows an excerpt of the user profiles and role assignments for the Infinica data set. The commit types development, backend, maintenance and refactor have been merged to a single type development, as the other, more specific types provided no additional value in this case. Also the types "addition", "removal" and "merge" have been omitted due to a lack of semantic meaning. The data visible in the table was used for the manual classification as well as the creation of the decision tree model.

We identified 4 role-based classes which were visible from the Infinica data. Those were testers, with more than 20% of their commits being of type test and between 30% and 50% of type development, developers with above 50% development commits, web developers with more than 40% of type web, and 20% of other development and non-technical users with less than 20% development. In addition to those, we found some less obvious hints for additional classes, represented by minor, secondary roles of users. These were not represented as actual existing roles in our data, so we could not verify our assumptions. The corresponding classes we suggest for those are technical writer with more than 40% documentation commits, designers with above 30% design commits, users with various administration tasks, represented by

Table 5.3: Excerpt from user profiles with roles for Infinica data set

test %	development %	web %	documentation %	vcsManagement %	build %	toolManagement %	data %	design %	class name
6.07	39.49	26.40	7.94	0.23	3.27	0.70	0.23	11.21	dev
9.41	40.90	15.99	7.00	13.37	1.96	2.04	1.46	2.00	dev
3.83	26.78	44.70	2.90	4.70	3.93	0.60	2.95	6.01	webdev
0.00	28.07	52.28	2.46	0.00	5.26	1.40	0.35	8.42	webdev
43.28	28.36	16.42	2.99	1.49	0.00	0.00	4.48	0.00	tester
23.99	27.73	15.26	7.79	0.00	2.49	1.56	1.25	4.67	tester
0.00	7.94	38.10	0.00	38.10	1.59	4.76	0.00	9.52	support
1.19	3.39	46.15	1.61	46.15	0.08	0.17	1.19	0.08	support
...	...	...	...	...	...	...	...	...	...

high numbers of the types build, vcsManagment and toolManagement and database experts with a large percentage of data commits.

When applying the optimal features and clusters deduced from the Infinica data set on the ProM data set, different but still meaningful user classes are derived. This is due to the fact that the Infinica data set represents the development process within a company, whereas ProM is an academic project where researchers continuously join and leave. However, it still required to get invited for working on the project which creates an entry barrier. All members are developers without any designated testers. The ProM data set can be divided into the following four classes as shown in Figure 5.5.

- **Core developers:** The employed users of the ProM project are situated in the square cluster. Their commit frequency and absolute number of commits is far above the others. There is also a system user in this class. Its main task is building the project.
- **Engaged developers:** The circle cluster contains developers which also write tests. They are more committed and they put more effort into the development.
- **One-time developers:** The enagement of this group ends with the addition of their required functionality. The majority of the users falls into this class and it is represented by the diamond cluster.
- **Testers:** Despite the lack of testers in the ProM data set two have been identified as such in the triangle cluster.

The Camunda data set was analyzed based on the optimal features and classes derived of the Infinica data set. The majority of the users are developers like in the ProM data set. However, it is an open source project and users can commit anything at any time without restrictions. There is a permanently appointed core team which evaluates, selects and integrates commits for the product. The clustering creates

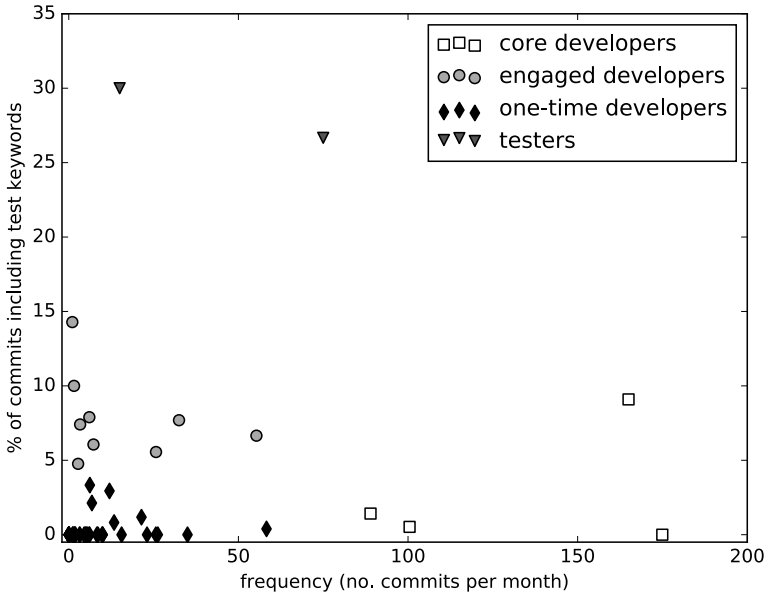


Fig. 5.5: Scatterplot with color-coded clusters, ProM data set

similar classes like in the ProM data set and it can be divided into the following four classes as shown in Figure 5.6.

- **Core developers:** The square cluster contains the employed users of the Camunda project. Their commit frequency and absolute number commits is far above the others.
- **Engaged developers:** Developers which stay longer with the project are situated in the circle cluster. They are refining their own committed code or they are extending the product in different areas.
- **One-time developers:** Similar to the ProM data set the majority of the users belongs to this group and it is represented by the diamond cluster. Code usefull or not is committed typically once and there is no lasting commitment.
- **Testers:** Also testers have been identified in the triangle cluster. They already have a lower frequency than developers and therefore it is difficult to make an estimation about their commitment.

For the user-based approach no further distinction can be made based on expertise, time in the company, teamwork, development area, project membership, room allocation, etc. In the case of a development from junior to a senior role in the course of the coverage of the log file the respective persons stay within their previous clusters. The associated increase or decrease in commit frequency can not be linked to the development.

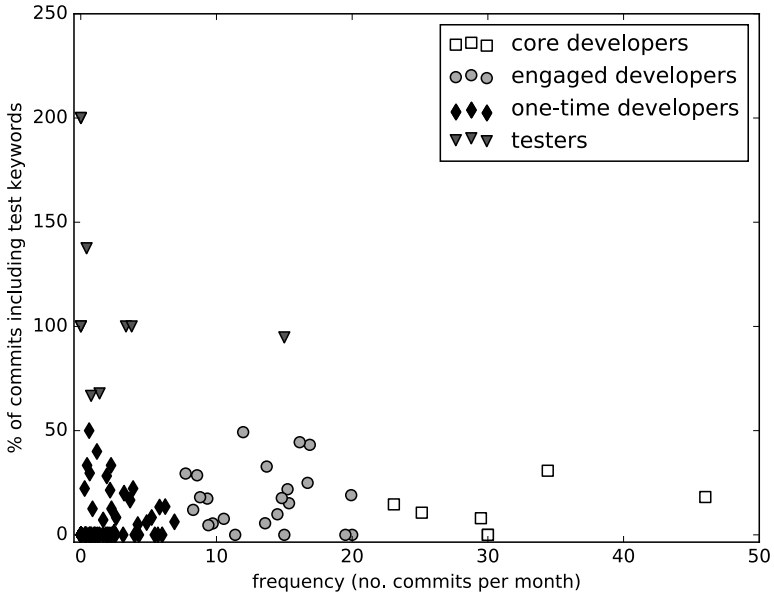


Fig. 5.6: Scatterplot with colorcoded clusters, Camunda data set

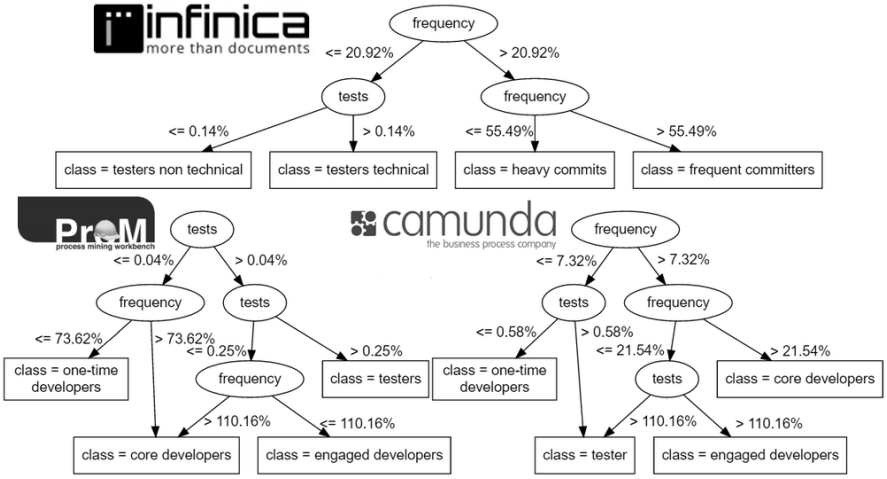


Fig. 5.7: Decision trees, all three data sets

Decision trees are an effective tool to display the formation and the partitioning of clusters in a tree-like model. The corresponding decision trees for the previous presented k-means clustering shown in Figure 5.7 display the boundaries of the clusters. Due to the differences in the structure of the projects, business vs open source, the data sets share only little similarities.

The Infinca decision tree initially splits the developers and the testers apart based on the commit frequency with a border value of 20.92%. The developers are split again into sublcasses based on the commit frequency, which now seperates them at 55.49% into frequent committers and heavy commits. The testers are split based on test related word occurances at 0.14%, where the non technical testers use these words less often than their technical counterparts. On the contrary, the other two trees start separating the subsets right away in different orders without displaying a clean seperation between developers and testers from the beginning.

The decision tree model for the commit-based approach is depicted in Figure 5.8. When comparing its rules with those of our manual classification there are some clear parallels. In the tree the differentiation between web developers and other developers is done with a border value of 36% for web commits, close to the 40% threshold of the manual table. Testers are identified having more than 17% test commits, similar to the 20% boundary. The decision tree separates the support users from the rest using the vcsManagement type. Looking at the data, this rule is definitely valid for the Infinica data, however as we were not able to provide a definite explanation for this class having higher percentages for that particular type, we can not say if this rule would also apply to other data sets. One possible explanation is that all users have a similar number of vcsManagement commits within a given timeframe, but because the support role has on average a significantly lower number of commits, they account for a higher percentage in the distribution, however this is only an assumption.

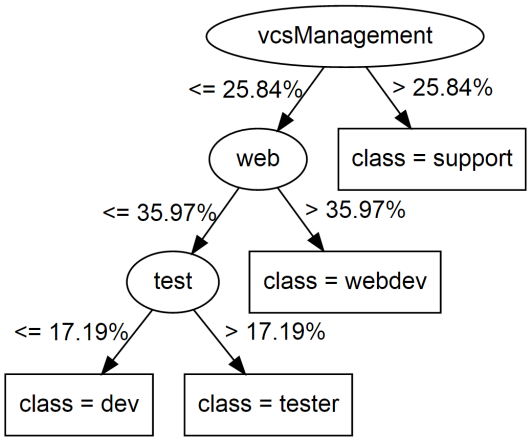


Fig. 5.8: Decision tree, commit based model

In general the decision tree is more precise, but our manually created rules capture more factors of differentiation and we were able to identify some potential classes which could not be included in the programmatic classification with reasonable effort. A clear advantage of the automated model is that it can be efficiently applied to other data sets. Applying it to the Camunda and ProM data sets provided some verification for our decision tree model. We only included users with five or more commits into the cross validation. For the Prom data set which consisted of 42 users, 35 (83%) were classified as developers. This is not surprising as we knew before that the contributors of open source projects are mostly developers, which was also confirmed by our contact persons for ProM and Camunda. Six (14%) were regarded as testers, which fits to our results in the user-based approach. One remaining user (2%) was classified as a web developer. As the ProM project has no web component, it makes sense that there are no web developers found by our algorithm. The one we found has only five commits, two of them web commits so this potential misclassification provides no evidence against the quality of the overall approach.

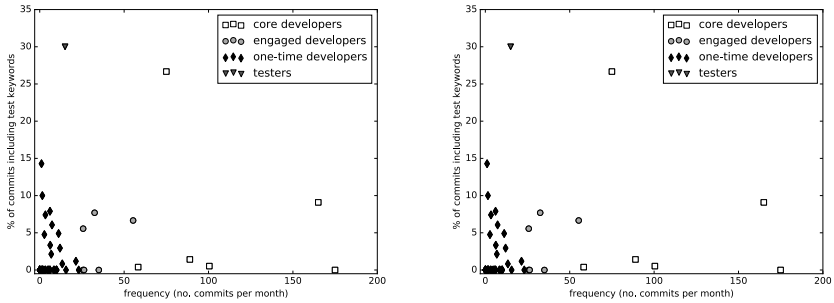


Fig. 5.9: Classication based on Infinica training set: Prom, Camunda

For the Camunda data the case is similar. Out of 66 total users, 49 (74%) percent were regarded as developers, the high number is again explained by the type of project and was confirmed by the contact person we interviewed. 10 (15%) testers were found, also mostly fitting to our knowledge of the data. Contrary to ProM, there is a web part in the analyzed Camunda project reflected by the fact that our algorithm found 7 (11%) web developers. In neither of the two validation data sets a support user was found. This might be due to the differences in the organisational structure between Infinica and the other two or it could stem from the classification rule based on the vcsManagement commit type, which we already discussed as potential source of error. All users in the Camunda and ProM data have below 15 percent vcsManagement commits, too small to fit the support class generated by the decision tree model. However there are other commit types with high percentages, such as build or toolManagement, in Camunda and Prom, which might fit the support class or a similar one.

When validating the model with the Infinica data set and then predicting the ProM and Campunda data set, only moderate results are achieved with the user-based approach as shown in Figure 5.9. For the Infinica data set meaningful classes can not be conveyed to the other data sets due to the structural differences of the projects.

For ProM (Figure 5.9 left) the trained classes are less representative than the ones created when clustering on its own, especially since the ProM data set does not include any designated testers. The prediction of Camunda (Figure 5.9 right) on the other hand performs better. However, the core developer class now also includes very committed contributors.

Because of the missing designated testers in both test data sets, the initial differentiation between technical and non-technical testers is not possible. Due to the moderate results of user based approach the commit based approach was initiated.

5.4.4 Discussion

Throughout this project we discovered a number of research directions to follow, methods to use and features to analyze, which made our research much more exploratory than originally planned. For our research we picked from a vast selection of potential methods and tools, a small set which seemed promising to us. The user-based approach, which was our initial plan, provided a useful insight into the data but fell short of providing generally applicable results in terms of a classification algorithm. The commit-based approach seems more valuable to us, due to its results and the much larger spectrum of information it delivers for analyzing users. The commit classification method seems quite robust and could with some extensions and refinements become a useful tool for analyzing arbitrary VCS repositories. The user profiles created based on this classification are definitely an interesting source of information and building a classification model on top of them has proven to be a legitimate approach.

Our first research question (RQ1) can be answered with: Yes, it is possible to classify users based on the information usually found in VCS logs. Our results show that at least for some typical roles in software development there can be enough information in commit messages and file types, to make certain statements about the users who created them. However this is not true for all users, especially when their commit styles and frequencies are very different.

The other two questions are more difficult to answer. The classes to be found can differ between repositories. From our experience, developers can be distinguished from other roles quite easily due to high numbers of commits and/or modified files as well as the usage of certain words and phrases. Testers can also be set apart in most cases, mainly through key words. Looking at the file types of added, modified and deleted files also reveals users in the field of web development. Beyond that there are certainly more differences and classes to be found but they are less obvious and require more research.

What do these results mean for the practical use of resource classification using VCS logs as discussed in Section 5.2? One clear finding of our work is that any project team trying to use this kind of classification, should specify and enforce some guidelines for how VCS should be used. They should at least make the usage of commit messages mandatory, ideally also define a structure and vocabulary for them. While the models we created are probably not applicable for an arbitrary team or project, a team could follow our approach to create their own individual model using specific knowledge. For a more generally applicable classification model, additional research and improvements to our solution would be necessary.

5.4.4.1 Limitations

While our algorithms and models work well for describing our three data sets, it still needs to be tested for other repositories, especially from other domains. This would require testing with more data sets, more verification information and more manual analysis. While fetching more VCS repositories and running them through our algorithms could be done in a short amount of time, obtaining the necessary role information is challenging and the manual analysis time consuming.

A current limitation of analyzing VCS logs is that commit messages are used differently between users, repositories and teams. We have to assume that it is challenging to find one classification model that produces valid results for any VCS repository, because there are ambiguities in what certain statements mean. In the worst case, users omit commit messages, which renders classification based on text impossible.

5.5 Discussion

Throughout this project we discovered a number of research directions to follow, methods to use and features to analyze, which made our research much more exploratory than originally planned. For our research we picked from a vast selection of potential methods and tools, a small set which seemed promising to us. The user-based approach, which was our initial plan, provided a useful insight into the data but fell short of providing generally applicable results in terms of a classification algorithm. The commit-based approach seems more valuable to us, due to its results and the much larger spectrum of information it delivers for analyzing users. The commit classification method seems quite robust and could with some extensions and refinements become a useful tool for analyzing arbitrary VCS repositories. The user profiles created based on this classification are definitely an interesting source of information and building a classification model on top of them has proven to be a legitimate approach.

Our first research question (R1) can be answered with: Yes, it is possible to classify users based on the information usually found in VCS logs. Our results show that at

least for some typical roles in software development there can be enough information in commit messages and file types, to make certain statements about the users who created them. However this is not true for all users, especially when their commit styles and frequencies are very different.

The other two questions (R2 and R3) are more difficult to answer. The classes to be found can differ between repositories. From our experience, developers can be distinguished from other roles quite easily due to high numbers of commits and/or modified files as well as the usage of certain words and phrases. Testers can also be set apart in most cases, mainly through key words. Looking at the file types of added, modified and deleted files also reveals users in the field of web development. Beyond that there are certainly more differences and classes to be found but they are less obvious and require more research.

What do these results mean for the practical use of resource classification using VCS logs as discussed in Section 5.2? One clear finding of our work is that any project team trying to use this kind of classification, should specify and enforce some guidelines for how VCS should be used. They should at least make the usage of commit messages mandatory, ideally also define a structure and vocabulary for them. While the models we created are probably not applicable for an arbitrary team or project, a team could follow our approach to create their own individual model using specific knowledge. For a more generally applicable classification model, additional research and improvements to our solution would be necessary.

While our algorithms and models work well for describing our three data sets, it still needs to be tested for other repositories, especially from other domains. This would require testing with more data sets, more verification information and more manual analysis. While fetching more VCS repositories and running them through our algorithms could be done in a short amount of time, obtaining the necessary role information is challenging and the manual analysis time consuming.

A current limitation of analyzing VCS logs is that commit messages are used differently between users, repositories and teams. We have to assume that it is challenging to find one classification model that produces valid results for any VCS repository, because there are ambiguities in what certain statements mean. In the worst case, users omit commit messages, which renders classification based on text impossible.

5.6 Conclusion and Future Work

We presented a first approach to resource classification from VCS and demonstrated feasibility of this non-trivial task. The data saved in log files contains much useful information. But in particular the structure of the underlying team and project has a big influence on finding classes and their identified attributes. Further, the many differences between individual repositories make it difficult to generalize findings.

Automating the classification with machine learning is a viable method. Even though not all required steps of creating the classification models can be done

automatically and further information about the users and structure is essential. Despite current challenges, the users can be classified to a certain extent. A more precise classification, however, requires future research. One direction is to further explore clustering and classification techniques to find better suited techniques. Finally, an extension of the approach with state-of-the-art semantic reasoning seems a promising vein of research to the authors.

Chapter 6
Article 4: Discovering the Control-Flow Perspective

Authors:

Saimir Bala, Paul Kneringer, and Jan Mendling

Published in:

Proceedings of the Forum at Practice of Enterprise Modeling 2020 co-located with the 13th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM 2020), Riga, Latvia, November 25-27, 2020

Abstract:

Software development processes are complex to monitor as they involve the coordination of many resources working with different tools. This makes it hard to apply mining techniques for monitoring the process. A key challenge for using traces of tools such as version control systems (VCS) is to find meaningful abstractions in order to identify the work that was actually done. In this paper, we use data from VCS to analyze the actual progress of software-development processes. We develop a technique that is able to mine the activity types of which the development processes consists. We implement our technique as a prototype in Java and evaluate its outputs in terms of effectiveness. In this way, we are able to graphically uncover new behavioural patterns in real-world data from existing open-source GitHub repositories.

6.1 Introduction

Software development processes involve the coordination of multiple resources working on different parts of the software at the same time. As these resources focus on specific parts of the overall development process, it is difficult to obtain transparency of the current status of the project. At the same time, monitoring such process is required in order to avoid risks of running out of time, budget or not meeting established quality objectives.

These type of processes have been referred to as *project-oriented* business processes or simply as *projects*. Monitoring a project is complex because there is hardly any central control of the work progress. One type of system that is extensively used in software development are VCSs. They are used to keep track of the changes of the different files that constitute a project. Trace data generated by VCS is a starting point for mining these type of processes. However, only a few approaches Bala et al. (2015); Jooken et al. (2019) focus on analyzing the status of software development from these fine-grained VCS traces. Thus, there is a need to fill in this gap with further techniques that allow to identify additional aspects of the development process, such as the actual activities done by developers.

In this paper, we provide a technique for capturing the progress of a project in such a way that it becomes clear what work activity is being done over time. We define fundamental concepts for representing these processes, upon which we develop a novel analysis technique. Our prototypical implementation of this technique is able to represent the status of the development process as well as the activity that is being done.

The rest of the paper is organized as follows. Section 6.2 details the problem, positions our contribution against existing literature, and defines the requirements for the design of the artifact that solves the stated problem. Section 6.3 defines preliminary concepts and presents the approach to mine the activities. Section 6.4 describes the implementation of the artifact and shows its application to real-world projects. Section 5.6 concludes the paper.

6.2 Background

6.2.1 Problem description

The problem discussed in this paper is the monitoring of software engineering projects. These projects present the following characteristics. First, they are hardly repetitive. That is, while best practices learned from one project endeavour can be reused, it is never the case that the project is rerun exactly in the same way. Second, they are worked on collaboratively by many participants who regularly document their progress in a semi-structured way. Third, the work is performed under clear constraints in terms of time, budget and quality. Fourth, the workflow does not follow

an imperative process model and is not managed by a process engine. Fifth, despite the aforementioned limitations, project managers require transparency of the process in terms of being able to distinguish what kind of work activities where done when and by which resource. In this paper, we consider the terms “type-of-work” and “activity” as synonyms.

Project managers need tools that help them understand where the project is currently standing and how they have performed in retrospect. Being able to see what work was done and when opens up possibilities to understand inefficiencies about how the process was conducted. Moreover, managers need to access information in a timely manner. Therefore, a representation of progress over time is important. Theoretical models of the development process, such as the RUP, are useful for planning software projects but they fall short when it comes to monitoring.

Fortunately, software development projects store rich trace data. Participants (e.g., resources, users) use many tools for different purposes. A common tools is used in any professional software engineering project are VCSs. These systems allow users to collaboratively work on the same project. They manage the different versions of files created by users at any point in time. As well, they keep track of all the changes done by resources at all times. These kind of systems represent a starting point for analyzing projects. As software projects may contain hundreds of thousands files, it becomes prohibitive to manually check what work was done in the project. This calls for automatic analyses and reporting.

Typical *activities* (i.e., type of work) traced by VCSs fall under the categories *Code*, *Documentation*, *Test*, etc. Major VCSs like Git and Subversion, do not provide direct support for understanding the activities executed by developers. However, these tools provide rich event logs which record all the changes made to any file. Listing 1 presents an excerpt of trace data from a publicly available GitHub repository. Specifically, the trace data shows information about two commits, which are activities performed by developers to save their work progress. These commits have a unique identifier and provide information about *i)* the author (i.e., the human resource who issued the commit); *ii)* the date (i.e., a timestamp recording the instant when the commit was made); *iii)* a natural-language textual description filled in by the author; and *iv)* a list of files that were either modified (M), added (A) or deleted (D).

Listing 1: Excerpt from VCS log data from git

```
commit b0346a47df142394da820e1e5d0f7e31b41a70d3
Author: s41m1r
Date:  Wed Feb 4 13:05:14 2015 +0100

    Deleted TODOs

D  MiningSVN/TODOs
M  MiningSVN/src/reader/GITLogReader.java

commit 30c5e536e88501295aa3f226645953c69e8f3947
Author: s41m1r
Date:  Wed Feb 4 15:31:05 2015 +0100

    Works with GIT (hopefully)

M  MiningSVN/src/model/git/GITLogEntry.java
```

```
M  MiningSVN/src/reader/GITLogReader.java
M  MiningSVN/src/reader/LogReader.java
M  MiningSVN/src/test/TestReadDate.java
A  MiningSVN/src/test/TestReadGIT.java
M  MiningSVN/src/test/TestReadSVNLog.java
```

As real-life event logs may contain a large amount of commits, it is imperative to use automatic tools to discover the activities. While it is possible to take into account the commit messages and apply NLP techniques to classify the various changes Agrawal et al. (2016), Listing 1 suggests that there are no guarantees that the textual descriptions are informative about the activity. For this reason, this work focuses on the type of file that was modified rather than relying of the commit comments.

Therefore, the problem is how to exploit low-level trace data for extracting project knowledge that is informative to the manager. We translate this problem into the following requirements.

RQ1. (Processing of VCS event logs). The prototype must extract valuable information fromVCS data.

RQ2. (Identification of the activities). The prototype shall classify what activity is done and when.

RQ3. (Computation of KPIs). The prototype shall provide KPIs that are understandable by project managers.

RQ4. (Visualization of project status). The prototype shall provide a high level overview of the project.

6.2.2 Related work

Literature related to the aforementioned requirements can be classified into two main groups: *(i)* software engineering; and *(ii)* business process management.

Contributions in *(i)* focus on event data generated by systems like VCS, issue tracking, bug tracking, mail archives, etc. They mainly aim at either finding correlations between activities performed by resources and the artifacts in the repositories Oliva et al. (2011) or at analyzing the evolution of changes over time Zimmermann et al. (2005). These works typically provide powerful techniques that help with processing events Zimmermann and Weißgerber (2004) from software data and further abstracting them into coarse-grained activities Oliva et al. (2011); Zaidman et al. (2008); Rodríguez et al. (2018) and understanding type of work (i.e., activites) and KPIs Vasilescu et al. (2014); Joonbakhsh and Sami (2018). While these works are fundamental in dealing with software repositories, they are typically process unaware. Therefore, do not provide a process representation.

Contributions in *(ii)* fall under the *process mining* umbrella. Typical approaches focus on transforming software development data in process-mining compatible event-logs Kindler et al. (2006b); Poncin et al. (2011c), by making assumptions on what to consider as a case identifier. Other works focus on enabling process analytics

on top of fine-grained events from evolving artifacts Beheshti et al. (2013). Finally, there are process-aware works that deal with software repositories. In particular, the work from Marques et al. (2018) analyses bug resolution processes and the work from Jooken et al. (2019) uses VCS data to analyse teams. Most of the techniques in the process mining area have specific requirements about their input (i.e., an event log with defined case, activity, and timestamp attributes). These works cannot be readily applied to data from software development Tsoury et al. (2018). In this paper, we focus on automatic identification of the activities based on file types as described in Vasilescu et al. (2014). Moreover, this work is process aware and presents the data from a perspective which is more targetted towards project managers.

6.3 Technique for discovering activities

We developed a technique that addresses the identified requirements according to the process model in Figure 6.1. This process starts when an event log from VCS needs to be analyzed, ends with a final visualization of project insights, and consists of four steps. First, an event log from VCS is taken as an input. In this step, an Extract, Transform and Load (ETL) procedure is followed to obtain a structured event log which is easily processed in the next steps. Second, the activities are discovered based on file changes. Third, project related KPIs are computed. Fourth, a visualization that summarizes the results of the previous two steps is provided to the project manager. In the following, we provide the implementation details of this process.

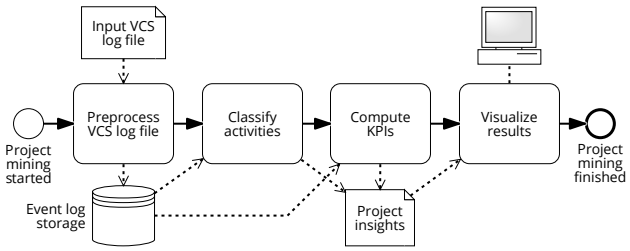


Fig. 6.1: Overview of the approach

6.3.1 Preprocess VCS log file.

The input of this phase is a log in the unified diff format, which is supported by the major VCSs, such as Git and Subversion. The information retrieved by the VCS is configurable by the user. More specifically, it is possible to obtain the basic information shown in Listing 1 along with details on the differences among versions

of the same file (i.e., which and how many lines changed from version 1 to version 2 of file X). In order to extract such information, the raw event log is parsed. We used the parser from Bala et al. (2017b). This parser generates events which are then stored into a Database Management System (DBMS) for further processing.

Figure 6.2 provides the Entity-Relationship (ER) model to represent the entities and relationships that are needed to capture an event. The entity *Project* models the software project at hand. By including this entity in the data model, we can gather data over multiple projects and store them in the same data store. *User* is the user who performs change operations on the files. *Commit* represents the status of the repository at a given point in time. Commits must contain a revision number, a timestamp and the user who issued them. *File* is a single file of the repository identified by its full path. *Edit* captures the change as numbers of lines added to or removed from a file. This step fulfills **RQ1**.

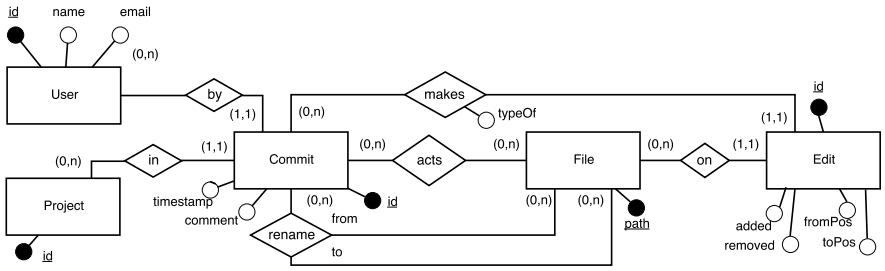


Fig. 6.2: ER diagram capturing the entities and relationships of a software repository

6.3.2 Classify activities.

Having the data stored in a DBMS, enables us to run several analyzes already at this level by simply issuing SQL queries. For example, we can obtain all changes that happened to single files during their lifetime. For the scope of this work, we collect all the file paths, all the changes that happened to files, the amount of change in terms of LOC, the type of change (e.g., addition, modification, deletion), the commit identifier, and the user who did the change. Next, we automatically categorize the type of change. For this, we apply regular expression on path attribute using the classes provided by literature Vasilescu et al. (2014). Examples of classes are *Documentation*, *Image*, *Testing*, *Coding*, *User interface*, etc. There are in total fourteen classes. When a type of change does not belong to any class, it is put under the category *Unknown*.

We have adapted the regular expressions to our case and enriched the list of rules from literature. Table 6.1 shows the activity types and the main regular expressions we use to classify files onto specific types of work. For the sake of space, the majority

Table 6.1: Set of activities that are discovered and main regular expressions

Activity	Abbreviation	Regular Expression
Unknown	unknown	.*
Documentation	doc	.*\/doc(-?)book(s?)\/.* .*\/info .*\.txt((\.bak)?) .*\.man .*\.tex
Image	img	.*\.jpeg .*\.bmp .*\.chm .*\.vdx .*\.gif
Localization	loc	.*\/locale(s?)\/.* .*\.po(~?) .*\.charset(~?)
User interface	ui	.*\.ui .*\.gladep(\\d?)((\.bak)?)(~?) .*\.theme
Multimedia	media	.*\.mp3 .*\.mp4 .*\/media(s?)\/.* .*\.ogg
Code	code	.*\.jar(~?) .*\/src\/.* .*\.r((\.swp)?)(~?)) .*\.py((\.swp)?)(~?) .*\.php((\.swp)?)(\\d?)(~?)
Meta	meta	.*\.svn(.*) .*\.git(.*) .*\.cvs(.*)
Configuration	config	.*\.conf .*\.cfg .*\.project .*\.ini .*\.prefs
Build	build	.*\.cmake .*\/install-sh .*\/build\/.* .*makefile.*
Development documentation	devdoc	.*readme.* .*\/changelog.* .*\/devel(-?)doc(s?)\/.*
Database	db	.*\.sql .*\.sqlite .*\.mdb .*\.db
Test	test	.*\.test(s?)\/.* .*\/.*test\..* .*/test.*\..*
Library	lib	.*\/library\/.* .*\/libraries\/.*

of the regular expressions is left out. The reader can access the full list of regular expressions on our GitHub repository whose link is provided in the following section.

For the categorization we consider both the extension of the file and its path. For example, a file with the path `"/test/file.java"` is labelled as *Testing* rather than *Coding*. To achieve this, we sort the matching rules in order of specificity. At a higher level, a commit involves multiple files. In order to fit the commit into a specific class, we rely on majority voting as follows. We iterate over the list of changes affected by the commit and sort the number of changes by their activity and the amount of change. We select the activity that is associated to the highest number of changes. With this step we fulfill **RQ2**.

6.3.3 Compute KPIs.

According to the process already presented in Figure 6.1, we compute KPIs with the help of the DBMS. This allows for a customized set of KPIs to be implemented. In the scope of this paper, we reproduced some of the main KPIs form literature (Vasilescu et al., 2014). We divide them into basic (absolute and relative) and specialization metrics. Basic metrics focus on descriptive statistics such as frequency counts of how many times each user works on a file. Specialization metrics focus on the measuring imbalance of work towards a specific file or author. Imbalance is captured by the Gini inequality index (Gini, 1921). We implemented the following basic project metrics: Project Workload (PW), Type of change Workload (TW), Number of Authors in Project (NAP), Number of Types of work in Project (NTP). Furthermore, we also

implemented the following specialization metrics: Specialization of Author in each Activity Type (PIS), Specialization of Relative Author in each Activity Type (RPIS), and Specialization of Relative Project Workload (RPWS).

Let U be the set of all unique users in the project, T the set of all activities in the project. Files that were edited in the context of a commit can be associated to a user $u \in U$ and an activity $t \in T$ computed as described above. We adapted the definition of KPIs from literature to support the extraction of information about the activity types as follows. As a first step, we redefine the two basic KPIs that are involved in the calculation of all other KPIs. User Activity Workload (UTW) is the number of files relative to activity t, a user u edits over the entire history of the activity. User Activity Involvement (UTI) is 1 if a user u has been involved in (i.e., has edited at least once) a file with activity w. It is 0 otherwise. Finally, using these definitions, we can present in Table 6.2 how the rest of the KPIs is computed. With this step we fulfill **RQ3**.

Table 6.2: KPIs computation details

KPI	Description	Calculation
PW	(absolute) project workload	$\sum_{t \in T, u \in U} UTW(t,u)$
TW (t)	workload of a specific activity	$\sum_{u \in U} UTW(t,u)$
NAP	number of authors in the project	$\sum_{u_j \in U} j$
NTP	number of activities in the project	$\sum_{t_j \in T} j$
PIS	specialization of user involvement the activities of the project	$Gini_{t_k \in T}(\sum_{u \in U} UTI(u,t_k))$
RPIS	specialization of relative user involvement over the activities of the project	$Gini_{t_k \in T}(\frac{\sum_{u \in U} UTI(u,t_k)}{NAP})$
RPWS	specialization of relative workload across all activities in the project	$Gini_{t_k \in T}(\frac{\sum_{u \in U} UTW(u,t_k)}{TW})$

6.3.4 Visualize results.

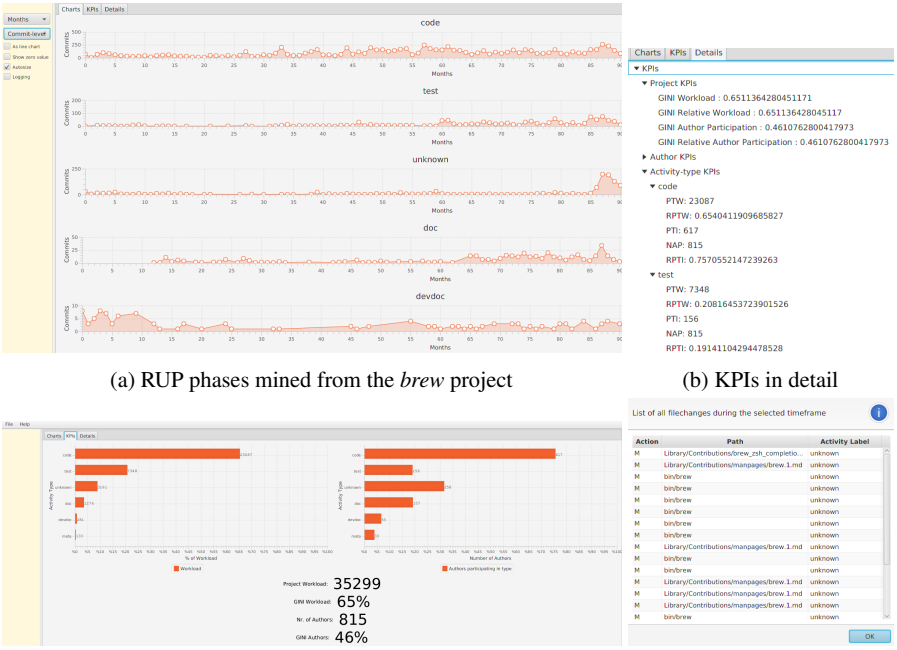
The last step of our technique deals with the presentation of the results. As our prototype needs to be informative to project managers, we chose to graphically display the results of the previous two steps on a friendly user interface. The user interface takes as input the results of the classification of the activities as well as the set of the computed KPIs. The main goal of the user interface is to show two fundamental aspects of the project at hand. First, it visualizes the evolution of each activity aggregated by customized periods of time (e.g., weeks, months). By doing so, our prototype helps at vizualing the general behaviour as RUP phases. This enables the project manager to compare the ideal project evolution to the actual one. Second, we display the various KPIs in a dashboard (e.g., barchart). Further information, such as the commit identifiers, file names, amount of changes, and users who worked on the files are also made available. This allows the project manager to zoom into specific parts of the project for more detailed analyses. With this step we fulfill **RQ4**.

6.4 Results and discussion

Our prototype is called ActiVCS and it is built in Java, using Java FX for the user interface. We tested its performance on a laptop with Intel®Core™ i5-6200U CPU @ 2.30GHz x 4 processor with 8 GB of DDR4 RAM and Linux kernel 4.15.0-88-generic 64-bit version. ActiVCS takes as input a Git event log generated via the command `git log --reverse --name-status > filename`. The output is presented to the user through an interactive interface. ActiVCS is publicly available and open source on GitHub (https://github.com/PaulKner/ActiVCS).

6.4.1 Visualization of projects

In the following, we present the graphical interface of our prototype.



(a) RUP phases mined from the *brew* project

(b) KPIs in detail

(c) Effort distribution

(d) Details on file changes

Fig. 6.3: ActiVCS applied on the real-life event log of the *brew* project

Figure 6.3 shows the visualization tool in action. We extracted the VCS log of the *brew* project from GitHub. ActiVCS has a Graphical User Interface (GUI) with four main frames. The main frame is depicted in Figure 6.3a. It allows the domain expert

to execute a number of actions. These actions are the following: *i)* load an event log; *ii)* save an event log for reuse and avoid parsing it anew; *iii)* get help information; *iv)* change granularity of the timeline by choosing to display data daily, weekly or monthly; *v)* change data series on X-axis between commit-level and file-level; and *vi)* four buttons to change size and type of the plots. The plots shown in the frame illustrate the evolution of the identified types of work over time. Through the drop-down menu on the left pane, the user can choose between showing the classification of the activity on a commit-level or on file-level.

It is possible to interact with each point on the plot shown in the main frame. Depending whether the X-axis is showing the information at commit-level or at file-level, a pop-up menu that summarizes the information related to that point of the plot is shown. Figure 6.3d shows this pop-up window that lists the 20 changes corresponding to a select point of the plot, in which the X-axis was set at file-level. Figure 6.3c presents the main KPIs on a pane with two bar-charts and some textual information below. The barchart on the left-hand-side shows the distribution of the workload by activity. The barchart on the right-hand-side shows the number of authors who participate on each activity (i.e., have worked at least once on that activity). The text in the middle provides KPIs about the overall project. Figure 6.3b shows the information presented by the fourth frame. Here it is possible to obtain details on all the KPIs about the users (e.g. on which activities they have worked on and how much), the overall-project (e.g., the distribution of workload) and the activities (e.g., how many different activities are in the project and what is the relative work done on them).

With this information at hand and their knowledge of the domain, the project manager can investigate, for example whether the team is *de facto* following a specific software methodology.

6.4.2 Analyses of real-life open-source projects

We use ActiVCS to analyze real-life open source projects from GitHub. We chose thirteen projects of different size, age, user counts, and programming languages. Selection criteria included having, *i*) a representative variety in terms of programming language, *ii*) variety in the size, *iii*) at least having two similar projects, *iv*) highly active versus inactive, and *v*) fast versus slow growing projects.

This resulted in the following projects. *MPAndroidChart* (*MP* is a visualization library for Android platforms. *Torque2D* (*Torque*) is a software engine for the development of video games. *Openage* (*openage*) is an open source clone of the Age of Empires II engine. *Incubator-dubbo* (*inc*) is a RPC (remote procedure call) framework for Java. *jekyll* is a blog-aware written Ruby that generates websites from user content. *scrapy* is a Python based framework to extract data from 24 websites *brew* is a software that automatically installs missing packages for MacOS and Linux operating systems. *Algorithms – Java* (*Java*) is a collection of different Java algorithms. *Flask* (*flask*) is a lightweight Web Server Gateway Interface (WSGI)

web application framework created in Python. *Tablesaw* is a framework for the transformation and visualization of data, implemented in Java. *Okhttp* (*ok*) is an HTTP client for Java and Android. *Retrotfit* (*retro*) is another HTTP client for Java and Android. *editor.js* (*editor*) is a JavaScript based editor software for the creation of documents and the transformation into JSON format.

Table 6.3: KPIs giving insights on real-life projects.

Name	COM	PW	NAP	NTP	PWS	PIS	C	D	T	U
MP	2012	8552	86	8	0.8	0.5	7352	53	24	351
Torque	970	7978	46	9	0.75	0.34	5410	134	241	1561
openage	3136	10529	168	9	0.74	0.5	7697	596	366	1126
inc	3308	31667	240	8	0.69	0.58	16116	6	8232	164
jekyll	10388	14369	1028	9	0.69	0.6	4085	1384	1492	7001
scrapy	7045	15014	409	9	0.68	0.57	7995	612	2784	525
brew	18410	35299	815	6	0.65	0.46	23087	1276	7348	3161
Java	755	902	175	5	0.65	0.58	636	5	7	203
flask	3505	5679	623	10	0.65	0.65	1950	206	1027	481
tablesaw	1893	23700	39	7	0.63	0.39	8779	11149	2188	652
ok	3826	11567	245	7	0.63	0.51	5837	225	3348	260
retro	1721	5168	173	7	0.6	0.43	2467	69	1039	103
editor	494	2337	27	7	0.56	0.24	927	314	0	814

Table 6.3 shows the KPIs resulting from these projects. Columns contain the following information: number of commits (COM), Project Workload (PW), Type of change Workload (TW), Number of Authors in Project (NAP), Number of Types of work in Project (NTP), Specialization of Project Workload (PWS), Specialization of Author in each Activity Type (PIS), code (C) , documentation (D) , testing (T) and unknown (U). Entries have been sorted by the Specialization of Project Workload (PWS). This means that, for instance, resources of project *MP* are highly occupied. Therefore, load balancing should be considered if the managers want to improve the capacity of the team to handle new tasks in the near future.

Finally, the types of work code (C), documentation (D), testing (T) and unknown (U) that are displayed in the table make up the most frequently identified types that were detected with ActiVCS. As expected from software development processes, the main workload in most analyzed projects were coding activities. One exception to this observation is the jekyll project. ActiVCS has detected 7001 file changes with an unknown type within the Ruby software application. Projects like these would provide a solid basis for the identification of additional file types. The total amount of workload that was captured across all projects is 172761. The number of file changes which could not be classified by ActiVCS and were marked as unknown is 16402. This represents 9,49% of the total workload. Excluding the jekyll project from the analysis would reduce this margin to 5,94%. Many of the identified projects neglect the creation of documentation. All of the analyzed projects with a workload value for coding activities of 8000 and more also have a noticeable amount of registered testing

activities. This could refer to the fact that large software development processes have a need for automated test activities which must be frequently updated.

6.4.3 Discussion

ActiVCS allows to visualize a number of KPIs and observe patterns in the project work. It also provides a tool that can be used for checking conformance to existing software development methodologies. Figure 6.4 shows a screenshot of the types of work Code and Test, taken from the *ok* project, described previously. A project manager can now check that work of the type Code and Test was consistently done throughout the lifetime of the project. Moreover, it is possible to observe that Code and Test were active together most of the time, with Code starting earlier. A typical development methodology that presents such pattern is *agile*.

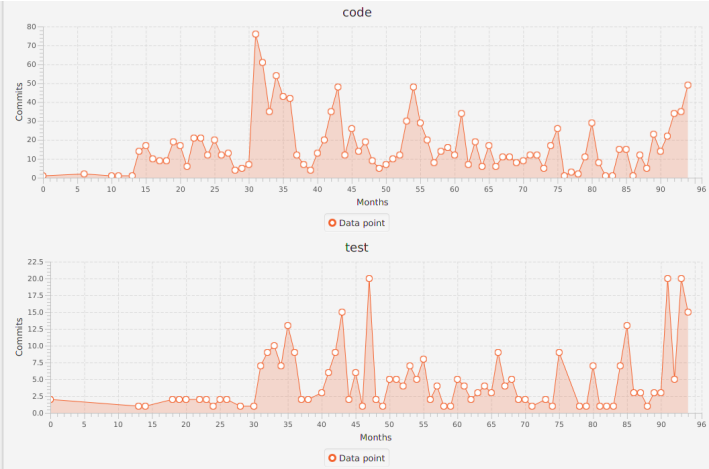


Fig. 6.4: Zoom on the Code and Test types of work from the *ok* project

This might confirm that the *de facto* software development method corresponds with what the organization has decided. Alternatively, organization might be following a Waterfall development model. In that case, this pattern may point at a lack of control on the project. The managers can use their domain knowledge along with the provided *factual* information for better decision making.

6.5 Conclusion

This paper provides an artifact for analyzing event logs from VCS. Our tool is able to visualize the type of work that was executed in a software development process. Starting from fine-grained changes in the different versions of files, it allows to understand what activity was done and when. It also provides important KPIs, such as the effort distribution. These features are important for managers to understand whether the project is deviating from target goals, and in case take corrective actions.

In the future, efforts will be made to integrate the ActiVCS tool with the Gantt chart miner from Bala et al. (2015). This would offer to the project manager complementary views on the current status of the project. We plan to conduct user studies with managers in order to receive more feedback from domain experts. Finally, we have already conducted a study on selecting KPIs for software development and we plan to incorporate them in the ActiVCS tool.

Chapter 7
Article 5: Software Process Evaluation from User Perceptions and Log Data

Authors:

Damjan Vavpotič, Saimir Bala, Jan Mendling and, Tomaž Hovelja

Published in:

Journal of Software: Evolution and Process (second round of review)

Abstract:

Purpose – Companies often claim to follow specific software development methodologies (SDM) when performing their software development process. These methodologies are often supported by dedicated tools that keep track of work activities carried out by developers. The purpose of this paper is to provide a novel approach that integrates analytical insights from both the perceptions of SDM stakeholders and software development tools logs to provide SDM improvement recommendations.

Design/methodology/approach – This paper develops a new process improvement approach that combines two significantly different sources of data on the same phenomenon. First, it uses a questionnaire to gather software development stakeholder SDM perceptions (managers, developers). Second, it leverages process mining to analyse software development tools logs to obtain additional information on software development activities. Finally, it develops recommendations based on concurrent analysis of both sources.

Findings – Our novel process improvement approach is evaluated in three directions: does the presented approach (RQ1) enable managers to gain additional insights into employees' performance, (RQ2) deliver additional insights into project performance, and (RQ3) enable development of additional SDM improvement recommendations. We find that integrated analysis of software development perception data and software development tools logs opens new possibilities to more precisely identify and improve specific SDM elements.

Research limitations – The evaluation of our novel process improvement approach follows a single case study design. Our approach can only be used in enterprises in which software development tools logs are available. The study should be repeated in different cultural settings.

Practical implications – We practically show how concurrently analysing data about developer SDM perceptions and event log data from software development tools enables management to gain additional insights in the software development process regarding the performance of individual developers.

Originality/value – The main theoretical contribution of our paper is a novel process improvement approach that effectively integrates data from management and developer perspectives and software development tools logs.

Keywords – software development methodology, evaluation, stakeholder perceptions, software development tool logs, case study

7.1 Introduction

Software development methodologies (SDMs) highly influence the management of software projects. Adopting a specific methodology can determine to what extent performance goals such as cost, quality and timeliness are achieved (Vavpotic et al., 2020). Existing studies show (Kneuper, 2008; Loon, 2007; Laporte et al., 2015) that the adoption of SDMs is critical for success of development teams, but not easy to apply in practice. More specifically, it is often the case that a specific methodology is either applied partially or incorrectly, is not suitable for the enterprise due to incompatible technical characteristics, or has unsatisfactory impact on software quality, cost and development time (Chan and Thong, 2009; Balasubramanian and Mnkandla, 2016). Therefore, understanding the performance of specific SDM elements is crucial for management's capability to continuously improve software development processes (Vavpotic and Hovelja, 2012). These SDM elements include SDM activities as well as artefacts produced and tools used in these activities.

In order to gain knowledge of the current state of SDMs within software companies, newest approaches in literature have focused on the analyses of their constituting SDM elements (Hovelja et al., 2015; Vavpotic and Bajec, 2009; Gill et al., 2018). These studies focus on perceptions of various stakeholders about software process activities, tools and roles and neglect software development process data from supporting tools. This additional source of information regarding development process became available in recent years with widespread use of tools used in software development activities like issue tracking, requirements management, test management etc. Such tools store valuable data about actual execution of the development process and can provide additional insights into SDM and its elements, thus complementing stakeholder perceptions (Choetkiertikul et al., 2017; Mäntylä et al., 2016; Destefanis et al., 2016; Ortu et al., 2015). What these recent works do not address is to what extent both sources of information complement each other.

In this paper, we propose a novel approach to evaluate a process improvement approach of the performance of SDM activities through concurrent analysis of stakeholder perceptions and relevant software development tools logs. In this way, we can consider both sources of insights about performance of specific SDM activities and consequentially improve organisational learning (Annosi et al., 2020). Based on this analysis we develop SDM improvement recommendations. We evaluated our approach through a case study in an Austrian SME software development company.

This work seeks answers to the following research questions:

RQ1. Which new insights into employees' performance for management does the proposed approach provide?

RQ2. Which new insights into project performance for management does the proposed approach provide?

RQ3. Which new insights does the concurrent analysis of stakeholder SDM perception data and software development logs provide and which SDM improvement recommendations would be otherwise unavailable?

The rest of the paper is organized as follows. Section 2 provides background on the state of the art. Section 3 describes the methodology. Section 4 applies the

methodology on a case study from industry and shows the results. Section 5 discusses the results and their implications. Section 6 concludes the paper.

7.2 State of the Art

This paper builds upon existing research on Software Development Methodology (SDM) evaluation and process mining. In the following, we briefly introduce these fields of research.

7.2.1 SDM evaluation

Existing research shows that using SDMs improves productivity of software development enterprises and quality of the developed software (Bass, 2016; Zdravkovic et al., 2015; Tuape et al., 2021). This is achieved by increasing enterprises' ability to transfer knowledge between employees, systematically manage software development process, etc. (Avison and Fitzgerald, 2006; Fitzgerald, 1998; Hovelja et al., 2015; Riemenschneider et al., 2002). However, it is not enough that an enterprise only describes its SDM in a document; the developers need to use it consistently in their everyday work. The use of SDM was often a topic of research in the last decades, since SDM adoption among software developers was relatively low and the developers often preferred different ad hoc approaches (Aaen, 2003; Fitzgerald, 1996; Huisman and Iivari, 2006). According to Destefanis et al. (2016) developers are the key factors for the success of a software development process, not merely as executors of tasks, but as protagonists and core of the whole development process. Therefore, understanding their perceptions is of key importance in SDM improvement.

The use of SDMs in enterprises can be analysed with the help of different approaches based on Technology Acceptance Model, Theory of Planned Behaviour, etc. (Aboelmaged, 2010b; Venkatesh and Davis, 2000; Wang et al., 2013). One of the foundational theories in the field of SDM evaluation is diffusion of innovations (Rogers, 2003), according to which SDM is considered as innovation that developers adopt (Gallivan, 2003; Green et al., 2005; Huisman and Iivari, 2002). To obtain information about the studied SDM and its elements the aforementioned studies focused on SDM perceptions held by different stakeholders, namely developers, managers, users, etc. to measure characteristics like level of use, assimilation, social and technical suitability, developer satisfaction and impact on performance (Atkinson, 1999; Cooper and Zmud, 1990; Rogers, 2003; Vavpotic and Bajec, 2009; Vavpotic and Hovelja, 2012; Hoda et al., 2011). By considering opinions of multiple stakeholders our approach fits well with agile SDM that emphasize team autonomy, decentralized decision making, flexible scope and managing a high degree of requirement changes (Scott et al., 2021; Jørgensen, 2019).

Detailed insight into SDM and its elements can be gained by comparing the perceptions of different stakeholders regarding the same SDM and/or its element (Hovelja et al., 2015). Another important theoretical development in the field was that SDM should be studied at the level of its constituent elements like activities, tools, roles, produced documents, etc. and not only as a whole. This improved the understanding of suitability of studied SDM elements for a certain development team and enabled comparison between the studied SDM elements; thus, allowing enterprises to better pinpoint problematic elements of SDM, prepare focused improvements and examine the link between a specific SDM element and overall project success (Atkinson, 1999; Hovelja et al., 2015). Such development is in line with findings in the field of situational method engineering (Karlsson and Ågerfalk, 2009; Ralyté et al., 2003; Gill et al., 2018; Malinova et al., 2022) that constructs a custom SDM from those SDM elements that fit with characteristics of certain development team and other situational factors (internal and external enterprise's environment). One of the key goals of these improvements was to tackle low SDM adoption rates which process and practice improvement initiatives often suffer from (Fontdevila et al., 2019). Such low adoption rates clearly indicate the inadequacy of existing SDM evaluation models and the need to develop better ones, for instance based on the analysis of log data.

7.2.2 Analysis of Event Logs

Process mining is a discipline that gained popularity in the last decade. The goal of process mining 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. Van der Aalst (van der Aalst, 2016), defines four partially overlapping perspectives of a business process that are widely used in business process modelling (BPM) community. First, the *time perspective* aims at analyzing time and frequency of process events. Second, the *case perspective* aims at identifying properties of process cases. Third, the *organizational perspective* aims at analyzing the event log to gain transparency on the resources involved in the process. Fourth, 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. These aspects can be, for example, related to the control flow (i.e., the various steps used by the company for generating value), resources (i.e., handover of work among the different process participants), activities (i.e., how the work is broken down into several tasks), and data (i.e., which artifacts are produced and consumed by the process).

Process mining is becoming widely adopted. While researchers can use general purpose data-mining tools like R, Orange, IBM SPSS, etc., a considerable number

of specialized tools such as Celonis[1], Disco[2], Apromore[3], minit[4], LANA Process Mining[5], etc. have been developed. A comparison of the current process mining tools can in found in Viner et al. (2021). In academia, log analysis is an established endeavor when it comes to gaining insights into software development. Recent works (Marques et al., 2018; Bala et al., 2015, 2017b; Bala and Mendling, 2018; Bala et al., 2020) demonstrate potential for understanding the different perspectives of processes. Process mining has also been considered as a research method (Grisold et al., 2020) with applications for analyzing process improvement methods (Gross et al., 2019) and reporting software experiments (Revoredo et al., 2021).

Continuous evolution of data analyses techniques (such as process mining), their adoption in new domains and their integration into other research methods in different mixed-methods approaches, indicate the future opportunities that novel SDM evaluation approaches can exploit. In this context, the necessity and importance of our research that attempts to integrate current state of the art process mining approaches and SDM evaluation approaches becomes clear, since the integration can deliver insights that neither of the two approaches alone can provide.

7.3 Proposed Approach

7.3.1 Overview

In this section, we present our novel process improvement approach. The development of our approach followed principles from design science (Hevner et al., 2004) and situational method engineering (Henderson-Sellers et al., 2014) as research methods. The proposed approach builds on the established SDM evaluation approaches. Gill et al. (2018) emphasize the need for project to meet stakeholders' expectations thus analysis of different stakeholders' perceptions must be an integral part of a modern SDM evaluation approach. Vavpotic and Bajec (2009) showed the importance of evaluating SDMs by their constituent parts and not only as a whole. Vavpotic and Hovelja (2012) presented the usefulness of considering economic success metrics of software development in SDM evaluation. However, unlike existing approaches that base their SDM evaluation solely on stakeholder perceptions, the proposed approach complements stakeholder perceptions with data from software development tools logs. The approach comprises four phases as shown in Figure 7.1. To our knowledge, this is the first attempt to combine perceptions and log data in SDM evaluation. Next, we elaborate on the phases of the approach.

[1] https://www.celonis.com

[2] https://fluxicon.com/disco

[3] https://apromore.org

[4] https://www.minit.io

[5] https://lanalabs.com/en

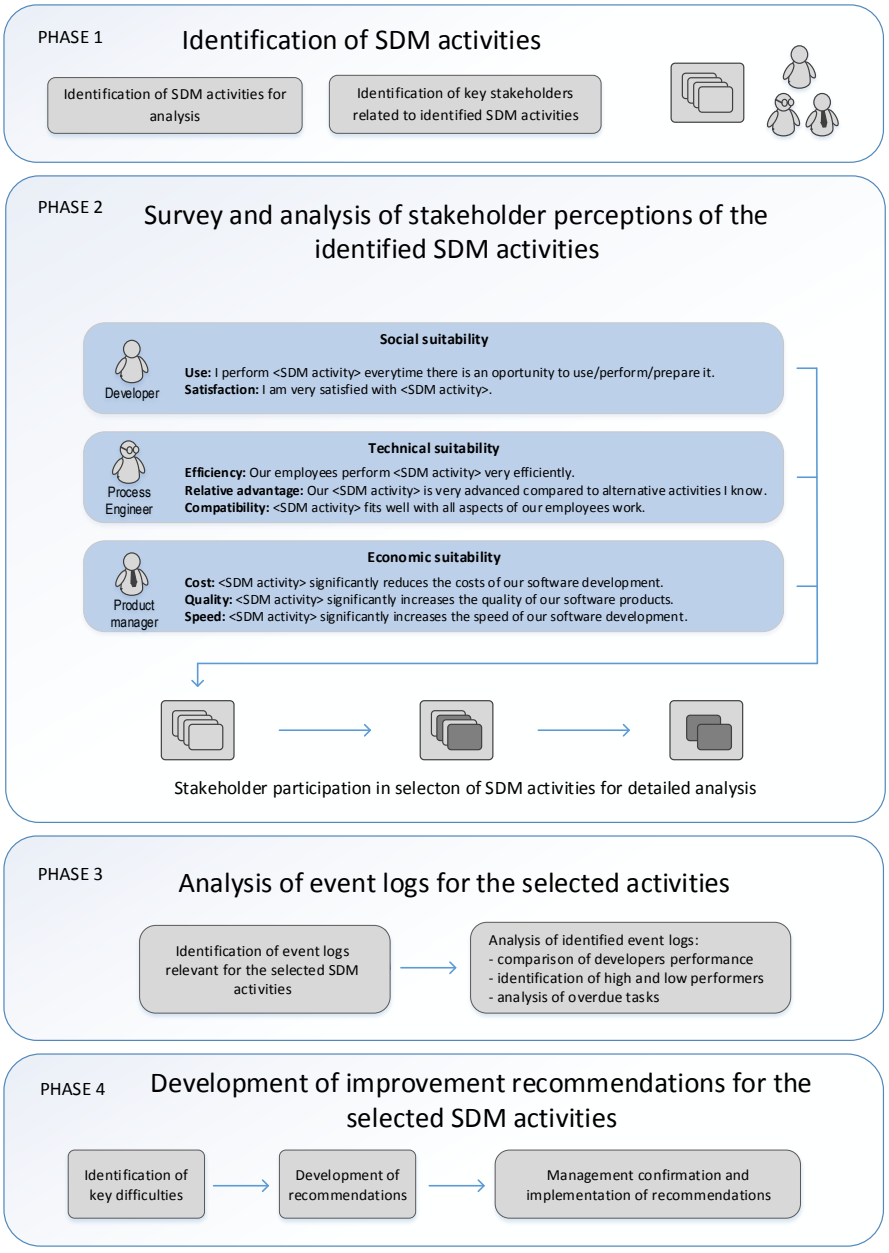


Fig. 7.1: The proposed SDM evaluation approach

7.3.2 Phase 1 – Identification of SDM Activities

In Phase 1, SDM activities for evaluation are first identified and catalogued with the help of key stakeholders knowledgeable about their SDM (e.g., lead developers, managers). In identification phase we focus on activities since log data are typically organized around them. Roles are considered indirectly as specific activities are typically performed by a specific role. If an enterprise formally follows a specific standard SDM (like Scrum, IBM Rational Unified Process, Kanban, etc.), the cataloguing starts with a predefined list of typical SDM activities. However, the key stakeholders still have to point out which SDM activities are actually used and identify possible additional activities in the enterprise and only those are included in the final catalogue. After the SDM activities are catalogued the management has to identify the actual stakeholders performing these activities in a specific project that will be evaluated. Managers responsible for a certain project and/or product should be the ones to evaluate the SDM activities used in project and/or product development from managerial perspective. Similar logic is also applied to other project stakeholders.

7.3.3 Phase 2 – Survey and analysis

Phase 2 starts with the creation of questionnaires that include each SDM activity catalogued in Phase 1 based on the template questions (shown in Fig. 1). Different questionnaires are created for each stakeholder. In line with the established measures from studies presented in the literature review the developers evaluate SDM activity social suitability (average of use and satisfaction). Similarly, product managers evaluate SDM impact on iron triangle measures of economic suitability (average of cost reduction, speed increase and quality increase). The questions are written in form of statements where answers are given on 7-point Likert scale.

To analyse stakeholder perceptions, we position SDM activities in a multidimensional space where each dimension represents a specific stakeholder perspective. It is typically difficult to further improve SDM activities that are considered highly beneficial by all stakeholders since they are all already satisfied with them. On the contrary, SDM activities that are considered unbeneficial by all stakeholders have high potential for improvement, since it is likely that most stakeholders will support their change. The SDM activities where perceptions of stakeholders are very low and/or greatly differ require further examination where log analysis (Phase 3) has an important role to identify appropriate improvement actions.

7.3.4 Phase 3 – Analysis of software development tools logs for selected activities

In Phase 3, log analysis is performed focusing on selected SDM activities. Log analysis provides additional information about certain SDM activities such as the time developers spent their execution of this activity and the quality of the execution. To guide the log analysis, we followed the PM^2 methodology (van Eck et al., 2015). PM^2 consists of six stages, namely (1) planning – in which the goals and the target processes are identified, (2) extraction – in which trace data of the identified processes are collected, (3) data processing – in which data is transformed into event logs, (4) mining & analysis – in which the event logs are analysed by means of process mining techniques, (5) evaluation – in which results of process mining are interpreted and linked to improvements ideas, and (6) process improvement & support – in which the improvement ideas are implemented.

When it comes to analysing SDMs, PM^2 can be adapted as follows. First (1), we start the log analysis with identification of software development tools logs relevant for the selected SDM activities. Second (2), event logs are gathered from the respective information systems. Then an inspection of the information contained in these logs is performed to confirm that relevant information about resources and activities is present. Third (3), data processing is performed. In fact, different software development tools have their own logging mechanisms and often use customized representations of event logs. Such logs are rarely in the format needed by process mining tools. Therefore, pre-processing is required in order to transform these logs into a format which can be further analysed by process mining techniques (van der Aalst et al., 2004; Leemans et al., 2014). Therefore, the data is stored into a form of a flat table with explicitly labelled notions of case, activity and timestamp. This representation can be easily transformed into the standard XES (Verbeek et al., 2010) format for event logs or be directly consumed process mining tools.

Fifth (5), we analyse instances of activities preformed during development that are described by at least three aspects: activity type (bug fixing, implementing new code, etc.), actor (employee ID1, employee ID2, etc.), and timestamp (date1, date2, etc.). These three aspects enable us to analyse identified software development tools logs to compare developers' performance, identify high and low performers and analyse overdue activity instances for specific types. To conduct these analyses depending on log structure different process mining techniques can be used like analyses of case durations, filtering on users, variant analyses. Moreover, some software development tools logs contain additional information like priority of specific types of activities, pointers to related or preceding activities, detailed unstructured descriptions of preformed activities and comments, etc. If needed, these can be used to gain additional insights into the software development process as a whole as well as into specific SDM activities (Hamdy and Ezzat, 2020). As we focus on a single case, we use process mining to support exploring these different aspects related to the selected SDM elements.

Finally (5 & 6), several analyses artifacts resulting from the applications of the aforementioned process mining techniques such as process maps, case/activity/user

frequency reports and durations of activities are collected. For instance, for the SDM element of bug-fixing, we can observe how often this was worked on by a certain user and whether it took the user one or more sprints to complete his task. These kinds of insights are then compared to the perceptions collected via the questionnaire to help resolving conflicting perceptions. For instance, it may the case that management does not perceive correctly the time and effort spent by developers on bugs.Equipped with evidence-based results from event logs, it is possible to align the perceptions of the different stakeholders.

7.3.5 Phase 4 – SDM improvement recommendations

In this phase, systematic and comprehensive data from Phase 2 and 3 is presented to external and/or internal SDM experts (if enterprise has sufficient competences). Their task is to identify key difficulties of the current development process based on the concurrent analysis of stakeholders SDM perceptions and software development tools logs. Next, based on identified key difficulties they prepare recommendations for process improvement using their knowledge and experience. Different problem-solving techniques like brainstorming, simulation, what-if-analysis, creativity techniques, etc. that help to support the act of improving can be used in this phase. An example of recommendations developed by such approach is presented in Case study section. Finally, the recommendations need to be presented, discussed with and confirmed by the management. After management acceptance of the proposed recommendations a timeline for their implementation can be developed.

7.4 Case Study

7.4.1 Case study description and research methodology

The proposed approach was tested in an Austrian SME software development company located in Vienna. The company can be considered a typical central European SME. The company develops a software platform in the field document composition, workflow and document distribution. Their customers come from eight different central and southern European countries. Seven developers and a product manager worked on company's customer communications management (CCM) platform in the studied year. The product manager oversaw the quality of the product features, software process speed and cost. All developers participating in our study were experienced developers having worked as members of the same team for at least two years.

The company uses Scrum (Takeuchi and Nonaka, 1986) an agile, lightweight, iterative and incremental methodology often used in software development. Scrum organizes work in iterations called sprints. The length of sprint is set in advance

and is typically between one week and one month, with two weeks being the most common. As in most Scrum implementations each sprint starts with sprint planning and ends with sprint review (presenting work to stakeholders) and sprint retrospective (identifying issues and improvements related to the development process).

The company uses the JIRA issue tracking software to monitor tasks completion by their developers. We obtained data for two activities (implementing new code, bug fixing) from 26 sprints for 5 developers working on the studied activities between 2016-01-11 and 2017-01-08. The JIRA event log was pre-processed to generate a flat table. In tabular datasets, we created additional attributes to track the number of performed activities. We aggregated the information into two-week intervals reflecting company sprints. This helped us understand if the studied activities were completed on schedule (in one sprint) or not. Furthermore, JIRA allows us to track activity completion rate on the level of the whole project and on the level of individual developers.

An exploratory case study design was employed to assess the proposed approach (Runeson and Höst, 2009). Such design is appropriate when it captures the circumstances and conditions of an everyday or commonplace situation (Yin, 2009). The studied company represents a typical SME in the software development industry. To collect the data, we conducted interviews with management, directly observed their workday, surveyed all developers working on the observed project and collected the corresponding JIRA event logs. The study results were presented and validated by the SME's management.

7.4.2 Identification of SDM activities (Phase 1)

In the first phase, we identified ten key activities (Table 7.1) of the company's software development process by interviewing the lead developer and product manager. They also specified the other developers who worked on this project. The identified ten activities cover key parts of the SDM used in the company. These include requirements specification, planning, coding with unit testing and maintenance. We also identified JIRA logs that were used in one or more of the identified ten activities in the observed period.

7.4.3 Survey and analysis of stakeholder perceptions of the identified SDM activities (Phase 2)

Based on the template questions (Fig. 1) we created two separate questionnaires, one for the seven developers and one for the product manager. For the purpose of the analysis, we visualize the results of the survey of the two identified stakeholders on a scatter chart comprising the developers' perspective as a horizontal dimension and the product manager's perspective as a vertical dimension (Figure 7.2).

Table 7.1: Identified activities and availability of JIRA logs

Development activities	Availability of JIRA logs related to the activity
Defining list of specifications (in Excel)	No
Defining feature specifications	Yes
Using prepared feature specification	Yes
Using prepared bug specification in Jira	Yes
Estimating (planning poker)	No
Assigning from Backlog to sprint	No
Implementing new code	Yes
Bug fixing	Yes
Sprint planning (bi-weekly)	Yes
Stand-ups (Daily meetings)	No

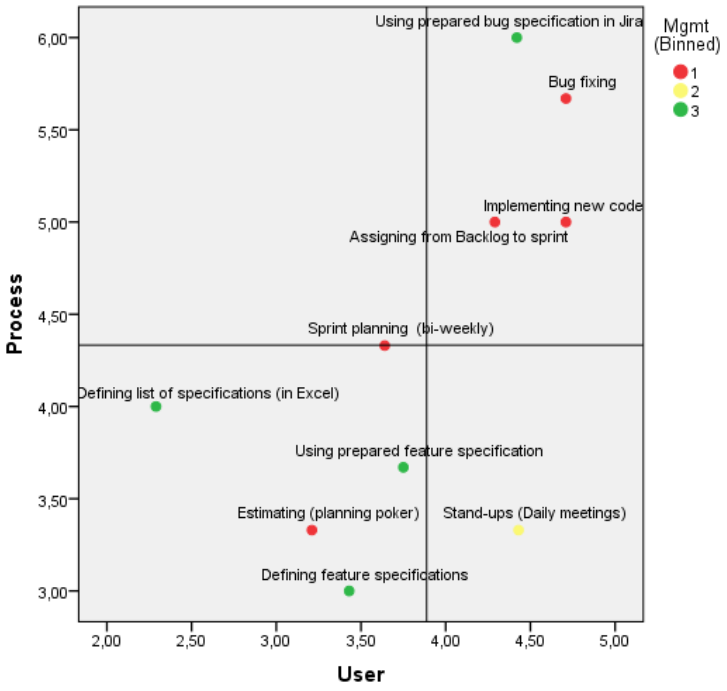


Fig. 7.2: Scatter chart of stakeholder views on the identified development activities

To better understand the views of the two stakeholders on specific SDM activities four quadrants were formed in the scatter chart by using the average scores of developers' and management perspectives. The four quadrants follow the logic suggested by Vavpotic and Hovelja (2012): the first quadrant contains managerially and socially unsuitable SDM activities, the second quadrant contains managerially unsuitable but socially suitable SDM activities, the third quadrant contains managerially suitable but socially unsuitable SDM activities and the fourth quarter contains managerially and socially suitable SDM activities. Each SDM activity is presented by a point in one of the four quadrants of the scatter chart. The red colour points represent the five activities that were selected for detailed evaluation. Management limited the selection of activities for analysis and improvement to maximum half of all SDM activities (i.e. five activities) due to limited capabilities and resources available for the improvement processes. The selection was done as follows. All SDM activities that were unbeneficial for both stakeholders (Sprint planning and Estimating) were selected for further analysis from the first quadrant (I). From the second quadrant (II) the two SDM activities (Implementing new code and Bug fixing) that consume significantly more work hours per sprint were selected. While from the third quadrant (III) one SDM activity was selected (Defining list of specifications) that was significantly lower graded by the developers.

User stories described in *Defining feature specifications* activity were prioritized and assigned to a list (backlog) in *Defining list of specification* activity. During the interviews we learned that that the main reason for dissatisfaction of developers with *Defining list of specification* activity was that management was often adjusting prioritizations of user stories to ensure customer satisfaction when customer priorities changed. Due to regular shifts in customers' priorities, this approach caused difficulties for developers who often had to de-prioritize user stories related to architecture and other technical aspects. The dissatisfaction of the developers was further exacerbated by the fact that they felt team autonomy and team ability to plan its work was affected. Moreover, the list of specifications was maintained in MS Excel, which caused additional difficulties for the developers due to delayed list synchronization.

The *Estimating (planning poker)* and *Sprint planning (bi-weekly)* activities were performed by the team each sprint as follows. First, the team performed *Estimating (planning poker)* to determine the effort needed for each task. Next, in *Sprint planning (bi-weekly)* these tasks were divided among the team members. According to the management, these activities were among the least efficient activities. The management was dissatisfied by often exceeded time limits as tasks were not completed in a single sprint as planned. The dissatisfaction of the team was mostly the result of increased but unsuccessful management pressure to raise performance.

Implementing new code and *Bug fixing* activities were time-wise the main activates in the development process. While the developers were satisfied with their performance, seeing themselves as highly capable performers and valuable employees, the management did not share their view and considered these two activities similarly inefficient as *Estimating (planning poker)* and *Sprint planning (bi-weekly)*. The main cause of management dissatisfaction was again poor compliance with task completion time limits set by the team.

7.4.4 Analysis of event logs (Phase 3)

We analyzed JIRA logs related to four out of the six selected development activities with available JIRA logs to gain new insights especially where the views of management and developers differed. *Defining list of specification* was maintained in MS Excel, thus no JIRA logs were available. First, we analyzed the frequency of bugs and tasks not completed in each examined sprint (26 sprints = 1 year) to get additional insights in *Estimating (planning poker)* and *Sprint planning (bi-weekly)* activities. Figure 7.3 shows the number of not completed bugs per sprint and Figure 4 shows the number of not completed tasks per sprint. In both cases, there is considerable variability in unfinished tasks/bugs between sprints. This exposes planning and performance difficulties as well as an unstable development process.

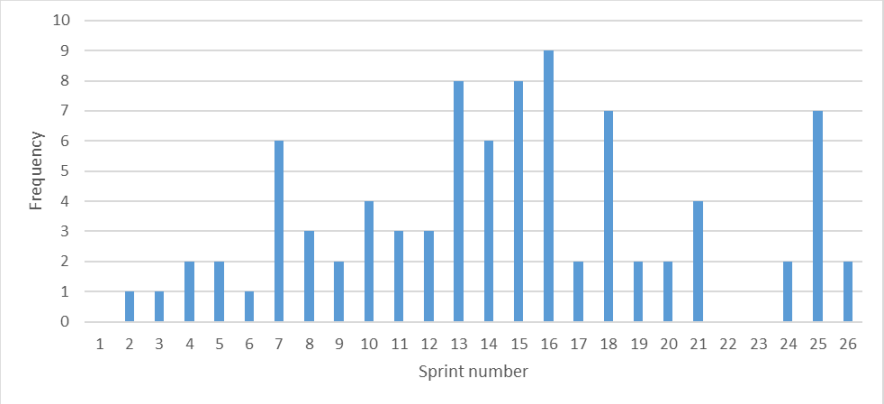


Fig. 7.3: Frequency of bugs not completed in a single sprint by sprint number

To gain additional insights in *Bug fixing* and *Implementing new code* activities we compiled Table 7.2. It shows the individual developer performance breakdown in number and percentage of bugs fixed and new code implementations (tasks completed) in a single sprint and those completed in more than one sprint.

The analysis of performance clearly shows significant differences between individual developers. Additionally, there were significant differences in on-time completion rate between *Bug fixing* and *Implementation of new code*, showing that if a developer is a top performer in one activity he can still struggle in the other. The most extreme cases were Developer 2 and Developer 5. On one hand, Developer 2 was the best performer in *Implementation of new code*, but was below average in *Bug fixing*. On the other hand, Developer 5 was the best performer in *Bug fixing*, but the worst in *Implementation of new code*. Developers 3 and 4 were below average in both activities. Additionally, Developer 3 was relatively worse in *Implementing new code*. Developer 1 was above average in Bug fixing, however he did not participate in *Implementing new cod*e.

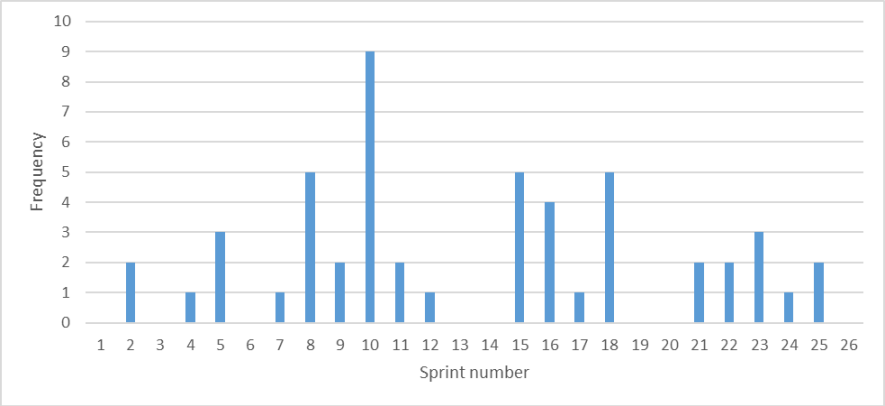


Fig. 7.4: Frequency of tasks not completed in a single sprint by sprint number

Table 7.2: Bugs and tasks completed in a single sprint and in more than one sprint for each developer
* = Developer 1 was performing only Bug fixing (bugs), while the other four developers were performing Implementation of new code (tasks) and Bug fixing.

		Developer 1	Developer 2	Developer 3	Developer 4	Developer 5	Total
Number of bugs	Completed in a single sprint	22	22	17	39	30	130
	Completed in more than one sprint	12	16	17	33	9	87
	Total	34	38	34	72	39	217
Percentage of bugs	Completed in a single sprint	64,7%	57,9%	50,0%	54,2%	76,9%	59,9%
	Completed in more than one sprint	35,3%	42,1%	50,0%	45,8%	23,1%	40,1%
Number of tasks	Completed in a single sprint	*	78	18	16	19	131
	Completed in more than one sprint	*	10	15	8	18	51
	Total	*	88	33	24	37	182
Percentage of tasks	Completed in a single sprint	*	88,6%	54,5%	66,7%	51,4%	72,0%
	Completed in more than one sprint	*	11,4%	45,5%	33,3%	48,6%	28,0%

This information derived from log analysis allowed us to gain the following additional insights for the four selected development elements with available JIRA event logs.

The logs show that there is significant variability in task and bugs completion rates between developers, thus the management perception that developers' performance was not satisfactory was only partially correct. JIRA event logs helped us determine that there are some great performers, some average performers and some poor performers. To tackle this problem successfully, these differences have to be better managed in order to improve *Estimating (planning poker)* and *Sprint planning (bi-weekly)* activities. If this issue is properly addressed, management can also expect improvements in *Implementing new code* and *Bug fixing* activities.

7.4.5 Development of improvement recommendations for selected SDM activities (phase 4)

Based on the above presented analysis in Phases 2 (section 7.4.3) and 3 (section 7.4.4) of the proposed approach, the following improvement recommendations were prepared for implementation by management (see Table 7.3).

7.4.6 Reflection workshop with company management

The identified difficulties and prepared recommendations were presented and discussed with the company management in order to assess their validity and usefulness. The management provided feedback as follows:

- The management strongly agreed with the first identified difficulty and the recommendation of the experts to give developers more autonomy and direct contact to the customer. They modified their requirements acquisition process so that it was performed by dedicated team members and not by the management.
- The management strongly agreed with the second identified difficulty and followed the recommendation to move the list of specifications to JIRA. This simplified the development process and reduced the process quality issues.
- The management strongly agreed with the third identified difficulty, but only partially followed the third recommendation. They did lessen the pressure on the developers but did not change the remuneration by linking it to performance. To lessen the pressure, they organized a yearly meet-up with core customers where developers themselves could estimate the time needed for development of requested features. This prevented the management to overpromise and overcommit to the core customers.
- The management strongly agreed with the fourth identified difficulty and followed the recommendation by providing the relevant performance data mined from JIRA event logs about the efficiency of task distribution between the developers to the team to improve their sprint planning.
- Management confirmed that this is a novel approach, and they could not specify any other similar approach. Existing SDM approaches (Scrum, Kanban, etc.) do not use process mining to evaluate SDM elements.
- Management confirmed that the proposed approach was useful and relevant in the context of SDM.

Overall, we found that management not only agreed with the identified difficulties but also found the results highly relevant to improve their process. As they stressed in the interview, despite the rich set of features provided by current supporting tools such as JIRA, there is still an unmet need for a higher-level integrated reporting process which can give better insights on the overall status of the SDM.

Table 7.3: Identified key difficulties and SDM improvement recommendations

	Identified key difficulties		SDM improvement recommendations
D1	Adjusting prioritizations of user stories to ensure customer satisfaction when customer priorities changed	R1	Currently manager is acting as an intermediary between developers and the customer. Management needs to start to act as connector that encourages and helps the team to establish direct collaborations between them and the customer's product owners (lead users). This would increase the team's autonomy and ability to plan its work and at the same time enable the team to better consider the architectural and other technical aspects when prioritizing user stories.
D2	Delayed synchronization of list of specifications in MS Excel	R2	The list of specifications that is currently managed in MS Excel should be moved to JIRA. This will integrate backlog with task planning and task completion administration.
D3	Unsuccessful management pressure to raise performance	R3	Instead of pressure to stop undesirable behaviour we suggest management focuses on rewarding desirable behaviour. Thus, we suggest to revamp the reward scheme and make a desired but still achievable percentage of tasks/bugs completed in time an important metric of remuneration.
D4	Often exceeded estimated sprint tasks time limits in implementing new code and bug fixing	R4	To address this difficulty, we propose using individual developer performance data mined from JIRA logs to increase the efficiency of task distribution between the developers. The developers who are significantly more efficient at bug fixing should predominantly work on bug fixing, while the developers who are significantly more efficient at implementing new code should predominantly work on new code. Additionally, the first, the second and the third improvement also partially address this difficulty.

7.5 Discussion

7.5.1 Addressing research questions

We successfully addressed the three research questions posed in the paper and found that the proposed approach enables managers to gain additional insights into employees' performance (RQ1), since the information from software development tools logs allows for an improved performance management of individual developers by knowing their exact on time sprint task/bug completion rates (Table 7.2). Next, we confirmed that the proposed approach delivers additional insights into project performance through monitoring the frequency of total tasks/bugs not completed in a single sprint (RQ2). Finally, we ascertained that the concurrent analysis of software development perception data and software development tools logs enables additional SDM improvement recommendations (RQ3).

The first and second recommendation (R1 and R2 in Table 7.3) are based on perception analysis alone since log analysis does neither help us identify the first two key difficulties (D1 and D2 in Table 7.3) as important software development issues nor it can be used to improve the two recommendations (R1 and R2). However, the development of the third and fourth recommendations (R3 and R4 in Table 7.3) would not be possible without the concurrent analysis of perception and log data. R3 and R4 require software development tools logs to provide exact sprint task and bugs completion rates needed for improved individual developer performance management, while the perceptions provide the context about managers' satisfaction with specific performance levels.

7.5.2 Contribution

Our case study showed several theoretical and practical implications of the proposed approach. The main theoretical contribution of our paper is a novel approach for evaluation of performance of SDM activities through concurrent analysis of stakeholder (managers and developers) perceptions and relevant software development tools logs. Its contribution is not primarily in specifying how managers and experts should develop insights and recommendations, but to present how combined sources of information (software development tools logs and perception data) will allow them to develop better recommendations and gain insights that otherwise would not be achievable. The approach combines the state of the art from the field of SDM evaluation (presented in Section 7.2.1) and the field of log data analysis (presented in Section 7.2.2) that were previously not connected. We hope that establishing this bridge enriches both fields and contributes to their further development.

We practically demonstrated how software development tools logs enable management to gain additional insights in the software development process regarding the performance of individual developers. Additionally, we showed how software

development tools logs could be used to monitor the performance of an agile team by tracking the percentage and number of unfinished tasks after each sprint. It is important to know that these findings can only be gained by concurrently analysing subjective qualitative SDM perceptions of developers and management, and quantitative software development tools logs data which importantly complement each other. On one hand, focusing only on log data lacks stakeholder context and is thus often difficult or even impossible to contextually correctly interpret. On the other hand, focusing only on stakeholder SDM perceptions can often lack important details that allow management to gain deeper understanding of specific issues since perceptions are typically collected on a more general level.

Management response that confirmed the validity of the proposed approach was added to the discussion. To ensure construct validity we used multiple sources of information including surveys and interviews with key stakeholders, log data and confirmation of the results with management. Internal validity was confirmed through two activities. The first activity was careful analysis of stakeholder perception and software development tools logs where we did not identify any conflicting information. The second activity was addressing rival explanations with management, which showed consistency between our interpretation and management views. This indicates strong consistency and internal validity of information and explanation. A typical central European software development SME using agile SDM was selected for the case study to ensure representative results and as much external reliability as possible. We believe that the approach can be scaled up and also be used in non-agile settings, however it was only tested in an agile SME setting. To ensure reliability use case study protocol was carefully followed (Yin, 2009).

7.5.3 Threats to validity and other limitations

Four important limitations need to be considered. Firstly, there are limitations of external validity, namely generalization. At this stage we were able to conduct only a single case study in an agile setting. Although, we picked an Austrian SME that is representative for SMEs in the IT sector in central Europe, additional case studies in multiple settings would help confirm the broader usefulness of the approach as well as strengthen its theoretical and practical implications. Secondly, the approach can only be used in enterprises in which software development tool logs are available. Thirdly, the study should be repeated in different cultural settings to see if other cultural contexts would importantly affect the usefulness of the approach. Lastly, due to variability of software development tools logs it is not possible to define a universal step-by-step procedure for analysis of software development tools logs, however this is often the case in the field of process mining.

Similarly, as other approaches in the field of process improvement (Zellner, 2011) our approach does not specify a step-by-step procedure with the aim to replace experts and managers and their ability to interpret data and develop recommendations. On the contrary, it empowers experts and managers with new information based on

concurrent analysis. Moreover, it is impossible to formalize a procedure that would produce appropriate recommendation for each and every company since the context of companies can differ greatly. Thus, experts and managers need to be employed to interpret the information provided by our approach and produce the final insights and recommendations.

7.6 Conclusion

The main contribution of the study is a novel approach for evaluation of software development process that concurrently considers stakeholder SDM perceptions and data from software development tools logs. In a case study, we confirmed the effectiveness of the proposed approach and identified its theoretical and practical contributions to the studied field. We successfully addressed the three research questions posed in the paper and discussed its validity and limitations.

Future work should expand the number of case studies in different cultural settings and test different sources of log data that would enable measuring not only developer performance, but also process and product quality. Furthermore, in the field of SDM perception analysis additional stakeholders should be considered (business partners, customers, etc.). Finally, as the field of process mining is rapidly evolving new methods and tools should be considered. Current automation trends in software development field generate ever increasing number of software development tools logs which will offer future researchers an opportunity to perform more comprehensive long-term analyses.

Chapter 8
Conclusion

This chapter summarizes the main contributions of the dissertation. Section 8.1 lists the results. Section 8.2 discusses in more detail their impact. Section 8.3 outlines future research.

8.1 Summary of Results

This doctoral thesis aims at bridging the gap between automatic analysis of software development data and process mining. It provides analyses techniques that help learning specific characteristics of the software process from its trace data. This thesis makes the following contributions.

- **Conceptualization of project-oriented business processes.** Processes have been seen so far as repeatable endeavors. However, there are processes which are not repeated exactly in the same way twice. This is the case with project-oriented business processes. Such processes, are conducted as projects, in that they are usually planned a priori and run under limited resources.
- **Visualization.** This thesis provides more suitable visualization techniques that help understanding the work process that lies behind software development. More specifically, we presented different possible visualizations to better represent each of the four perspectives of a business process. By mining a GANTT chart, artifact A1 gives insights into the time perspective. By uncovering the hidden co-evolution of software components such a files, artifact A2 shows explicitly the variation of different cases. By mining roles, artifact A3 visualizes de-facto profiles for each developer. By mining activities and their order, artifact A4 gives insights into the control-flow.
- **Extension of the scope of process mining.** Process mining techniques rely on structured data. Typically these data come from Process-Aware Information Systems (PAIS). The artifacts constructed in this thesis handle data which were

not generated by PAIS. Thus, they provide a use case for extending the scope of process mining.

- **Datasets for further research.** During the course of this research a number of datasets have been generated. The artifacts produced by this dissertation provide preprocessed datasets and algorithms that can be used for a variety of analyses on the software process. These datasets are available for further use by researchers who want to replicate these studies or pursue different goals using similar concepts to the ones presented in this thesis.

8.2 Discussion

On a broader context, we discuss the implications of this research for industry and academia.

For industry, these results of this thesis provide the basis for developing a monitoring tool that allows the manager to obtain insights on software development from a process perspective. The four artifacts can be combined together into a dashboard which gives rich information on the various perspectives of the process. Furthermore, existing and new performance indicators can be included to the ones already presented in this thesis, according to the specific needs of the domain. Such dashboard would provide new insights to help managers in making informed decisions based on facts.

For academia, this work informs both areas of business process management and software engineering, in particular their sub-fields of process mining and mining software repositories, respectively. Process mining researchers can use data from a new domain, such as software. Researchers from the mining software repositories area can exploit a new lens on analyzing their data, i.e., a process point of view.

8.3 Future Research and Concluding Remarks

There are several ways in which current research can be improved.

An integrated visualization that brings all four perspectives together would be beneficial to the managers. This would give a clear picture of the impact of each perspective and let the manager decide which of these to prioritize when it comes to improvement. Therefore, the manager has more control over the project and capacity to react to potential issues.

Qualitative analyses of the impact of the proposed artifacts in an industrial context. User studies should be conducted for each of the artifacts in order to investigate their perceived usefulness and ease of use. With this feedback the artifacts can be adapted and improved.

Integrate data from different information systems. Data present in VCS only store part of the overall end-to-end software development methodology. Integrating data from systems, such as project planning systems, emails, documents, etc, would enable

for learning more about how ideas turn into specific process steps. This, in turn, would open up chances for process redesign or innovation.

References

Aaen, I. (2003). Software process improvement: Blueprints versus recipes. *IEEE Softw.*, 20(5):86–93.

Aboelmaged, M. G. (2010a). Predicting e-procurement adoption in a developing country: An empirical integration of technology acceptance model and theory of planned behaviour. *Ind. Manag. Data Syst.*, 110(3):392–414.

Aboelmaged, M. G. (2010b). Predicting e-procurement adoption in a developing country: An empirical integration of technology acceptance model and theory of planned behaviour. *Industrial Management & Data Systems*.

Agrawal, K., Aschauer, M., Thonhofer, T., Bala, S., Rogge-Solti, A., and Tomsich, N. (2016). Resource classification from version control system logs. In Dijkman, R. M., Pires, L. F., and Rinderle-Ma, S., editors, *20th IEEE International Enterprise Distributed Object Computing Workshop, EDOC Workshops 2016, Vienna, Austria, September 5-9, 2016*, pages 1–10. IEEE Computer Society.

Aldave, A., Vara, J. M., Granada, D., and Marcos, E. (2019). Leveraging creativity in requirements elicitation within agile software development: A systematic literature review. *J. Syst. Softw.*, 157.

Alonso, O., Devanbu, P. T., and Gertz, M. (2008). Expertise identification and visualization from CVS. *Proc. 2008 Int. Work. Min. Softw. Repos. - MSR '08*, page 125.

Annosi, M. C., Martini, A., Brunetta, F., and Marchegiani, L. (2020). Learning in an agile setting: A multilevel research study on the evolution of organizational routines. *Journal of Business Research*, 110:554–566.

Atkinson, R. (1999). Project management: cost, time and quality, two best guesses and a phenomenon, its time to accept other success criteria. *International journal of project management*, 17(6):337–342.

Avison, D. and Fitzgerald, G. (2003). *Information systems development: methodologies, techniques and tools*. McGraw-Hill.

Avison, D. and Fitzgerald, G. (2006). *Information systems development: methodologies, techniques and tools (4th ed.)*. McGraw-Hill.

Azemi, E. and Bala, S. (2019). Exploring bpm adoption and strategic alignment of processes at raiffeisen bank kosovo. In *BPM (Industry Forum)*, volume 2428 of *CEUR Workshop Proceedings*, pages 37–48. CEUR-WS.org.

Azemi, E. and Bala, S. (2021). Exploring BPM adoption and strategic alignment of processes at Raiffeisen Bank Kosovo. In vom Brocke, J., Mendling, J., Rosemann, M., and (Eds.), editors, *Bus. Process Manag. Cases*. Springer (to-appear), 2nd edition.

Bacchelli, A. (2016). Structure your unstructured data first! In Menzies, T., Williams, L. A., and Zimmermann, T., editors, *Perspectives on Data Science for Software Engineering*, pages 161–168. Academic Press.

Baier, T., Mendling, J., and Weske, M. (2014). Bridging abstraction layers in process mining. *Inf. Syst.*, 46:123–139.

Bala, S. (2017). Mining projects from structured and unstructured data. In *CEUR Workshop Proc.*, volume 1859.

Bala, S., Cabanillas, C., Haselböck, A., Havur, G., Mendling, J., Polleres, A., Sperl, S., and Steyskal, S. (2017a). A framework for safety-critical process management in engineering projects. In Ceravolo, P. and Rinderle-Ma, S., editors, *Data-Driven Process Discovery and Analysis*, pages 1–27, Cham. Springer International Publishing.

Bala, S., Cabanillas, C., Mendling, J., Rogge-Solti, A., and Polleres, A. (2015). Mining project-oriented business processes. In Motahari-Nezhad, H. R., Recker, J., and Weidlich, M., editors, *Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings*, volume 9253 of *Lecture Notes in Computer Science*, pages 425–440. Springer.

Bala, S., Havur, G., Sperl, S., Steyskal, S., Haselböck, A., Mendling, J., and Polleres, A. (2016). Shapeworks: A bpms extension for complex process management. In *CEUR Workshop Proc.*, volume 1789, pages 50–55.

Bala, S., Kneringer, P., and Mendling, J. (2020). Discovering activities in software development processes. In Asensio, E. S. and Stirna, J., editors, *Proceedings of the Forum at Practice of Enterprise Modeling 2020 co-located with the 13th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modeling (PoEM 2020), Riga, Latvia, November 25-27, 2020*, volume 2793 of *CEUR Workshop Proceedings*, pages 54–63. CEUR-WS.org.

Bala, S. and Mendling, J. (2018). Monitoring the Software Development Process with Process Mining. In Shishkov, B., editor, *Business Modeling and Software Design - 8th International Symposium, BMSD 2018, Vienna, Austria, July 2-4, 2018, Proceedings*, pages 432–442, Cham. Springer International Publishing.

Bala, S., Mendling, J., Schimak, M., and Queteschiner, P. (2018). Case and activity identification for mining process models from middleware. In Buchmann, R. A., Karagiannis, D., and Kirikova, M., editors, *The Practice of Enterprise Modeling - 11th IFIP WG 8.1. Working Conference, PoEM 2018, Vienna, Austria, October 31 - November 2, 2018, Proceedings*, volume 335 of *Lecture Notes in Business Information Processing*, pages 86–102. Springer.

Bala, S., Revoredo, K., de A. R. Gonçalves, J. C., Baião, F. A., Mendling, J., and Santoro, F. M. (2017b). Uncovering the hidden co-evolution in the work history of software projects. In Carmona, J., Engels, G., and Kumar, A., editors, *Business Process Management - 15th International Conference, BPM 2017, Barcelona, Spain, September 10-15, 2017, Proceedings*, volume 10445 of *Lecture Notes in Computer Science*, pages 164–180. Springer.

Balasubramanian, L. and Mnkandla, E. (2016). An evaluation to determine the extent and level of agile software development methodology adoption and implementation in the botswana software development industry. In *2016 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, pages 320–325. IEEE.

Bani-Salameh, H., Ahmad, A., and Aljammal, A. (2016). Software evolution visualization techniques and methods - a systematic review. *2016 7th Int. Conf. Comput. Sci. Inf. Technol.*, 00:1–6.

Bass, J. M. (2016). Artefacts and agile method tailoring in large-scale offshore software development programmes. *Inf. Softw. Technol.*, 75:1–16.

Baumgrass, A., Schefer-Wenzl, S., and Strembeck, M. (2012). Deriving process-related rbac models from process execution histories. In *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, pages 421–426. IEEE.

Bavota, G. (2016). Mining unstructured data in software repositories: Current and future trends. In *Leaders of Tomorrow Symposium: Future of Software Engineering, FOSE@SANER 2016, Osaka, Japan, March 14, 2016*, pages 1–12. IEEE Computer Society.

Begel, A., Khoo, Y. P., and Zimmermann, T. (2010). Codebook: discovering and exploiting relationships in software repositories. In *2010 ACM/IEEE 32nd Int. Conf. Softw. Eng.*, volume 1, pages 125–134.

Beheshti, S., Benatallah, B., and Motahari Nezhad, H. R. (2013). Enabling the analysis of cross-cutting aspects in ad-hoc processes. In *CAiSE*, volume 7908 of *LNCS*, pages 51–67. Springer.

Beheshti, S. M. R., Benatallah, B., Sakr, S., Grigori, D., Motahari-Nezhad, H. R., Barukh, M. C., Gater, A., and Ryu, S. H. (2016). *Process analytics: Concepts and techniques for querying and analyzing process data*. Springer.

Bender, M. a. and Farach-Colton, M. (2000). The LCA problem revisited (Presentation). *Resumos*, 1776 LNCS:88–94.

Berente, N., Seidel, S., and Safadi, H. (2019). Research commentary - data-driven computationally intensive theory development. *Inf. Syst. Res.*, 30(1):50–64.

Bertram, D., Voida, A., Greenberg, S., and Walker, R. J. (2010). Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams. In Inkpen, K., Gutwin, C., and Tang, J. C., editors, *Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, CSCW 2010, Savannah, Georgia, USA, February 6-10, 2010*, pages 291–300. ACM.

Bhattacharya, P., Neamtiu, I., and Faloutsos, M. (2014). Determining Developers' Expertise and Role: A Graph Hierarchy-Based Approach. *2014 IEEE Int. Conf. Softw. Maint. Evol.*, pages 11–20.

Boettiger, C. (2018). Managing larger data on a github repository. *J. Open Source Softw.*, 3(29):971.

Boldi, P., Pietri, A., Vigna, S., and Zacchiroli, S. (2020). Ultra-large-scale repository analysis via graph compression. In Kontogiannis, K., Khomh, F., Chatzigeorgiou, A., Fokaefs, M., and Zhou, M., editors, *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, pages 184–194. IEEE.

Canfora, G. and Cerulo, L. (2006). Supporting change request assignment in open source development. In *SAC*, pages 1767–1772. ACM.

Chan, F. K. Y. and Thong, J. Y. L. (2009). Acceptance of agile methodologies: A critical review and conceptual framework. *Decis. Support Syst.*, 46(4):803–814.

Choetkiertikul, M., Dam, H. K., Tran, T., and Ghose, A. (2017). Predicting the delay of issues with due dates in software projects. *Empir. Softw. Eng.*, 22(3):1223–1263.

Cooper, R. B. and Zmud, R. W. (1990). Information technology implementation research: a technological diffusion approach. *Management science*, 36(2):123–139.

D'Ambros, M. and Lanza, M. (2008). A Flexible Framework to Support Collaborative Software Evolution Analysis. In *Softw. Maint. Reengineering*, pages 3–12.

Dehghan, A., Neal, A., Blincoe, K., Linaker, J., and Damian, D. (2017). Predicting likelihood of requirement implementation within the planned iteration: an empirical study at ibm. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 124–134. IEEE.

Destefanis, G., Ortu, M., Counsell, S., Swift, S., Marchesi, M., and Tonelli, R. (2016). Software development: do good manners matter? *PeerJ Comput. Sci.*, 2:e73.

Dingsøyr, T., Nerur, S. P., Balijepally, V., and Moe, N. B. (2012). A decade of agile methodologies: Towards explaining agile software development. *J. Syst. Softw.*, 85(6):1213–1221.

Dumas, M., Rosa, M. L., Mendling, J., and Reijers, H. A. (2018). *Fundamentals of Business Process Management, Second Edition*. Springer.

Dyer, R., Nguyen, H. A., Rajan, H., and Nguyen, T. N. (2015). Boa: Ultra-large-scale software repository and source-code mining. *ACM Trans. Softw. Eng. Methodol.*, 25(1):7:1–7:34.

Estublier, J. (2000). Software configuration management: a roadmap. In Finkelstein, A., editor, *22nd International Conference on on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4-11, 2000*, pages 279–289. ACM.

Feiner, J. and Andrews, K. (2018). Repovis: Visual overviews and full-text search in software repositories. In *2018 IEEE Working Conference on Software Visualization, VISSOFT 2018, Madrid, Spain, September 24-25, 2018*, pages 1–11. IEEE.

Feldt, R., Staron, M., Hult, E., and Liljegren, T. (2013). Supporting software decision meetings: Heatmaps for visualising test and code measurements. In *Softw. Eng. Adv. Appl. (SEAA), 2013 39th EUROMICRO Conf.*, pages 62–69. IEEE.

Fitzgerald, B. (1996). Formalized systems development methodologies: a critical perspective. *Inf. Syst. J.*, 6(1):3–24.

Fitzgerald, B. (1998). An empirical investigation into the adoption of systems development methodologies. *Inf. Manag.*, 34(6):317–328.

Fontdevila, D., Genero, M., Oliveros, A., and Paez, N. (2019). Evaluating the utility of the usability model for software development process and practice. In Franch, X., Männistö, T., and Martínez-Fernández, S., editors, *Product-Focused Software Process Improvement - 20th International Conference, PROFES 2019, Barcelona, Spain, November 27-29, 2019, Proceedings*, volume 11915 of *Lecture Notes in Computer Science*, pages 741–757. Springer.

Frank, M., Buhman, J. M., and Basin, D. (2013). Role mining with probabilistic models. *ACM Trans. Inf. Syst. Secur.*, 15(4):15.

Füller, J., Hutter, K., Hautz, J., and Matzler, K. (2014). User Roles and Contributions in Innovation-Contest Communities. *J. Manag. Inf. Syst.*, 31(1):273–308.

Gallivan, M. J. (2003). The influence of software developers' creative style on their attitudes to and assimilation of a software process innovation. *Inf. Manag.*, 40(5):443–465.

Germán, D. M. and Hindle, A. (2006). Visualizing the evolution of software using softchange. *Int. J. Softw. Eng. Knowl. Eng.*, 16(1):5–22.

Gill, A. Q., Henderson-Sellers, B., and Niazi, M. (2018). Scaling for agility: A reference model for hybrid traditional-agile software development methodologies. *Information Systems Frontiers*, 20(2):315–341.

Gini, C. (1921). Measurement of inequality of incomes. *The Economic Journal*, 31(121):124–126.

Gîrba, T., Ducasse, S., and Lanza, M. (2004). Yesterday"s weather: Guiding early reverse engineering efforts by summarizing the evolution of changes. In *20th International Conference on Software Maintenance (ICSM 2004), 11-17 September 2004, Chicago, IL, USA*, pages 40–49. IEEE Computer Society.

Gonçalves, J. C. D. A. R., Santoro, F. M., and Baião, F. A. (2011). Let Me Tell You a Story - On How to Build Process Models. *J. Univers. Comput. Sci.*, 17(2):276–295.

Gousios, G., Kalliamvakou, E., and Spinellis, D. (2008). Measuring developer contribution from software repository data. In *Proc. 2008 Int. Work. Conf. Min. Softw. Repos.*, pages 129–132. ACM.

Green, G. C., Hevner, A. R., and Collins, R. W. (2005). The impacts of quality and productivity perceptions on the use of software process improvement innovations. *Inf. Softw. Technol.*, 47(8):543–553.

Greevy, O., Ducasse, S., and G\^\irba, T. (2006). Analyzing software evolution through feature views. *J. Softw. Maint. Res. Pr.*, 18(6):425–456.

Gregor, S. (2006). The nature of theory in information systems. *MIS Q.*, pages 611–642.

Gregor, S., Chandra Kruse, L., and Seidel, S. (2020). Research perspectives: The anatomy of a design principle. *J. Assoc. Inf. Syst.*, 21(6):1622–1652.

Grisold, T., Wurm, B., Mendling, J., and vom Brocke, J. (2020). Using process mining to support theorizing about change in organizations. In *53rd Hawaii International Conference on System Sciences, HICSS 2020, Maui, Hawaii, USA, January 7-10, 2020*, pages 1–10. ScholarSpace.

Gross, S., Malinova, M., and Mendling, J. (2019). Navigating through the maze of business process change methods. In Bui, T., editor, *52nd Hawaii International*

Conference on System Sciences, HICSS 2019, Grand Wailea, Maui, Hawaii, USA, January 8-11, 2019, pages 1–10. ScholarSpace.

Gubichev, A., Bedathur, S., Seufert, S., and Weikum, G. (2010). Fast and accurate estimation of shortest paths in large graphs. *Proc. 19th ACM Int. Conf. Inf. Knowl. Manag. - CIKM '10*, page 499.

Hamdy, A. and Ezzat, G. (2020). Deep mining of open source software bug repositories. *International Journal of Computers and Applications*, pages 1–9.

Henderson, F. (2017). Software engineering at Google. *arXiv Prepr. arXiv1702.01715*.

Henderson-Sellers, B., Ralyté, J., Ågerfalk, P. J., and Rossi, M. (2014). *Situational Method Engineering*. Springer.

Hevner, A. R., March, S. T., Park, J., and Ram, S. (2004). Design Science in Information Systems Research. *MIS Q.*, 28(1):75–105.

Hoda, R., Noble, J., and Marshall, S. (2011). The impact of inadequate customer collaboration on self-organizing agile teams. *Inf. Softw. Technol.*, 53(5):521–534.

Hoda, R., Noble, J., and Marshall, S. (2013). Self-organizing roles on agile software development teams. *Softw. Eng. IEEE Trans.*, 39(3):422–444.

Hou, Q., Ma, Y., Chen, J., and Xu, Y. (2014). An empirical study on inter-commit times in SVN. In Reformat, M., editor, *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013*, pages 132–137. Knowledge Systems Institute Graduate School.

Hovelja, T., Vasilecas, O., and Vavpotič, D. (2015). Exploring the influences of the use of elements comprising information system development methodologies on strategic business goals. *Technological and economic development of economy*, 21(6):885–898.

Hu, G., Peng, M., Zhang, Y., Xie, Q., Gao, W., and Yuan, M. (2020). Unsupervised software repositories mining and its application to code search. *Softw. Pract. Exp.*, 50(3):299–322.

Huisman, M. and Iivari, J. (2002). The Organisational Deployment of Systems Development Methodologies. In Kirikova, M., Grundspenkis, J., Wojtkowski, W., Wojtkowski, W. G., Wrycza, S., and Zupančič, J., editors, *Information Systems Development: Advances in Methodologies, Components, and Management*, pages 87–100. Springer US, Boston, MA.

Huisman, M. and Iivari, J. (2006). Deployment of systems development methodologies: Perceptual congruence between IS managers and systems developers. *Inf. Manag.*, 43(1):29–49.

Iivari, J. and Huisman, M. (2001). The relationship between organisational culture and the deployment of systems development methodologies. In Dittrich, K. R., Geppert, A., and Norrie, M. C., editors, *Advanced Information Systems Engineering, 13th International Conference, CAiSE 2001, Interlaken, Switzerland, June 4-8, 2001, Proceedings*, volume 2068 of *Lecture Notes in Computer Science*, pages 234–250. Springer.

Jooken, L., Creemers, M., and Jans, M. (2019). Extracting a collaboration model from VCS logs based on process mining techniques. In *Business Process Management Workshops*, volume 362 of *LNBIP*, pages 212–223. Springer.

Joonbakhsh, A. and Sami, A. (2018). Mining and extraction of personal software process measures through IDE interaction logs. In *MSR*, pages 78–81. ACM.

Jørgensen, M. (2019). Relationships between project size, agile practices, and successful software development: Results and analysis. *IEEE Softw.*, 36(2):39–43.

Kagdi, H., Yusuf, S., and Maletic, J. I. (2006). Mining Sequences of Changed-files from Version Histories. In *Work. Min. Softw. Repos. (MSR '06)*, pages 47–53. ACM.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., Germán, D. M., and Damian, D. E. (2016). An in-depth study of the promises and perils of mining GitHub. *Empir. Softw. Eng.*, 21(5):2035–2071.

Karlsson, F. J. and Ågerfalk, P. J. (2009). Exploring agile values in method configuration. *Eur. J. Inf. Syst.*, 18(4):300–316.

Kindler, E., Rubin, V., and Schäfer, W. (2006a). Incremental Workflow Mining Based on Document Versioning Information. In Li, M., Boehm, B., and Osterweil, L., editors, *Unifying Softw. Process Spectr.*, volume 3840 of *Lecture Notes in Computer Science*, pages 287–301. Springer Berlin Heidelberg.

Kindler, E., Rubin, V. A., and Schäfer, W. (2006b). Activity mining for discovering software process models. In *Software Engineering*, volume P-79 of *LNI*, pages 175–180. GI.

Kneuper, R. (2008). *CMMI: improving software and systems development processes using capability maturity model integration*. Rocky Nook.

Kouters, E., Vasilescu, B., Serebrenik, A., and van den Brand, M. G. J. (2012). Who's who in gnome: Using LSA to merge software repository identities. In *28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012*, pages 592–595. IEEE Computer Society.

Laporte, C. Y., O'Connor, R. V., and Paucar, L. H. G. (2015). The implementation of ISO/IEC 29110 software engineering standards and guides in very small entities. In Maciaszek, L. A. and Filipe, J., editors, *Evaluation of Novel Approaches to Software Engineering - 10th International Conference, ENASE 2015, Barcelona, Spain, April 29-30, 2015, Revised Selected Papers*, volume 599 of *Communications in Computer and Information Science*, pages 162–179. Springer.

Leemans, S. J. J., Fahland, D., and van der Aalst, W. M. P. (2014). Process and deviation exploration with inductive visual miner. In Limonad, L. and Weber, B., editors, *Proceedings of the BPM Demo Sessions 2014 Co-located with the 12th International Conference on Business Process Management (BPM 2014), Eindhoven, The Netherlands, September 10, 2014*, volume 1295 of *CEUR Workshop Proceedings*, page 46. CEUR-WS.org.

Licorish, S. A. and MacDonell, S. G. (2014). Understanding the attitudes, knowledge sharing behaviors and task performance of core developers: A longitudinal study. *Inf. Softw. Technol.*, 56(12):1578–1596.

Lindberg, A., Berente, N., Gaskin, J. E., and Lyytinen, K. (2016). Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project. *Inf. Syst. Res.*, 27(4):751–772.

Lohmann, N., Verbeek, E., and Dijkman, R. M. (2009). Petri net transformations for business processes - A survey. *Trans. Petri Nets Other Model. Concurr.*, 2:46–63.

Loon, H. v. (2007). *Process Assessment and ISO/IEC 15504: A Reference Book*. Springer-Verlag, Berlin, Heidelberg.

Lu, H., Hong, Y., Yang, Y., Duan, L., and Badar, N. (2015). Towards user-oriented RBAC model. *J. Comput. Secur.*, 23(1):107–129.

Maalej, W. and Happel, H.-J. (2010). Can Development Work Describe Itself? *7th IEEE Work. Conf. Min. Softw. Repos. (MSR 2010)*, pages 191–200.

Majchrzak, A. and Malhotra, A. (2016). Effect of Knowledge-Sharing Trajectories on Innovative Outcomes in Temporary Online Crowds. *Inf. Syst. Res.*, 27(4):685–703.

Malinova, M., Gross, S., and Mendling, J. (2022). A study into the contingencies of process improvement methods. *Information Systems*, 104.

Mäntylä, M., Adams, B., Destefanis, G., Graziotin, D., and Ortu, M. (2016). Mining valence, arousal, and dominance: possibilities for detecting burnout and productivity? In Kim, M., Robbes, R., and Bird, C., editors, *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, pages 247–258. ACM.

Marques, R., da Silva, M. M., and Ferreira, D. R. (2018). Assessing agile software development processes with process mining: A case study. In Proper, H. A., Strecker, S., and Huemer, C., editors, *20th IEEE Conference on Business Informatics, CBI 2018, Vienna, Austria, July 11-14, 2018, Volume 1 - Research Papers*, pages 109–118. IEEE Computer Society.

McNair, A., Germán, D. M., and Weber-Jahnke, J. H. (2007). Visualizing software architecture evolution using change-sets. In *14th Working Conference on Reverse Engineering (WCRE 2007), 28-31 October 2007, Vancouver, BC, Canada*, pages 130–139. IEEE Computer Society.

Mitra, B., Sural, S., Vaidya, J., and Atluri, V. (2016). A Survey of Role Mining. *ACM Comput. Surv.*, 48(4):1–37.

Mittal, M. and Sureka, A. (2014). Process Mining Software Repositories from Student Projects in an Undergraduate Software Engineering Course. In *Companion Proc. 36th Int. Conf. Softw. Eng.*, ICSE Companion 2014, pages 344–353. ACM.

Ogawa, M. and Ma, K.-L. (2010). Software evolution storylines. In *Proc. 5th Int. Symp. Softw. Vis.*, pages 35–42. ACM.

Oliva, G. A., Santana, F. W., Gerosa, M. A., and de Souza, C. R. B. (2011). Towards a classification of logical dependencies origins: a case study. In *EVOL/IWPSE*, pages 31–40. ACM.

OMG (2011). BPMN 2.0. Recommendation, OMG.

Ortu, M., Destefanis, G., Kassab, M., and Marchesi, M. (2015). Measuring and understanding the effectiveness of JIRA developers communities. In Tonelli, R., Tempero, E. D., Counsell, S., and Visaggio, C. A., editors, *6th IEEE/ACM International Workshop on Emerging Trends in Software Metrics, WETSoM 2015, Florence, Italy, May 17, 2015*, pages 3–10. IEEE Copmuter Society.

Ortu, M., Murgia, A., Destefanis, G., Tourani, P., Tonelli, R., Marchesi, M., and Adams, B. (2016). The emotional side of software developers in JIRA. In Kim, M., Robbes, R., and Bird, C., editors, *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, pages 480–483. ACM.

Peffers, K., Tuunanen, T., Rothenberger, M., and Chatterjee, S. (2008). A Design Science Research Methodology for Information Systems Research. *J. Manag. Inf. Syst.*, 24(January):45–77.

Pilato, C. M., Collins-Sussman, B., and Fitzpatrick, B. W. (2008). *Version control with subversion*. "O'Reilly Media, Inc.".

Poncin, W., Serebrenik, A., and van den Brand, M. (2011a). Process Mining Software Repositories. *2011 15th Eur. Conf. Softw. Maint. Reengineering*, pages 5–14.

Poncin, W., Serebrenik, A., and van den Brand, M. (2011b). Process Mining Software Repositories. *2011 15th Eur. Conf. Softw. Maint. Reengineering*, pages 5–14.

Poncin, W., Serebrenik, A., and van den Brand, M. (2011c). Process mining software repositories. In *CSMR*, pages 5–14. IEEE Computer Society.

Ralyté, J., Deneckère, R., and Rolland, C. (2003). Towards a generic model for situational method engineering. In Eder, J. and Missikoff, M., editors, *Advanced Information Systems Engineering, 15th International Conference, CAiSE 2003, Klagenfurt, Austria, June 16-18, 2003, Proceedings*, volume 2681 of *Lecture Notes in Computer Science*, pages 95–110. Springer.

Revoredo, K., Djurica, D., and Mendling, J. (2021). A study into the practice of reporting software engineering experiments. *Empir. Softw. Eng.*, 26(5):113.

Riemenschneider, C. K., Hardgrave, B. C., and Davis, F. D. (2002). Explaining software developer acceptance of methodologies: A comparison of five theoretical models. *IEEE Trans. Software Eng.*, 28(12):1135–1145.

Robles, G., González-Barahona, J. M., Cervigón, C., Capiluppi, A., and Izquierdo-Cortázar, D. (2014). Estimating Development Effort in Free/Open Source Software Projects by Mining Software Repositories: A Case Study of OpenStack. In *Proc. 11th Work. Conf. Min. Softw. Repos.*, MSR 2014, pages 222–231. ACM.

Rodríguez, A., Tanaka, F., and Kamei, Y. (2018). Empirical study on the relationship between developer's working habits and efficiency. In *MSR*, pages 74–77. ACM.

Rogers, E. M. (2003). *Diffusion of innovations (5. ed.)*. Free Press.

Rubin, V., Günther, C. W., Van Der Aalst, W. M. P., Kindler, E., Van Dongen, B. F., and Schäfer, W. (2007). Process mining framework for software processes. In *Softw. Process Dyn. Agil.*, volume 4470, pages 169–181. Springer.

Rubin, V., Lomazova, I., and van der Aalst, W. M. P. (2014). Agile development with software process mining. In *Proc. 2014 Int. Conf. Softw. Syst. Process*, pages 70–74. ACM.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empir. Softw. Eng.*, 14(2):131–164.

Scheer, A., Thomas, O., and Adam, O. (2005). Process modeling using event-driven process chains. In Dumas, M., van der Aalst, W. M. P., and ter Hofstede, A. H. M., editors, *Process-Aware Information Systems: Bridging People and Software Through Process Technology*, pages 119–145. Wiley.

Schönig, S., Cabanillas, C., Jablonski, S., and Mendling, J. (2015). Mining the Organisational Perspective in Agile Business Processes. In *BPMDS*, pages 37–52.

Scott, E., Milani, F., Kilu, E., and Pfahl, D. (2021). Enhancing agile software development in the banking sector - A comprehensive case study at LHV. *J. Softw. Evol. Process.*, 33(7).

Song, M. and van der Aalst, W. M. P. (2007). Supporting process mining by showing events at a glance. In *Proc. 17th Annu. Work. Inf. Technol. Syst.*, pages 139–145.

Song, M. and van der Aalst, W. M. P. (2008). Towards comprehensive support for organizational mining. *Decis. Support Syst.*, 46(1):300–317.

Takeuchi, H. and Nonaka, I. (1986). The new new product development game. *Harvard business review*, 64(1):137–146.

Torvalds, L. and Hamano, J. (2010). Git: Fast version control system. *URL http//git-scm. com*.

Tsoury, A., Soffer, P., and Reinhartz-Berger, I. (2018). A conceptual framework for supporting deep exploration of business process behavior. In *ER*, volume 11157 of *LNCS*, pages 58–71. Springer.

Tuape, M., Hasheela-Mufeti, V. T., Kayanda, A., Porras, J., and Kasurinen, J. (2021). Software engineering in small software companies: Consolidating and integrating empirical literature into a process tool adoption framework. *IEEE Access*, 9:130366–130388.

van der Aalst, W. M. P. (1999). Formalization and verification of event-driven process chains. *Inf. Softw. Technol.*, 41(10):639–650.

van der Aalst, W. M. P. (2016). *Process Mining - Data Science in Action, Second Edition*. Springer.

van der Aalst, W. M. P. and ter Hofstede, A. H. M. (2005). YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275.

van der Aalst, W. M. P., Weijters, T., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142.

van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A., and Van Der Aalst, W. M. P. (2005). The ProM framework: A new era in process mining tool support. In *Appl. Theory Petri Nets 2005*, pages 444–454. Springer.

van Dongen, B. F. and der Aalst, W. M. P. (2005). A Meta Model for Process Mining Data. *EMOI-INTEROP*, 160:30.

van Eck, M. L., Lu, X., Leemans, S. J. J., and van der Aalst, W. M. P. (2015). PM ^2 : A process mining project methodology. In Zdravkovic, J., Kirikova, M., and Johannesson, P., editors, *Advanced Information Systems Engineering - 27th International Conference, CAiSE 2015, Stockholm, Sweden, June 8-12, 2015, Proceedings*, volume 9097 of *Lecture Notes in Computer Science*, pages 297–313. Springer.

Vasilescu, B., Serebrenik, A., Goeminne, M., and Mens, T. (2014). On the variation and specialisation of workload - A case study of the gnome ecosystem community. *Empirical Software Engineering*, 19(4):955–1008.

Vavpotic, D. and Bajec, M. (2009). An approach for concurrent evaluation of technical and social aspects of software development methodologies. *Inf. Softw. Technol.*, 51(2):528–545.

Vavpotic, D. and Hovelja, T. (2012). Improving the evaluation of software development methodology adoption and its impact on enterprise performance. *Comput. Sci. Inf. Syst.*, 9(1):165–187.

Vavpotic, D., Robnik-Sikonja, M., and Hovelja, T. (2020). Exploring the relations between net benefits of IT projects and cios' perception of quality of software development disciplines. *Bus. Inf. Syst. Eng.*, 62(4):347–360.

Venable, J., Pries-Heje, J., and Baskerville, R. (2016). FEDS: a Framework for Evaluation in Design Science Research. *Eur. J. Inf. Syst.*, 25(1):77–89.

Venkatesh, V. and Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management science*, 46(2):186–204.

Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., and van der Aalst, W. M. P. (2010). Xes, xesame, and prom 6. In Soffer, P. and Proper, E., editors, *Information Systems Evolution - CAiSE Forum 2010, Hammamet, Tunisia, June 7-9, 2010, Selected Extended Papers*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer.

Vidgof, M., Djurica, D., Bala, S., and Mendling, J. (2020). Cherry-picking from spaghetti: Multi-range filtering of event logs. In *BPMDS/EMMSAD@CAiSE*, volume 387 of *Lecture Notes in Business Information Processing*, pages 135–149. Springer.

Vidgof, M., Djurica, D., Bala, S., and Mendling, J. (2021). Interactive log-delta analysis using multi-range filtering. *Software and Systems Modeling*.

Viner, D., Stierle, M., and Matzner, M. (2021). A process mining software comparison.

Voinea, L. and Telea, A. (2006a). An Open Framework for CVS Repository Querying, Analysis and Visualization. In *Int. Work. Min. Softw. Repos. (MSR '06)*, pages 33–39. ACM.

Voinea, L. and Telea, A. (2006b). Multiscale and Multivariate Visualizations of Software Evolution. In *Symp. Softw. Vis.*, pages 115–124. ACM.

Voinea, L. and Telea, A. C. (2007). Visual data mining and analysis of software repositories. *Comput. Graph.*, 31(3):410–428.

Waibel, P., Novak, C., Bala, S., Revoredo, K., and Mendling, J. (2021). Analysis of Business Process Batching Using Causal Event Models. In Leemans, S. J. J. and Leopold, H., editors, *ICPM 2020 Work.*, pages 1–13. Springer Nature Switzerland AG.

Wang, W., Li, X., and Hsieh, J. J. P. (2013). The contingent effect of personal IT innovativeness and IT self-efficacy on innovative use of complex IT. *Behav. Inf. Technol.*, 32(11):1105–1124.

Weicheng, Y., Shen, B., and Xu, B. (2013). Mining GitHub: Why Commit Stops - Exploring the Relationship between Developer's Commit Pattern and File Version Evolution. In Muenchaisri, P. and Rothermel, G., editors, *APSEC 2013, Ratchathewi,Thailand, December 2-5, 2013 - Vol. 2*, pages 165–169. IEEE Computer Society.

Weske, M. (2019). *Business Process Management - Concepts, Languages, Architectures, Third Edition*. Springer.

Wettel, R. and Lanza, M. (2007). Visualizing software systems as cities. In Maletic, J. I., Telea, A., and Marcus, A., editors, *Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, VISSOFT 2007, Banff, Alberta, Canada, June 25-26, 2007*, pages 92–99. IEEE Computer Society.

Wilson, J. M. (2003). Gantt charts: A centenary appreciation. *Eur. J. Oper. Res.*, 149(2):430–437.

Wolinski, B. and Bala, S. (2018). Comprehensive business process management at siemens: Implementing business process excellence. In vom Brocke, J. and Mendling, J., editors, *Business Process Management Cases, Digital Innovation and Business Transformation in Practice*, Management for Professionals, pages 111–124. Springer.

Wu, J., Spitzer, C. W., Hassan, A. E., and Holt, R. C. (2004). Evolution Spectrographs: visualizing punctuated change in software evolution. In *Work. Princ. Softw. Evol.*, pages 57–66.

Xes-standard.org (2015). openxes:start | XES.

Yin, R. K. (2009). Case study research: Design and methods (applied social research methods). *London and Singapore: Sage*, 23:24.

Ying, A. T. T., Murphy, G. C., Ng, R., and Chu-Carroll, M. C. (2004). Predicting Source Code Changes by Mining Change History. *IEEE Trans. Softw. Eng.*, 30(9):574–586.

Ying, A. T. T. and Robillard, M. P. (2014). Developer profiles for recommendation systems. *Recomm. Syst. Softw. Eng.*, pages 199–222.

Yu, L. and Ramaswamy, S. (2007). Mining CVS repositories to understand open-source project developer roles. In *Proc. - ICSE 2007 Work. Fourth Int. Work. Min. Softw. Repos. MSR 2007*, pages 7–10.

Zaidman, A., Rompaey, B. V., Demeyer, S., and van Deursen, A. (2008). Mining software repositories to study co-evolution of production & test code. In *ICST*, pages 220–229. IEEE Computer Society.

Zdravkovic, J., Svee, E., and Giannoulis, C. (2015). Capturing consumer preferences as requirements for software product lines. *Requir. Eng.*, 20(1):71–90.

Zellner, G. (2011). A structured evaluation of business process improvement approaches. *Bus. Process. Manag. J.*, 17:203–237.

Zimmermann, T. and Nagappan, N. (2008). Predicting defects using network analysis on dependency graphs. *Proc. 13th Int. Conf. Softw. Eng. - ICSE '08*, page 531.

Zimmermann, T. and Weißgerber, P. (2004). Preprocessing CVS data for fine-grained analysis. In *MSR*, pages 2–6.

Zimmermann, T., Weißgerber, P., Diehl, S., and Zeller, A. (2005). Mining version histories to guide software changes. *IEEE Trans. Software Eng.*, 31(6):429–445.