INSTITUTO
UNIVERSITÁRIO
DE LISBOA

Software Development Process Mining: Discovery, Conformance Checking and Enhancement

João Carlos Palmela Pinheiro Caldeira

PhD in Information Science and Technology

Supervisors:
Doctor Fernando Brito e Abreu, Associate Professor,
Iscte – Instituto Universitário de Lisboa

Doctor Jorge Cardoso, Associate Professor,
Faculdade de Ciências e Tecnologia da Universidade de Coimbra

January, 2021

TECNOLOGIAS
E ARQUITETURA

Department of Information Science and Technology

**Software Development Process Mining:
Discovery, Conformance Checking and Enhancement**

João Carlos Palmela Pinheiro Caldeira

PhD in Information Science and Technology

Jury:

Doctor Ricardo Ribeiro, Associate Professor, Iscte – Instituto Universitário de Lisboa (President)

Doctor Pasquale Ardimento, Assistant Professor, Universidade Degli Studi Di Bari Aldo Moro

Doctor Toacy Oliveira, Associate Professor, Universidade Federal do Rio de Janeiro

Doctor André Leal Santos, Assistant Professor, Iscte – Instituto Universitário de Lisboa

Doctor Fernando Brito e Abreu, Associate Professor, Iscte – Instituto Universitário de Lisboa

January, 2021

Software Development Process Mining:
Discovery, Conformance Checking and Enhancement

Copyright © 2021, João Carlos Palmela Pinheiro Caldeira, School of Technology and Architecture, University Institute of Lisbon.
The School of Technology and Architecture and the University Institute of Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

[This page has been intentionally left blank]

To my children, Francisco and Martim.

[This page has been intentionally left blank]

Acknowledgements

"I have no special talent. I am only passionately curious."
—Albert Einstein(1879-1955)[1]

It was a long journey, longer than anything I could have initially aspired or even predicted.

There were periods of doubt, disbelief, turmoil and procrastination. However, there were also those magic moments where the vision, the execution and the delivery were so aligned that the feeling of achievement was immense.

Concluding a task like this, is to achieve a lifetime dream. Nonetheless, only in the last few years it gently became a serious goal, recently turned into a major occupancy and finally emerges as an important personal achievement. It is indeed a time of genuine joy and indescribable emotions.

On reaching this moment, I have a strong impression that this milestone was only possible, not because I may possess any special talent, but yet, as a result of a persistence posture, no fear of failing, curiosity when facing challenges and a resilient attitude over long periods.

Now, on submitting this dissertation for evaluation under the rules of the *"judgement day"*, I cannot do it without referring the individuals and organizations that, without them, this work would not have been possible, or its quality would have been substantially reduced.

However, any remaining mistakes, failures, inaccuracies, incoherence and eventual shortness of talent in presenting the contents, are of my own and total liability.

I would like to thank and express my sincere appreciation and recognition to my supervisor, Professor Fernando Brito e Abreu, for his permanent encouragement and endless support in the preparation, execution and conclusion of this dissertation. During this work, he was an inexhaustible source of valuable knowledge and friendship. Highlighting his merits is not only appropriate and just, but yet an expression of my gratitude for his commitment and professionalism.

I would like to thank to my co-supervisor, Professor Jorge Cardoso, for the encouragement and fundamental guidance provided me in many aspects involving process mining and for his always on-time collaboration in all my articles, as well as many other contents included in this dissertation.

[1] German-born theoretical physicist who developed the theory of relativity, one of the two pillars of modern physics alongside quantum mechanics, and Nobel Prize in Physics in 1921.

I would like to thank to my follow-up group member, Professor André Santos, for giving me several hints in order to build a coherent alignment for my articles and in this dissertation.

Even facing the risk of forgetting someone, I would also like to thank to all of those that more actively, silently or anonymously have contributed to this dissertation.

By no order of importance, they were:

- Professor Vítor Basto-Fernandes and José Reis for their involvement in the experiments I conducted. Without them, the technical conditions and all the logistics to carry the data collection needed for this research wouldn't have been possible. I have an endless debt towards them.
- My other colleagues at ISTAR-Iscte, mainly Américo Rio, for the valuable insights, discussions and knowledge sharing we had in the last five years.
- Professor Ricardo Ribeiro for his valuable contributions in one of my articles, and for guiding me in the right direction regarding n-gram language models and text mining methods in general.
- Professor Toacy Oliveira for accurate and always on-time comments made in order to improve the main contents I have published in different articles.
- Professor Cláudia Werner for participating in one of my published works, and for the meticulous suggestions given regarding document structure and contents quality improvement.
- Javier Munoz at Fusion Global Business Solutions, for permanent encouragement during the last two years, and for giving me the possibility to run in parallel this work and my professional duties.
- Pablo Ferrari at ProcessLabs.ai for the ideas we shared, for making me believe I was on the right track, constant encouragement and his touching comments provided in this dissertation regarding my research.
- A warm recognition to all the participants in the experiments I carried. In reality, without them, nothing in this work would have been made possible.
- My parents and closest family members, for being a catalyst to my determination.
- Finally, to my wife and sons, for their presence, love, and patience during the endless moments I was absent on trying to bring this work to reality and with the level of quality it requires. I acknowledge the *"price paid"* was high enough on multiple circumstances.

During this journey, and particularly in the final year, I went through a set of new and enriching experiences, either in the academic realm, as well as in my professional life, where difficulties and unpredictability were always present.

Looking back, even if the journey was hard and long, I must say it was a period I will remember forever and that I am grateful for everything I learnt and to the ones I acquired such knowledge from.

I hope others may give extra attention to the research path initiated in this work, considering I have produced a honest work, I tried to the best of my abilities to be accurate, sometimes even perfectionist, and always trying to go beyond my limitations. Even if the obstacles are significant and relevant outcomes may look distant, one should never be afraid of challenges or continuous change, as both cases, at least to me, make me somehow believe there is always a bright future ahead.

Lisboa, January of 2021

João Carlos Palmela Pinheiro Caldeira

[This page has been intentionally left blank]

Abstract

Context. Modern software projects require the proper allocation of human, technical and financial resources. Very often, project managers make decisions supported only by their personal experience, intuition or simply by mirroring activities performed by others in similar contexts. Most attempts to avoid such practices use models based on lines of code, cyclomatic complexity or effort estimators, thus commonly supported by software repositories which are known to contain several flaws.

Objective. Demonstrate the usefulness of process data and mining methods to enhance the software development practices, by assessing efficiency and unveil unknown process insights, thus contributing to the creation of novel models within the software development analytics realm.

Method. We mined the development process fragments of multiple developers in three different scenarios by collecting Integrated Development Environment (IDE) events during their development sessions. Furthermore, we used process and text mining to discovery developers' workflows and their fingerprints, respectively.

Results. We discovered and modeled with good quality developers' processes during programming sessions based on events extracted from their IDEs. We unveiled insights from coding practices in distinct refactoring tasks, built accurate software complexity forecast models based only on process metrics and setup a method for characterizing coherently developers' behaviors. The latter may ultimately lead to the creation of a catalog of software development process smells.

Conclusions. Our approach is agnostic to programming languages, geographic location or development practices, making it suitable for challenging contexts such as in modern global software development projects using either traditional IDEs or sophisticated low/no code platforms.

Keywords: Software Development Process Mining, Process Mining, Empirical Software Engineering, Software Development Analytics, Programming Practices, Efficiency Assessment

[This page has been intentionally left blank]

Resumo

Contexto. Projetos de software modernos requerem a correta alocação de recursos humanos, técnicos e financeiros. Frequentemente, os gestores de projeto tomam decisões suportadas apenas na sua própria experiência, intuição ou simplesmente espelhando atividades executadas por terceiros em contextos similares. As tentativas para evitar tais práticas baseiam-se em modelos que usam linhas de código, a complexidade ciclomática ou em estimativas de esforço, sendo estes tradicionalmente suportados por repositórios de software conhecidos por conterem várias limitações.

Objetivo. Demonstrar a utilidade dos dados de processo e respetivos métodos de análise na melhoria das práticas de desenvolvimento de software, colocando o foco na análise da eficiência e revelando aspetos dos processos até então desconhecidos, contribuindo para a criação de novos modelos no contexto de análises avançadas para o desenvolvimento de software.

Método. Explorámos os fragmentos de processo de vários programadores em três cenários diferentes, recolhendo eventos durante as suas sessões de desenvolvimento no IDE. Adicionalmente, usámos métodos de descoberta e análise de processos e texto no sentido de modelar o fluxo de trabalho dos programadores e as suas características individuais, respetivamente.

Resultados. Descobrimos e modelámos com boa qualidade os processos dos programadores durante as suas sessões de trabalho, usando eventos provenientes dos seus IDEs. Revelámos factos desconhecidos sobre práticas de refabricação, construímos modelos de previsão da complexidade ciclomática usando apenas métricas de processo e criámos um método para caracterizar coerentemente os comportamentos dos programadores. Este último, pode levar à criação de um catálogo de boas/más práticas no processo de desenvolvimento de software.

Conclusões. A nossa abordagem é agnóstica em termos de linguagens de programação, localização geográfica ou prática de desenvolvimento, tornando-a aplicável em contextos complexos tal como em projetos modernos de desenvolvimento global que utilizam tanto os IDEs tradicionais como as atuais e sofisticadas plataformas *"low/no code"*.

Palavras-chave: Análise de Processo de Desenvolvimento de Software, Análise de Processos, Engenharia de Software Experimental, Análise de Desenvolvimento de Software, Práticas de Programação, Avaliação de Eficiência

[This page has been intentionally left blank]

Contents

List of Figures

[This page has been intentionally left blank]

List of Tables

[This page has been intentionally left blank]

Listings

[This page has been intentionally left blank]

Acronyms

ANOVA	Analysis of Variance.
API	Application Program Interface.
BPMN	Business Process Model and Notation.
BTT	Bug/Issue Tracking Tools.
CASP	Critical Appraisal Skills Programme.
CD	Continuous Deployment.
CI	Continuous Integration.
CI/CD	Continuous Integration and Continuous Deployment.
CMS	Configuration Management System.
CSV	Comma Separated Values.
CVR	Cockpit Voice Recorder.
CVS	Concurrent Versions System.
DFG	Directly Follows Graph.
DSS	Decision Support Systems.
ESE	Empirical Software Engineering.
FaaS	Function as a Service.
FDR	Flight Data Recorder.
GA	Genetic Algorithm.
GPL	General-Purpose Languages.
GSD	Global Software Development.
GSE	Global Software Engineering.
GUI	Graphical User Interface.
ICA	Independent Component Analysis.
IDE	Integrated Development Environment.

JSON	JavaScript Object Notation.
LDA	Latent Dirichlet Allocation.
LSA	Latent Semantic Analysis.
LSI	Latent Semantic Indexing.
MOOC	Massive Open Online Courses.
MSA	MicroServices Architectures.
MSRA	Microsoft Research Asia.
MSRR	Microsoft Research Redmond.
NLP	Natural Language Processing.
OAG	Online Application Generators.
PLSI	Probabilistic Latent Semantic Indexing.
PM	Process Mining.
PSP	Personal Software Process.
SA	Software Analytics.
SCM	Software Configuration Management.
SCS	Source Code Systems.
SDA	Software Development Analytics.
SDLC	Software Development Life Cycle.
SDPM	Software Development Process Mining.
SLR	Systematic Literature Review.
SOA	Service-Oriented Architecture.
SPOTS	Software Process On-the-run Tracking System.
SRILM	SRI Language Modeling.
TDD	Test-Driven Development.
TSP	Team Software Process.
VCS	Version Control Systems.
XES	eXtensible Event Stream.

Part I.

Fundamentals

PART I : FUNDAMENTALS

Introduction
Chapter 1

State-of-the-Art
Chapter 2

PART II : SOFTWARE PROCESS IMMERSION

Assessing Teams' Efficiency
Chapter 3

Unveiling Process Insights
Chapter 4

PART III : TOWARDS THE PRESCRIPTIVE COMMITMENT

Practices and Fingerprints
Chapter 5

PART IV : CONCLUSION

Conclusions and Future Work
Chapter 6

This Part covers the motivation, scope, research problems and main contributions of this work and highlights the fundamental topics, such as: Software Development Process, Software Analytics and Process Mining. It also presents a Systematic Literature Review about Software Development Analytics in Practice.

Chapter 1.

Introduction

Contents

This chapter introduces the motivation and scope, describes problems faced by software development and suggests methods to overcome those. Finally, it summarizes the expected contributions for Software Engineering.

Companion Soundtrack: Time - Hans Zimmer (Inception Soundtrack)

"...All things -from the tiniest virus to the greatest galaxy- are, in reality, not things at all, but processes..."[1]

—Alvin Toffler(1928-2016)[2]

1.1 Motivation and Scope

Failed software development projects happen neither unexpectedly, nor by coincidence. They are the result of a sequence of activities, events, and decisions that produce, unfortunately, undesirable outcomes.

As common examples, we may find that inaccurate planning and/or project plan deviations, improper resource allocation, or the inability to take corrective actions adequately, cause substantial financial losses on software development projects [62, 145].

On the opposite side, when one faces a successful project, the search for the most relevant factors towards success is the groundwork to comprehend how to prosper in software development related activities [7, 45].

Software development is intrinsically a process[3] - *"a series of actions that you take in order to achieve a result"*. Accordingly, a software development process is a series of activities performed by developers in order to create a new product or maintain an existing one.

Regardless of the business domain under consideration, it is commonly accepted that better processes tend to produce higher quality products. This rationale is supported by many reputable authors, which spent a major part of their research and professional lives around quality and process improvement in production lines, companies, or even entire industries [48, 58, 60, 66, 74, 98, 193].

To fully understand a process, it is required that all the facets of it are, in a timely manner, properly mined [206]. In trying to comprehend the software development process, many data sources, methods, and tools have been used and validated, but some others are yet to be fully exploited [136]. For example, since Version Control Systems (VCS), are widely used by developers, researchers get easy access to historical data of many projects and use file-based VCS as the primary source of code evolution data. Although it is practical to use such repositories, research based on VCS data is incomplete and imprecise. Moreover, answering questions that correlate code changes with development process activities (e.g., test runs, refactoring) becomes almost impossible [157].

In the following sections, we highlight some of the limitations in mining traditional software repositories with the purpose of improving the software development process. In the subsequent chapters, we elaborate on a few proposals to mitigate them.

[1] In *"Future Shock"*, 1970.

[2] American writer, futurist, and businessman known for his works discussing modern technologies, including the digital and the communication revolutions, with emphasis on their effects on cultures worldwide.

[3] https://dictionary.cambridge.org/dictionary/english/process

1.1.1 Aviation Industry Metaphor

Fortunately, tragic aviation accidents are rare. However, when they occur, we hear the media mention something about the search for the planes' black boxes. These devices contain vital information that can reveal why a particular airplane might have crashed. In fact, they helped airline manufacturers, aviation professionals, and accident investigators, improve flight safety for everyone since the 1950s. Black boxes were proposed by David Warren, an Australian research scientist which helped to investigate the repeated plane crashes of a commercial airliner back in those days. He argued that if researchers had knowledge of what happened on the plane right before it went down, it would help them figure out how to improve the next flight.

Black boxes are technically made of two parts — the Flight Data Recorder (FDR), and the Cockpit Voice Recorder (CVR). During a flight, the FDR tracks information about the plane itself, like its direction, speed, altitude, weather conditions, and many other metrics flight related. The CVR records audio of the crews' conversations, radio transmissions, engine sounds, and alarm noises.

While mechanical or electronic failure is always a possibility, it has been noted by several sources that the most common cause of airline crashes is pilot mistake [187]. Since 1950, ≈50% of the crashes are assigned to pilot error, ≈23% to mechanical failures, and the rest to other causes, such as weather conditions or sabotage [16]. This is arguably why the CVR exists, and not solely the FDR.

Software development researchers and practitioners have long been using software repositories to study the software development process. However, it has been repeatedly claimed that these repositories have many flaws in terms of the process data [8, 112, 204]. Many studies claim to study the process underneath the product being developed. However, most of them use only artifact-related repositories to study what is frequently associated with human behavior and/or teams' activities.

Mapping the software development domain to the aviation industry, we may say that researchers are mainly using the equivalent of the FDR, which contains artifact-related data. We will show in this dissertation that a CVR-like mechanism could be of great value in the software development realm, and we argue that the IDE might be a fundamental component towards that goal. It aggregates some fine-grain data related to the communications, activities, decisions, and actions taken by software developers, testers, and project managers, which can be easily pushed to a common events repository for posterior analysis.

This dissertation leverages on this rationale by using fine grain event data captured from the IDE to discover developers' processes and behaviors, to confront executed tasks with initial plans, and to comprehend and improve the software development process in general.

1.1.2 *"Pollock Effect"*

The XX century has seen the birth of one of the most talented painters and a major figure in the abstract expressionist movement, Jackson Pollock [100, 101]. Despite many controversies around his lifestyle, he was admired for his innovative painting technique. It consisted of pouring or splashing liquid household paint onto a horizontal surface ('drip technique'), enabling him to view and paint his canvases from all angles. The final work would be the culmination of

numerous layers of paint on top of each other, resulting from multiple sessions of creative work, which, only when combined produce the overwhelming visual effects we found in Pollocks' art.

For many years, art critics, historians, and admirers have looked at Pollocks' art only as of the result of his geniality from the creative perspective. Nowadays, researchers, mainly physicists, argue that his work is not just the product of a creative session, it is also supported by a few laws of physics [89]. Many are convinced that these laws are able to capture the author fingerprints and authenticity, thus allowing to distinguish a genuine Pollock painting from a similar product created by another artist[4] that was inspired by Pollocks' technique.

Figure 1.1: Pollock Process/Painting

Despite many style differences to antecedent painters, one thing remained unchanged between them - the painting was the final delivery. However, the means by which Pollock achieved it, coins the variation to other artists. The workspace conditions were different: canvas on the floor, ability to paint from multiple perspectives, using different materials and objects, exploring fluid dynamics, and finally, letting gravity perform the final touch.

Jackson Pollock, unconsciously, created not only a new painting technique that inspired many artists after him, but he also authored a new painting process[5].

Software development processes have some similarities with Pollock's process. It consists of many overlaid sessions of work of one or many individuals, having a multitude of workspace conditions, looking and interacting with it from multiple angles, and yet contributing to a unified product build. These work sessions overlap in time and when we look at them globally and from different perspectives, many insights come to the daylight. Several aspects, which initially may appear the result of an intrinsic talent, can potentially be seen as the outcome of a structured process that science may explain.

[4] In Figure 1.1 (right), Adam Zafrian art inspired by Pollocks' painting process - https://bit.ly/35aivrK

[5] In Figure 1.1 (left), Jackson Pollock at work in 1949, photographed by Martha Holmes. Pollock appears to be using a relatively viscous paint that forms a continuous jet of fluid he controls by moving a trowel up, down, or across the canvas. (Photograph @ Time, Inc, Getty Images) - Adapted from [89].

In the software development process, developers' implicit behaviors, actions, and decisions are yet to be fully researched and understood. As such, in the next sections and chapters, we will emphasize how important is to mine infrequently used software development repositories, from different perspectives and with refreshingly new methods.

1.1.3 Software Development Process

Even if a uniform definition for the software development life cycle does not exist, we find its phases generally grouped as: Requirements Elicitation, Coding, Debugging, Unit and Functional Testing, Maintenance, Deployment, and Operation.

Software projects outcomes and pertinent tasks are frequently evaluated across four perspectives: quality, scope, time, and cost [45]. These perspectives, are related with the planning and execution of the project's main activities. Each perspective contains its own critical success and failure factors, which can be grouped into five different dimensions: organizational, people, project, technical, and finally, the followed process[195].

Starting a software development project from scratch is a complex activity on its own [95], mainly due to the frequent absence of a methodology or formalized process [160] which may acts as a referential. We now have enough evidence to suggest, that in addition to the initial project planning, the way people are organized, the tools they use and the processes they follow are fundamental features for the success or failure of any software project [160].

Even if prescribed process models exist, projects often do not comply with them. This happens mainly because each developer or team has some freedom to interpret the process and because its compliance is not verified on the run, since it is mainly intangible. As a consequence, researchers found that process executions (i.e. projects) often deviate from what was planned [120].

Software development became a fundamental process on any business or organization, therefore, it is vital to carefully study, understand, and improve it[72]. It is surprising though, that so many studies around the software development process, end up using, as a data source, data/metrics not about the activities, tasks, or actions taken by the developers along the way, but instead, about the outcome: the product [1]. Moreover, when those studies use process metrics, they are based on inputs from management tools like Software Configuration Management (SCM) or Bug/Issue Tracking Tools (BTT), meaning, they are based on what the developers said they have done, and not on irrefutable evidence of what (and how) they actually did. Besides, some repositories, as seen in 1.1, have flaws that may lead managers to make estimations, such as effort and duration of projects, based on potentially problematic illusions.

1.1.4 Software Effort Estimation Illusions

Developers following best practices use some sort of repository[6] to keep track of changes made to the artifacts. They perform check-outs to start adding, deleting, or changing code, and then execute check-ins when those tasks are finished. By accessing those repositories, one will know with some detail how the software was in moment **t0** and then, how the software became on moment **t1** once changes were committed.

[6](Concurrent Versions System (CVS), Subversion, Mercurial, Github, etc)

However, the period between **t0** and **t1** remains a black box. What were the developers doing between commits? What features of the IDE were mostly used? What third-party IDE plugins were used ? What type of environment had the developer? (IDE version, Java version, Operating System). Are there any correlations between the complexity of the process and the final complexity of the product? What are, in reality, the causes and effects of a development process?

As we will see in chapter chapter 2, the literature does not provide good examples answering those questions, even if only partially.

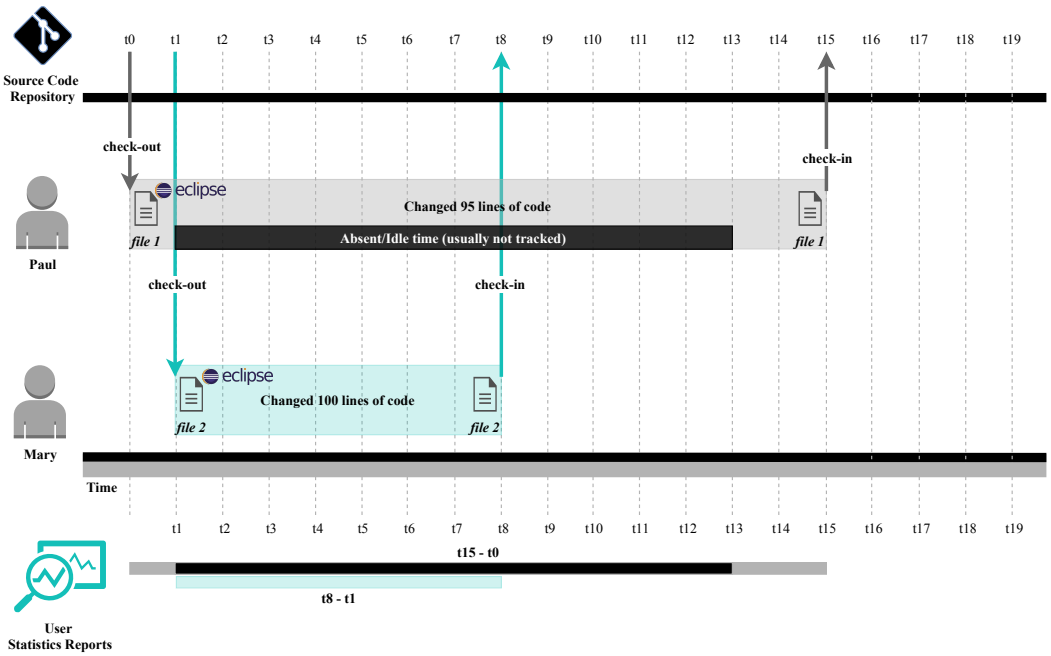

Figure 1.2: Productivity estimation illusion between developers

Figures 1.2 and 1.3 show two practical examples of how data gathered from traditional source code repositories may mislead an analysis of estimating the effort of a project.

- In 1.2, developers **Paul** and **Mary** check-out two different files(*file1*, *file2*) for refactoring, with a single time unit difference. **Paul** check-in his file much later than **Mary** and with less absolute changes. However, because absent time is usually not tracked by source code control systems, **Paul** seems to be less productive than **Mary**. This is simply not correct, because **Paul** produced more code per effective period of work time than **Mary**. Besides, the complexity of the tasks they both performed is not possible to measure accurately in the statistics report.
- In 1.3, developer **Paul**, check-out two different files(*file1*, *file2*) for debugging. He checks-in both files at the same time. During the working period, in reality, he was idle for certain periods in both files, and in some other periods, he worked in only one file at a time. Statistics reports will show the same work time for both files, and more code changed in the file where he in fact worked less.

Possible Assumptions. A powerful insight we get from this analysis is that the empirical software engineering community might have been working with very erroneous/misleading

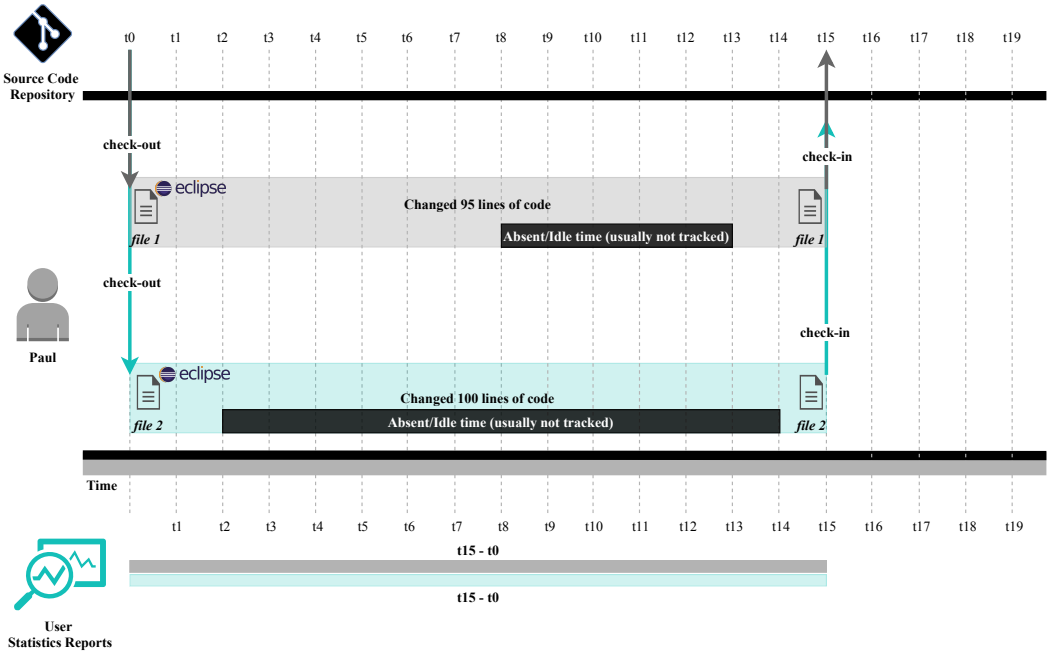

Figure 1.3: Developer workload allocation estimation illusion

datasets obtained from traditional software repositories, such as BTT and/or Source Code Systems (SCS) tools. If these datasets are used to derive conclusions on the effective effort taken by developers when working with some artifacts, those conclusions might very well be invalid.

1.2 Software Analytics

As a result of the direct or indirect dependence on software in our daily lives, and the importance it represents for the software industry, the software development process has become the focus for many studies and analysis [15, 43].

Defining new processes and allocating the right resources, particularly for large organizations, is a stimulating task for project managers, primarily because it requires acquaintance with existing processes and tools, the understanding of different stakeholders, and the coordination of technical expertise in multiple domains.

Failing to attain the above requirements may cause software development projects to produce non-coherent deliveries, with scattered technical artifacts that are hard to maintain, quite often overpass the allocated budget, and even more important, they may not have the quality which was initially aimed.

To mitigate the above risks, analytics was also brought into the software domain as a way to gather insights about the software products and processes. The goal was to provide stakeholders' decision-making process with data-driven evidence instead of the very often used gut feeling or personal experience from past projects.

Analytics has seen widespread adoption as a way to gather insights on many different realms. How people use collaborative/productivity management suites (see figure 1.4) and personal device usage (see figure 1.5) are among a few examples one may find present on our daily lives.

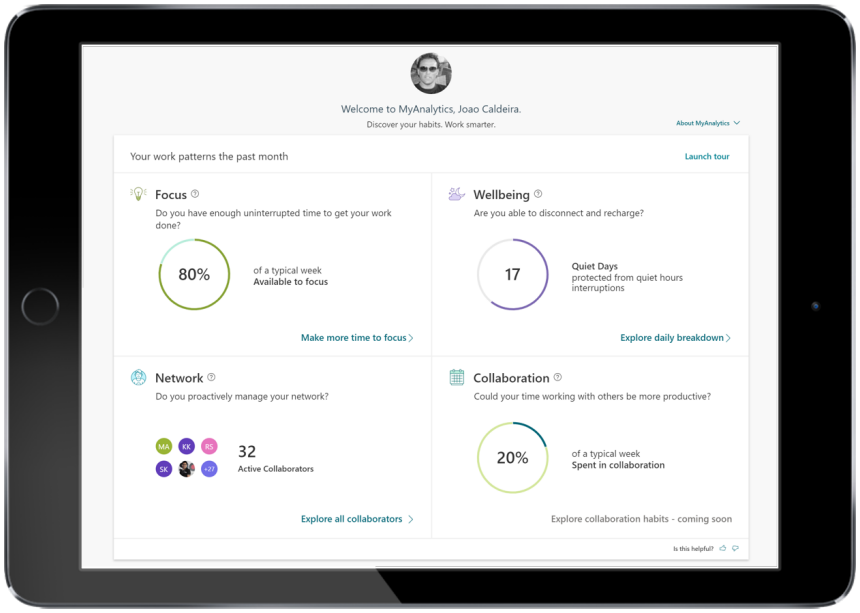

Figure 1.4: Email/Skype Usage

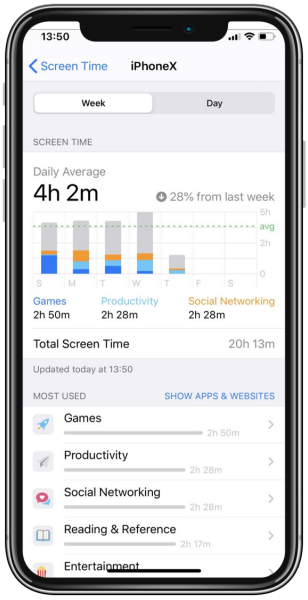

Figure 1.5: iPhone Usage

Software Analytics (SA) was first coined and proposed by Dongmei Zhang, founder of the SA Group at Microsoft Research Asia (MSRA). After a series of articles, tutorials and talks, the term became well known in the Software Engineering research community.

1.2.1 Software Development Analytics

In late 2010, the term SDA (Software Analytics with focus on Software Development) was proposed by Thomas Zimmermann and his colleagues at the Empirical Software Engineering (ESE) at Microsoft Research Redmond (MSRR) in their FoSER 2010 paper [34].

Since then, a vast amount of literature was produced presenting stakeholders with new ways of improving the efficiency and effectiveness in developing software products by providing insights to streamline the processes or to optimize resource allocation [1].

SDA is currently a widespread method to gather insights from the software development process [76, 142, 166]. As this method evolved, the Software Engineering practice took advantage of lessons learned and applied them in real live scenarios [135].

Recently we have seen the birth of a multitude of analytics related companies, solutions, and methodologies [135, 166, 202]. This was promoted mainly by the large attention practitioners gave to Artificial Intelligence, more precisely, to Machine Learning techniques. It was also a period where Process Mining saw boundless adoption in several business domains [73, 206, 208].

Both approaches, Machine Learning, and Process Mining are today being used to reduce the costs of producing new software products, to improve their quality, reduce time-to-market and support the decision-making process. These practices would not be possible or effective without the possibility of mining multiple repositories.

1.2.2 A Multitude of Repositories

Together with other emergent technologies, new development platforms (IDEs) are being created, mainly in the cloud (e.g., Eclipse Orion, Cloud9, Codio), requiring different approaches on the way software development can be studied.

Empirical studies on software development most often are based on data taken from repositories such as: SCM, SCS, BTT, App Stores and Wikis, just to name a few. Rarely, IDEs are used as a data source, either because they do not easily record data regarding developers' activities and the associated context, or because new perspectives to analyze that data were not yet unfolded, as we show in chapter 2.

The increasing role of the open-source movement has allowed a considerable increase in the availability of free software engineering tools, namely in Eclipse, the dominant IDE for its most widespread programming language, Java. That availability has led to the progressive use of a multitude of tools during a software development project, far beyond the traditional ones that dominated the first decades of computer programming, namely the code editor, compiler, and linker.

Figure 1.6 shows in single clipart, a large set of repositories[7] mentioned in [202] with potential to advance analytics in the software development practice.

Nowadays, developers use code and test generators, modeling, and code recommendation tools, code smell detection and refactoring plugins, metrics collection adapters, and software structure visualization tools, to name just a few. Those tools are becoming increasingly intertwined within the IDE and the latter is progressively integrated with cloud-based services that allow cooperative work (e.g. GitHub, Sourceforge) that provide services such as Configuration Management System (CMS), BTT, project documentation and Wikis. The importance of the IDE is the foundation for this dissertation, therefore, we will provide details on it in the next sections.

1.2.3 The Integrated Development Environment

Nowadays, most software practitioners develop their work upon an IDE, such as `Eclipse`, `IntelliJ IDEA`, `Netbeans` or `Visual Studio Code`. To a greater or lesser extent, those IDEs support different software development life cycle activities, such as requirements elicitation, producing analysis and design models, programming, testing, dependencies management, or continuous integration. As an example, `Eclipse`, which owes its wide adoption to the vast plethora of plugins available in its marketplace, is customized for specific users/purposes, such as for modelers, programmers, testers, integrators, or language engineers.

An IDE, in addition to the artifacts it handles, contains metadata about the developers' activities that may reveal the reasons why some individuals and teams are more efficient than others. Moreover, it may have hidden in its usage, parts of the logic why some projects are successful and others fail.

We advocate that those development activities can be identified by mining the large number of events created during the execution of the IDE core components and the installed plugins.

[7] The IDE was kept in the middle to represent a fundamental component in studying the software development process.

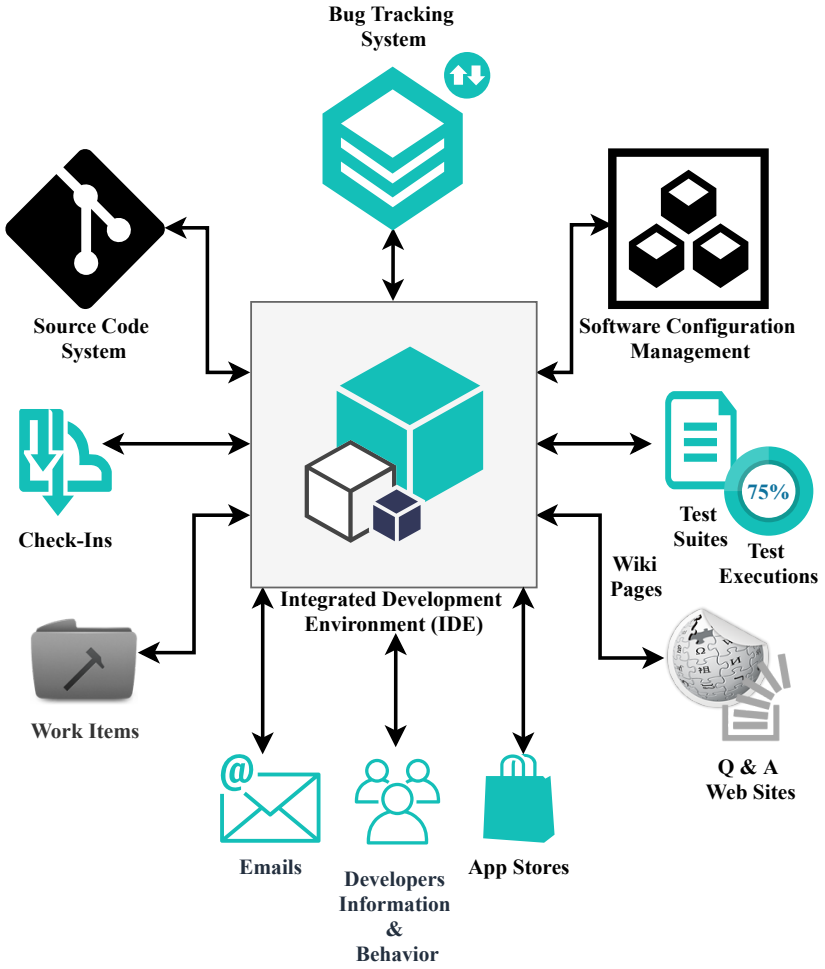

Figure 1.6: A few repositories identified in the Dagstuhl 2014 Seminar Report on SDA [202].

From those, one may expect to discover process models and consequently, detect hidden patterns or build prediction models to provide operational support to the software development process.

1.3 On the Role of Process Mining

Process Mining is now a mature discipline with validated techniques producing accurate outcomes on several business domains [165, 208]. A process mining project, if best practices are followed [126], should use goals, questions to answer, and event logs as inputs, and produces actions to implement as outputs as seen in Figure 1.7. The goals may consist of improving some performance indicators, such as time, risks, and costs associated with a specific process, or simply maximizing a service level. Actions may be the redesign of a specific project, adjust a current process or, if there is a fluctuation in process instances, one may want to include more resources.

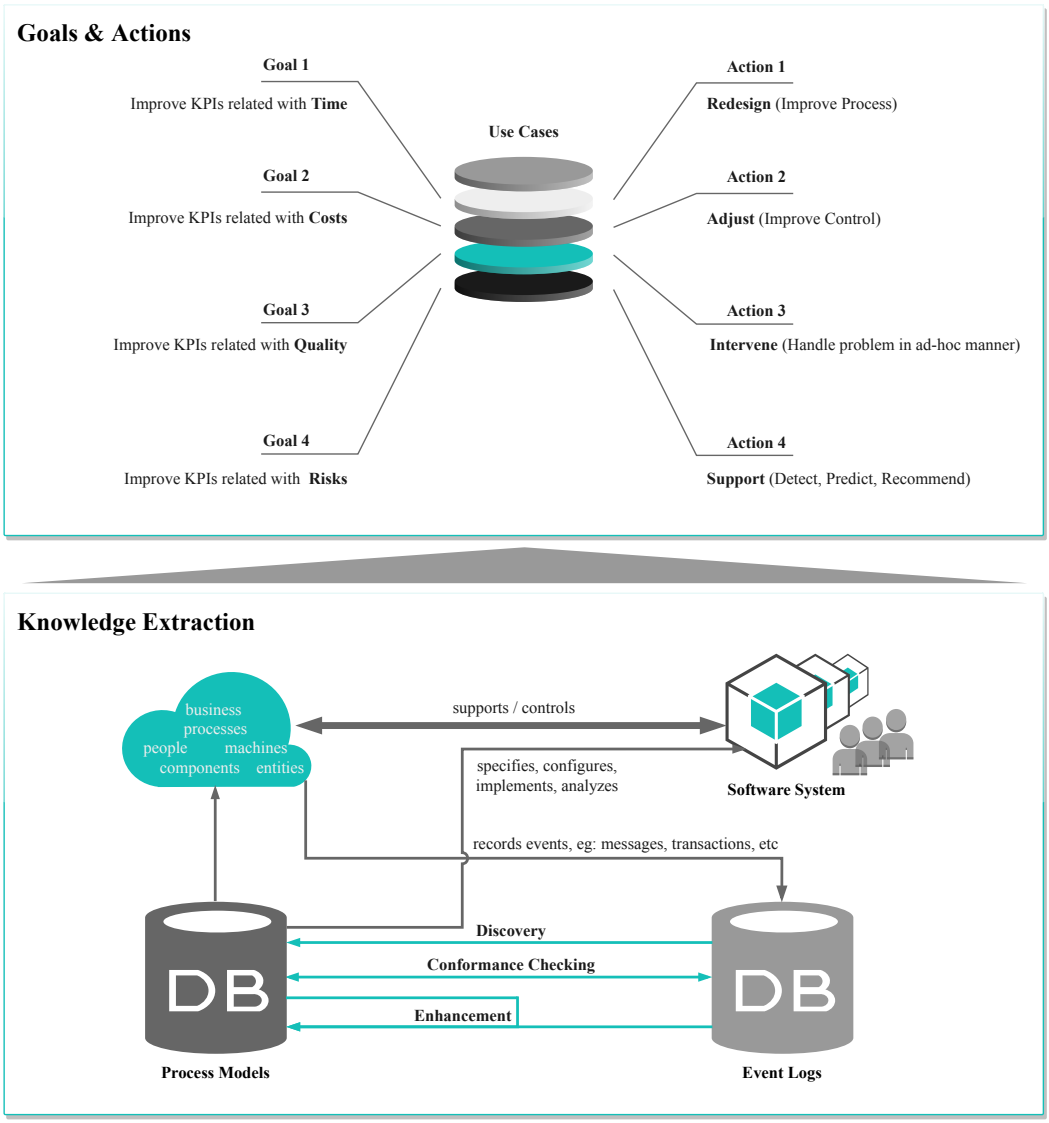

Figure 1.7: Process Mining: Data Science in Action. Adapted from [206]

Within the most used process mining tasks we have:

- **Discovery.** This stands for the ability to construct a process model, thus capturing its behavior based on an event log [206]. Several well-known algorithms exist to discover process models, such as the α-algorithm, the heuristics, genetic, inductive, and fuzzy miner, amongst others [73]. Table 1.1 shows the most relevant algorithms and their applicability in real-life scenarios.

- **Conformance Checking.** Commonly used to confront the model discovered with the reality consisting of events in the event log. It can be used to check for deviations from prescribed processes, detect differences and/or similarities between cases and verify the accuracy of documented processes [206].

  It can also be used to calculate the efficiency or to measure the quality of a process model. Quality is normally assessed considering four dimensions:

- **Fitness.** Represents how much behavior in a log is correctly captured (or can be reproduced) by a discovered model.

- **Precision.** Refers to how much more behavior is captured in the model than what was observed in the log. It deals on avoiding overly under fitted models.

- **Generalization.** Focuses on avoiding overly precise models based on the assumption that logs are by their nature incomplete, meaning that, to a certain extent, a model should be able to reproduce not yet seen behavior in the log.

- **Simplicity.** Alludes to the rule that the simplest model that can describe the behavior found in a log, is indeed the best model. This means that the complexity of a model is dependent on the number of nodes and arcs in the underlying graph.

- **Enhancement.** This stands for how relevant information can be used to extend the model. It is the type of activity that provides operational support, the most ambitious form of process mining. Quite often involves the combination of process mining, statistics, and machine learning to improve process models aiming at understanding deeply processes and optimize them [206].

Figure 1.8 summarizes the best practices methodology for a process mining project.

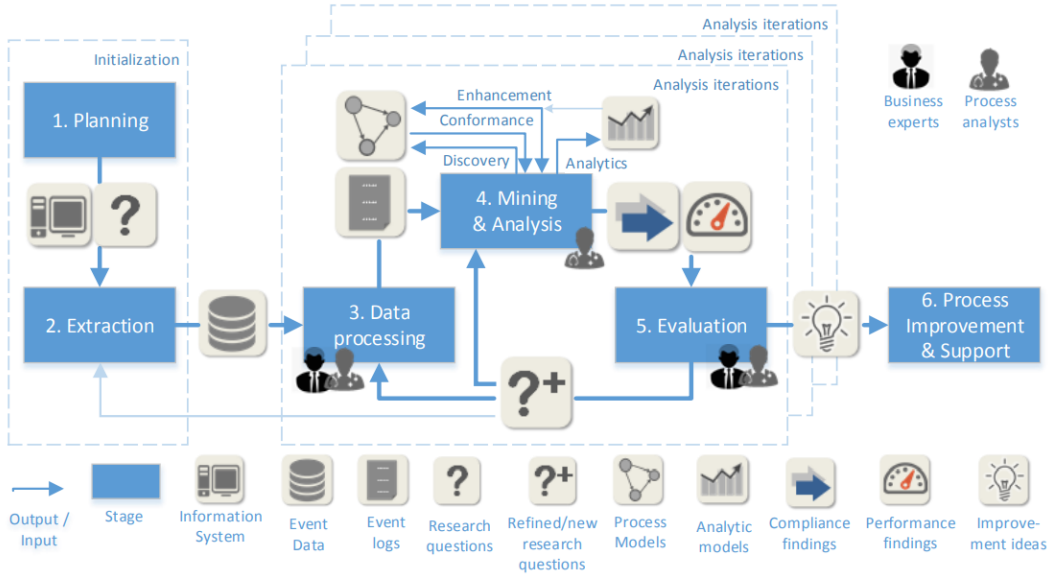

Figure 1.8: Process Mining Process Methodology. Adapted from [126]

Table 1.1: Commonly Used Process Mining Algorithms(Miners)

A/A+	Heuristics Miner	Inductive Miner	Genetic Miner	Fuzzy Miner
Cannot handle loops of length one and length two / A+ Can handle loops of length one and length two	Takes frequency into account	Can handle invisible tasks	Mimics the process of evolution in biological systems	Support large numbers of activities and highly unstructured behavior
Invisible and duplicated tasks cannot be discovered	Detects short loops	Model is sound	Use elitism, crossover and mutation to build the population elements of the next genetic generation	Uses significance/correlation metrics to interactively simplify the process model at desired level of abstraction
Discovered model might not be sound	Does not guarantee a sound model	Most used process mining algorithm	Tackle non-free-choice, invisible and duplicate tasks	Cannot be converted to other types of process modeling languages
			Computationally Expensive	Can leave out less important activities (or hide them in clusters) if hundreds of them exist
Weak against noise			Robust to noise	Strong against noise
Inputs				
Event log file	Event log file	Event log file	Event log file	Event log file
Outputs				
Petri Net	Heuristic Net	Petri Net or Process Tree	Petri Net or W-F Net	Fuzzy Model
When to Use				
Not recommended for real-life data (Obsolete)	When you have real-life data with not too many different events, or when you need a Petri net model for further analysis	For discovering process delays, deviations, and animation of the model	When all the behavior in the log needs to be represented by the model	When you have complex and unstructured log data, or when you want to simplify the model in an interactive manner

1.4 Research Drivers

In section 1.4.1, we initiate our research process by formulating the main problems we are undertaking and highlight why they are important. This drives the research objectives presented on section 1.4.2. The methods used to answer them, and the main contributions of this dissertation are described in section 1.4.3.

1.4.1 Research Problems

Software development is a socio-technical activity. Every project has its own needs in terms of requirements, technologies, and human resources that should be allocated to each task. Successful software development projects not only require people with the right programming skills, but also with the right behavior. Productivity is derived from both: skills and behavior [71].

A major difficulty in identifying the best human resources for a specific project is caused by the fact that we often have no clue on how programmers behave individually or in groups. In some cases, reports are not accurate, as seen in 1.1.4. In other situations, studies on software-related topics cover only product issues. As for the ones targeting the process dimension and developers behavior, many open research problems have been identified [8, 71, 155].

As such, we summarize the main problems as follows:

- **RP1. Incompleteness of the repositories used to assess software processes.** Many studies use software repositories as data sources for software development process-related analysis. These repositories, although important, have many flaws and do not reflect all the activities developers perform during their daily duties, as recognized by researchers and practitioners [106, 196, 203]. Alternative and/or complementary perspectives have been called for [32, 34, 35, 78, 111, 135, 136, 137, 202, 226].

  To the best of our knowledge, only a few satisfying approaches have been made public. The Eclipse Usage Data Collector[8], with filtered datasets published[9] [135], Mylar [149] and the Mylyn Monitor [109, 150], the Fluorite[10] [220, 221, 222], the Coding Spectator [155, 157, 204, 205], and more recently, the ActivitySpace[11] [17] . All of these projects seem to be on hold, discontinued, or not anymore available for public use. Apart from that, as we don't have the context associated with the data collected, none of them is very useful when one wants to make use of advanced process mining techniques.

- **RP2. Inability to accurately express developers' workflows and assess their impact on the software product and process dimensions.** A common practice in analyzing software development processes is the use of methods such as statistics, machine learning, time series, and others. These methods are proved efficient in producing descriptive statistics and valid results on predicting the number of software failures, time to solve defects, and summarize developers time on certain tasks [14, 18, 52, 83, 138]. However, mere numbers

[8] https://www.eclipse.org/epp/usagedata/
[9] https://archive.eclipse.org/projects/usagedata/
[10] http://www.cs.cmu.edu/fluorite/research.html
[11] http://baolingfeng.weebly.com/ase2015-demonstration.html

are just not enough, the above methods lack the ability to express the software development process as a sequence of activities with a clear beginning and end. In summary, they fail to represent the real processes followed, or in other words, they do not discover processes. As an example, one may want to know (discover) the real process followed by a team of developers and check the conformance against the process that was prescribed as guidelines or best practices by their organization.

Regarding the measurement of product and process dimensions, we observe they have been seriously debated in the software development domain [30, 39, 136, 209, 210]. Nevertheless, the comprehension of the liaison between them is still in its infancy [137].

- **RP3. Improper methods used in profiling developers due to the great variability of behaviors during their development sessions.** Software development analytics aiming to build models for development aid are frequently executed based on metrics extracted from the source code systems. In modern software projects, with the existing diversity of languages, methodologies, and tools, quite often, those metrics are not possible to be properly collected or shared. This, limits the ability to implement already proven models in the SDA practice [42, 129, 160]. Furthermore, the clustering techniques used to profile developers use probabilistic models that take mainly into consideration only the frequencies of activities and disregarding their order. Such an approach does not perform well in situations where the variability between developers is related to the flow of the activities, instead of the frequencies.

Detailed information about these and other research challenges are presented in chapter 2 - State of the Art where a Systematic Literature Review (SLR) is given.

1.4.2 Research Objectives

To address the research problems of section 1.4.1, the following research objectives were formulated:

- **RO1.** Evaluate IDE-based event data and Process Mining as valid options to advance SDA practices.
- **RO2.** Using only process-driven metrics to comprehend developers' coding practices and the entanglement of product and process dimensions.
- **RO3.** Profiling developers' behaviors during software development sessions.

Each **RO** is related with a correspondent **RP** with the same index number located in section 1.4.1.

1.4.3 Main Contributions

This section presents a summary of the main contributions delivered with the development of this dissertation:

- **C1. To address the incompleteness of data sources related to software repositories, we proposed the mining of software development processes based on the IDE usage data and by adopting process mining techniques.** This approach, Software Development Process Mining (SDPM), which we proposed [36], allows us to reverse engineer the software development process by mining event logs taken from real software development activities. The artifacts used to collect the data, their installation manuals, user guides and a sample event instance are presented in Appendix A.2 - Software Development Process Mining Plugin for Eclipse and A.2 - Software Development Process Mining Plugin for PyCharm. The proven methodology we followed to embrace this approach, is detailed in [126].

  Process Mining (PM) is expected to be a valid contribution to expand comprehension of a software process by introducing one more of the so needed different data perspectives [34, 35, 202]. Analytics was imported to the software development domain (SDA) to provide stakeholders with data-driven insights. With our proposed method, SDA become expanded in breadth and depth, thus enabling software project managers and developers to make even more meticulous decisions.

- **C2. To express developers' interactions and compliance adherence to prescribed processes, we proposed the process discovery and conformance checking methods.** To foster a shared understanding of what the current software process really is, we used process discovery techniques. Activities are expected to characterize the underlying process model, since a given execution flow (sequence of consecutive or parallel) of activities, from start to end, corresponds to what we call a *"process instance"*. We used conformance checking techniques for diagnosing if actual software development activities (again captured as event logs) were following a given process model, and if not, how close were they. The implementation for this approach is presented in chapters 3 - Assessing Teams' Efficiency and 4 - Unveiling Process Insights.

- **C3. To improve the comprehension about developers behaviors and characterize development processes by means of fingerprints, we propose a stack of process mining and n-gram language models.** We used a stack of text and process mining algorithms to validate the outcomes of the fingerprints detected by assessing the behaviors of a group of developers. The implementation and validation for this approach is presented in chapter 5 - Practices and Fingerprints.

The main contents presented in this dissertation were submitted in the following venues:

1. **Software Development Process Mining: Discovery, Conformance Checking and Enhancement.** Proceedings of 10^{th} International Conference on the Quality of Information and Communications Technology, QUATIC2016, Lisbon, Portugal. - **Accepted.**

2. **Assessing Software Development Teams' Efficiency using Process Mining.** Proceedings of 1^{st} ICPM2019, Aachen, Germany. - **Accepted.**

3. **Software Development Analytics in Practice: A Systematic Literature Review.** Information and Software Technology Journal, 2020. - **Conditionally accepted.**

4. **Unveiling process insights from refactoring practices.** Computer Standards and Interfaces Journal, 2020. - **Submitted.**

5. **Profiling Software Developers with Process Mining and N-Gram Language Models**, Journal of Systems and Software, 2021. - **Submitted.**

Our approach is aligned with European recommendations for Software Engineering related studies [163] and has the potential to inspire some software industry businesses and, coincidentally or not, some products in the portfolio of established players in Software Development Analytics and Process Mining domains.

ProcessLabs.ai.

"Joao's work on Software Processes analysis has inspired us for years, and his latest research getting inside the developers IDEs has done it again. Gaining insights into Software Developers Behaviors opens a whole set of products and possibilities to help both the developer and the organization improve at multiple levels, João is laying out the map for us. Just to name a few, Joao's research can be the inception of beyond state-of-the-art products to better assess candidates prior to hiring them, to help developers improve in their craft by letting them learn from top tier developers, to ensure certain desired practices are enforced inside an organization (like Test-Driven Development (TDD) for example), or to understand or even predict the reasons of defects in software. Improving software developers productivity is a $300B untapped market, and we strongly believe Joao's work is a cornerstone for anyone willing to take on that challenge."

—Pablo Ferrari[12]

Celonis.

Recently introduced Task Mining into their portfolio. In Figure 1.10, we have an example of user interactions across different tools used on their daily process activities. This approach is based on the principles also addressed by this research.

[12]ProcessLabs Co-Founder and CEO(2020), https://www.processlabs.ai/, https://www.linkedin.com/in/pablo-ferrari/

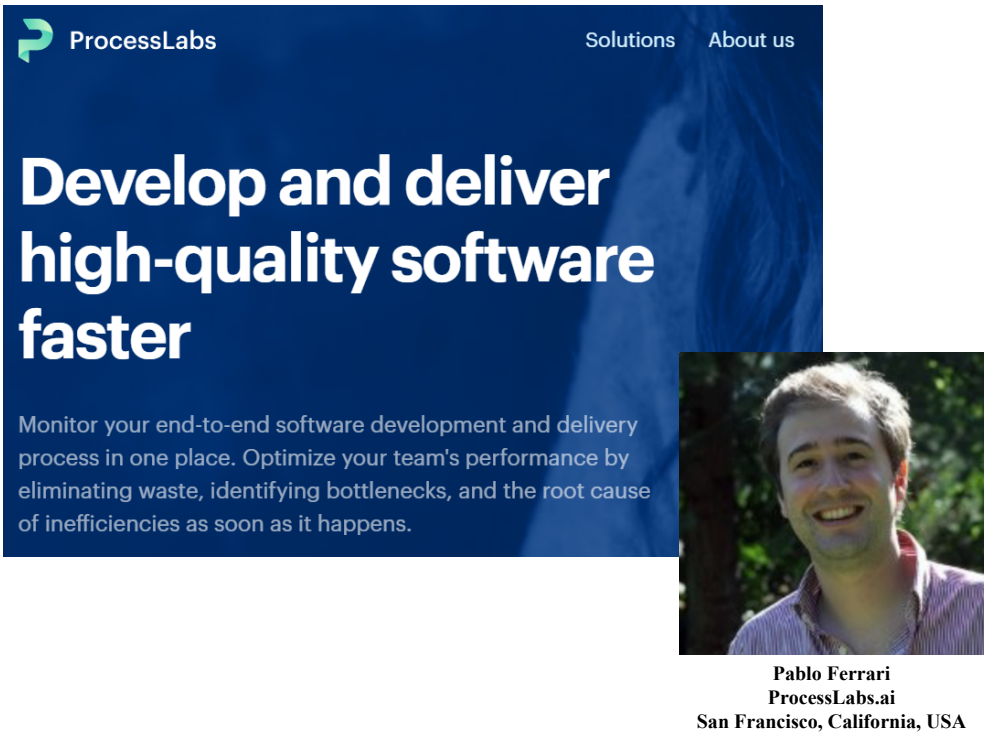

Figure 1.9: ProcessLabs portfolio covers a few aspects addressed by this dissertation contributions

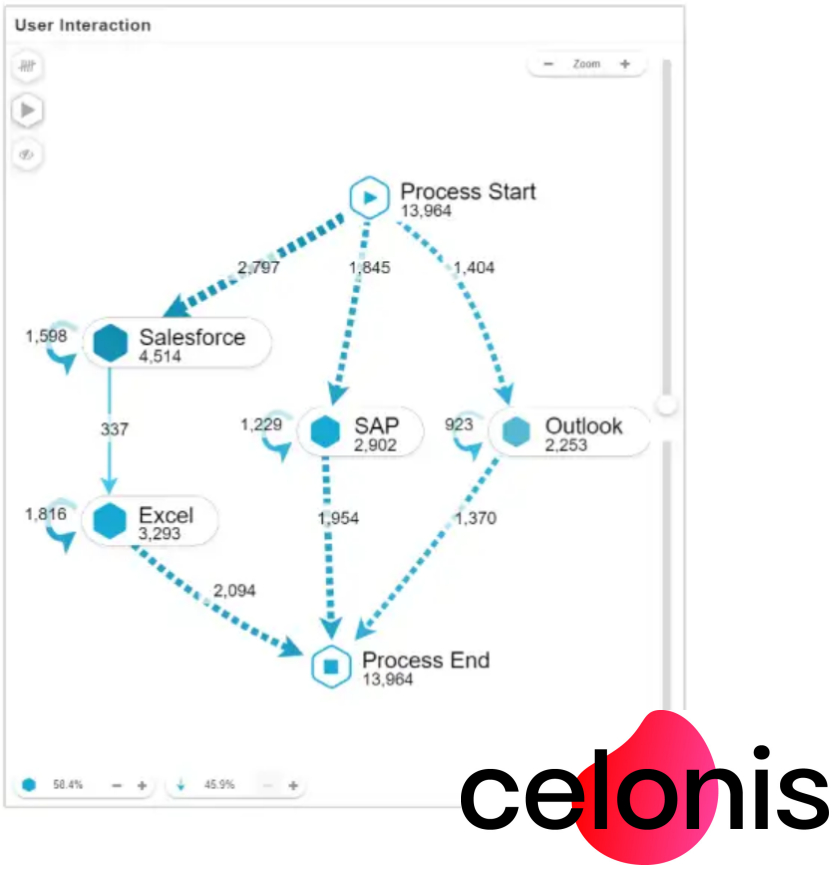

Figure 1.10: Celonis Task Mining product features

1.5 Dissertation Outline

This dissertation is organized in four parts, each containing a set of chapters which are briefly summarized as follows:

Part I - Fundamentals . Introduces this dissertation, the fundamental topics and a SLR.

Chapter 1 - Introduction. Provides context to this work, identifies some of the main problems in research, and detail the solutions prescribed to mitigate them. Finally, it summarizes the benefits and highlights the dissertation structure.

Chapter 2 - State of the Art. Gives an overview of the related work, proposes a taxonomy to categorize it and identifies research gaps within the SDA area.

Part II - Software Process Immersion. The solution proposed and its applications.

Chapter 3 - Assessing Teams' Efficiency. Presents the experiment conducted and the findings of assessing software development teams' efficiency using process discovery and conformance checking.

Chapter 4 - Unveiling Process Insights. Assesses the process vs. product complexities, their correlations and builds novel software models using only process metrics.

Part III - Towards the Prescriptive Commitment. An extension to the methods proposed towards the prescription of best practices to software developers.

Chapter 5 - Practices and Fingerprints. Demonstrates the feasibility of understanding deeply the developers behaviors and is a foundation for building a catalog of software process smells.

Part IV - Conclusion. Draws the conclusions and raises new research opportunities.

Chapter 6 - Conclusions and Future Work. Concludes and summarizes the achievements of this dissertation. As an evolutionary step it also provides some guidance and opens the discussion for challenges and future work in this area.

1.6 Summary

The first chapter provides an overview of the research work developed in the scope of this dissertation. Starts with a brief introduction of what a Software Development Process is in section 1.1.3, how Process Mining can help in studying such a process in section 1.3 and justifies the research problems present in section 1.4.1. Research objectives and main contributions are described, respectively, in section 1.4.2 and 1.4.3. Finally, it outlines, in section 1.5, what is included in each chapter. Figure 1.11 shows how the six chapters of this dissertation are organized, and how they are linked with the methods used to solve the problems and the correspondent contributions and benefits obtained.

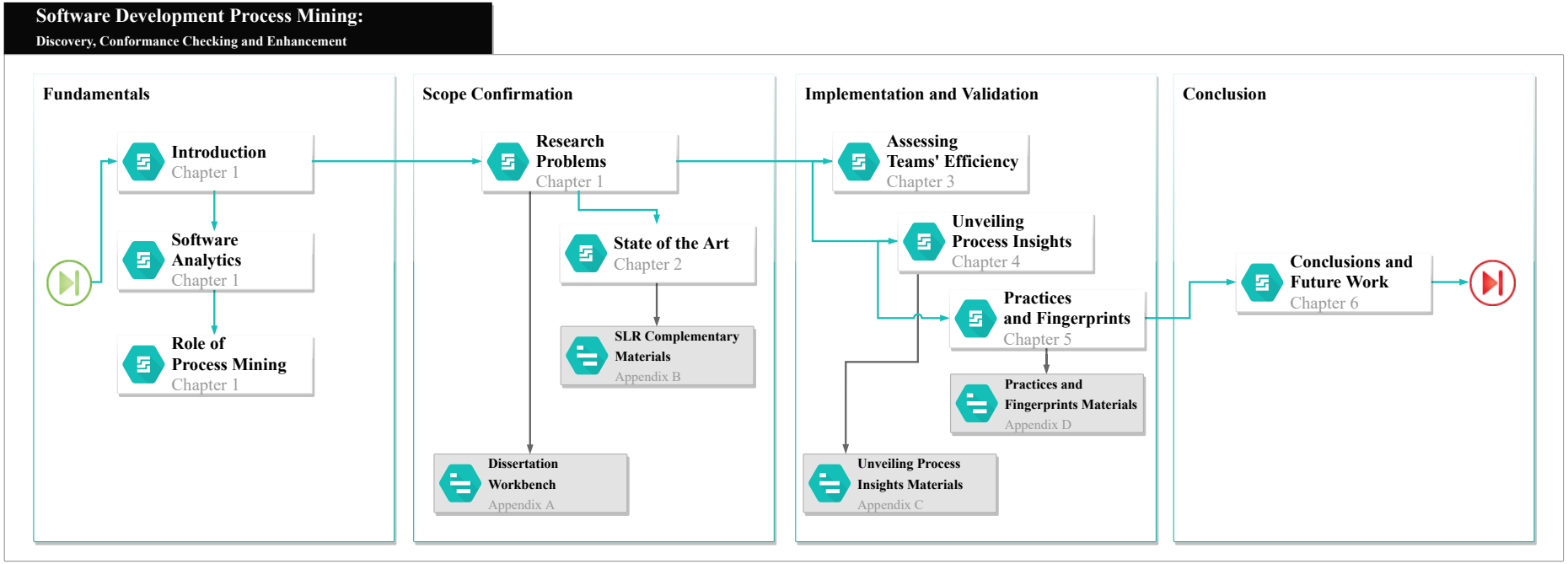

Figure 1.11: Dissertation Outline

Chapter 2.

State of the Art

Contents

This chapter present the protocol design, execution and findings of a SLR on Software Development Analytics in practice.

Companion Soundtrack: Time - Hans Zimmer (Inception Soundtrack)

"Science is a way of thinking much more than it is a body of knowledge."
—Carl Sagan(1934-1996)[1]

2.1 Introduction

Defining new processes and allocating the right resources, particularly for large organizations, is a challenging task for software project managers, primarily because it requires acquaintance on existing processes and tools, the understanding of different stakeholders, and the coordination of technical expertise in multiple domains [134]. Failing to properly manage these various aspects, namely when decisions are based on "gut feeling"(often dubbed "personal experience from past projects") may cause software development projects to produce hard to maintain technical artifacts, to surpass budget and schedule, and deliver defective products [62, 145].

The Software Development Analytics SDA research field aims at mitigating the aforementioned risks by providing the stakeholders' decision-making process with data-driven pieces of evidence, such as insights on software products and processes.

2.1.1 Motivation

The term "software analytics"SA was coined by Dongmei Zhang, founder of the Software Analytics Group at Microsoft Research Asia MSRA [224]. After a series of articles, tutorials and talks, the term became well-known in the software engineering research community. The SDA research field was proposed by Thomas Zimmermann and his colleagues from the Empirical Software Engineering Group ESE at Microsoft Research Redmond MSRR [34]. Since then, a vast amount of literature was produced presenting stakeholders with new ways of improving the efficiency and effectiveness in developing software products, by providing insights on how to streamline the processes or to optimize resource allocation [1].

2.1.2 Contributions

A decade has elapsed since the first discussions on methodologies, techniques and tools to boost the adoption of analytics in the software development practice. This systematic literature review SLR on SDA identifies, analyzes and aggregates the relevant primary studies in this period, following a well defined protocol, aligned with the best practices [61, 114]. Its main objectives are to:

- summarize the main types of empirical studies performed, target software life cycle activities, and corresponding data sources;
- identify the mining methods and analytics that were applied;
- evaluate the contributions of the selected primary studies;

[1] American astronomer, cosmologist, astrophysicist, astrobiologist, author, science popularizer, and science communicator in astronomy and other natural sciences.

- define a taxonomy to classify the impact provided by each primary study on software development dimensions such as: quality/technical debt, time, costs, risks and security.

2.2 Background

Mining software repositories is currently a widespread method to gather insights from the software development process [76, 141, 165]. As these methods evolved, the software engineering practice took advantage of lessons learned and applied them in real live scenarios [135]. The last decade has seen the birth of a multitude of analytics related companies, solutions and methodologies [135, 165, 202], often powered by machine learning techniques. It was also a period where process mining saw boundless adoption in several business domains [73, 206, 208]. Both approaches, machine learning and process mining, are nowadays being used to reduce the costs of producing software products, to improve their quality, reduce time-to-market, and support the decision making-process.

2.2.1 Related Work

Many SLRs have been published in the field of software engineering [114]. However, the ones addressing SDA concerns, from a holistic perspective, are scarce and often insufficiently detailed, since several aspects we deem relevant to advance the current state of the art are lacking or did not have exhaustive scrutiny. Notwithstanding, we briefly describe hereinafter all the systematic reviews whose scope somehow intersects the usual topics of SDA.

A SLR covering primary studies from 2000 to 2014, aiming to identify gaps in knowledge and open research areas in SA was presented in [1]. It considered 19 primary studies out of 135 and the authors concluded that the practitioners who benefited most from SA studies were developers, testers, project managers, portfolio managers, and higher management, with 47% of the considered studies supporting only developers. Maintainability and reverse engineering, team collaboration and dashboards, incident management and defect prediction, the SA platform, and software effort estimation were among the domains mostly studied, with 47% of them analyzing only one artifact. Based on their analysis, since most of the research addresses only the low-level analytics of source code, the authors recommended researchers to use more datasets, to achieve higher confidence level in the results. They also suggested to target higher-level business decision making profiles, like portfolio management, marketing strategy, and sales directions.

A survey of the publicly available repositories and the classification of the most common ones is presented in [176]. Authors also discussed the problems faced by researchers when applying machine learning or statistical techniques to them. The conclusions highlight the fact that some of the problems, such as outliers or noise, have been extensively studied in software engineering, whilst others need further research. They authors pointed out the need of further research work to deal with the imbalance and data shifting from the machine learning point of view and replication of primary studies.

A mapping study on the investigation of frequently applied empirical methods, targeted research purposes, used data sources, and applied data processing approaches and tools in

empirical software engineering ESE was reported in [225]. The goal was to identify new trends and obtain interesting observations of ESE across different sub-fields of software engineering on 538 selected articles from January 2013 to November 2017. The authors observed that the trend of applying empirical methods in software engineering is continuously increasing and the most commonly applied methods are experiments, case studies and surveys, with open source projects being frequently used as data sources.

A systematic mapping study aiming at identifying the quantity, topic, and empirical methods used, targeting the analysis of how software development practices are influenced by the use of a distributed social coding platform like GitHub, was presented in [47]. The authors assessed 80 publications from 2009 to 2016, and the results showed that most works focus on the interaction around coding-related tasks and project communities. They also identified some concerns about how reliable were those results based on the fact that, overall, papers used small data sets and poor sampling techniques, employed a scarce variety of methodologies and/or were hard to replicate. As a conclusion, they attested the high activity of research work around the field of open source collaboration, identified shortcomings and proposed actions to mitigate them.

A systematic mapping study providing an overview of the concerns addressed in the different phases of the software development life cycle Software Development Life Cycle (SDLC), was published in [55]. Results are reported from different viewpoints and conclusions highlight that there is a considerable variation in the use of terminologies and addressing concerns in different phases of the SDLC.

Inspired by the increasing usage of data analytics in all areas of science and engineering, a systematic mapping study, aiming to investigate the usage of different types of analytics for software project management was presented in [154]. The authors analyzed the accessibility of the data, as well as the degree of validation reported in the final 115 studies selected for appraisal. Results provided evidences that the majority of studies were focusing on predictive and prescriptive analytics, with almost half of the studies being essentially predictive. When comparing information versus insight as the direction of analytics, the authors found that information oriented analytics (descriptive and predictive) had a greater number of related studies (60% of papers) than analytics searching for insight (diagnostic or prescriptive). As a final remark, their systematic mapping findings was compared with the results obtained by [35].

A systematic mapping study published in [9] aims at providing an overview of the subdomains, contribution types, research types, research methods and identify the role of software analytics in the field of “green software engineering". Findings show, that 163 papers out of the 260 initially found on digital libraries, used software analytical methods like statistical analysis and static analysis. Furthermore, only 11 out of the 50 papers kept for final data extraction, used software analytics techniques to foster green software engineering. Results revealed the need to develop new/improved automated software analytics tools for software practitioners, along with metrics explaining the correlation between energy usage and other quality attributes.

Our SLR aims to expand the existing knowledge about SDA, by adapting and extending the data perspectives, dimensions, and concerns identified and used by the above works. The

target properties we deem as most important for a primary study to be considered relevant in this SLR are the following:

- **Quality.** To assess the delivery of a good product or project outcome.
- **Scope.** To evaluate the meeting of requirements and objectives.
- **Time.** To track the project delivering on time.
- **Cost.** To manage the delivery within estimated cost and effort.
- **Reusability.** The use of existing assets in some form within the software product development process.
- **Maintainability.** To asses the degree to which an application is understood, repaired, or enhanced.
- **Evolvability.** Used to describe a multifaceted quality attribute to evaluate a software system's ability to easily accommodate future changes.
- **Performance.** To measure how effective a software system is with respect to the allocation of resources and correspondent time constraints.
- **Security.** A cross-cutting appraise that takes into account mechanisms, such as access control, and robust design to prevent software attacks.
- **Risk.** To address the possibility that one or more of the above properties are exposed to such levels of uncertainty that may lead them to produce undesired outcomes.

Based on this set, we propose a taxonomy to classify primary studies.

2.3 Research Methodology

In contrast to a non-structured review process, a SLR reduces bias and follows a precise and rigorous sequence of methodological steps to research literature [113, 212]. A SLR relies on a well-defined and evaluated review protocols to search, extract, analyze, and document results as stages. This section describes the methodology applied for those activities.

2.3.1 Planning the Review

2.3.1.1 Research Questions

This work is driven by the following research questions:
RQ1. *What type of empirical studies have been conducted in SDA?*
Justification. The list of the main types of studies reported in SDA literature can provide a comprehensive view, both for practitioners and researchers, not only to identify areas of opportunity, but also to optimize established methods.

RQ2. *What are the main data sources used for SDA related studies?*
Justification. Identifying those data sources is helpful, to provide soundness to the corresponding studies, to facilitate replication, and to stimulate the appearance of new datasets to address knowledge gaps in the field.

RQ3. *What type of process/project perspective analysis was conducted?*
Justification. It refers to the ability to identify if the studies are being done before **(pre-mortem)** or after **(post-mortem)** a process/project is finished. While the latter is more frequent, namely due to the use of existing software repositories, a pre-mortem perspective can add additional value in the decision making process, as taking corrective actions on a timely manner is fundamental to keep projects or processes on track.

RQ4. *What are the most studied SDLC activities?*
Justification. Understanding what SDLC activities are targeted the most (and those that are not), will help practitioners identify where most concerns and challenges are within the software development practice. It can also contribute to open new research streams to foster a deeper understanding of the complete SDLC.

RQ5. *Who were the target stakeholders of these studies?*
Justification. Software projects are risky to conduct and continue to be difficult to predict [34]. SDA in practice, holds out the promise to provide decision-makers with data-driven evidences in order to better manage risk, improve efficiency and effectiveness on development projects. Studies should address the needs of different stakeholders. Identifying those beneficiaries is vital to understand if the right tools, methods and insights are reaching the ones that most need support on their daily activities.

RQ6. *What are the main mining methods being used?*
Justification. Assessing the types of mining methods utilized helps to comprehend deeper the goals of past and current research, the limitations of their methods, benefits and conclusions and, highlight opportunities for novel approaches in future research.

RQ7. *Which type/form of analytics was applied?*
Justification. When exploring large volumes of data and many types of metrics, one may exploit different levels of analytics; **descriptive/diagnostics, predictive and prescriptive** [57]. Providing stakeholders in the development process with deep insights and potentially prescribing actions to take under certain circumstances is desirable. Predicting the future and prescribing actions are advanced forms of analytics which researchers and practitioners in the software development domain are expected to use.

RQ8. *What were the relevant contributions to the SDLC?*
Justification. On every single software development study, we should have clear benefits identified, either from using a new tool or by improving a process using a specific method. Failing to do so, reduces substantially the interest we may find in that literature and shortens the applicability of those methods in the field. SDA in practice is expected to contribute at least (but not limited to) to the following areas of concern in a software project: **technical debt/quality, costs, time, risk and security**.

2.3.1.2 Search Strategy

Search Terms. Based on the research questions, keywords were extracted and used to search the primary study sources. The search string consists of determine the main terms from the topics being researched, including synonyms, related items and alternative spelling. It is based on the same strategy used by [42] and is presented as follows:

("software analytics"OR "software development analytics") AND ("process mining"OR "data mining"OR "big data"OR "data science") AND ("study"OR "empirical"OR "evidence based"OR "experimental"OR "in vivo")

Digital Libraries Searched. A significant phase in a SLR is the search for relevant literature within the domain under study. To search for all the available literature pertinent to our research questions, the following digital libraries were queried:

- ACM Digital Library
- IEEE Xplore
- ScienceDirect
- Scopus
- SpringerLink
- Web of Science
- Wiley Online
- Google Scholar

Publications Time Frame. As mentioned earlier, the SDA research field emerged a decade ago. Since then, as studies have gained a more structured and formal approach, it makes sense to only account for publications in journals, conferences papers, workshops and book chapters, starting from January, 1st of 2010 till July, 15th of 2019.

2.3.1.3 Selection Criteria

We selected the above libraries based on the eagerness of collecting as many articles/papers as possible, not only because they are recognized as the most representative for Software Engineering research, but also are used in other similar works. Google Scholar was selected to account for articles eventually not published, but yet relevant to the software development domain.
Search Stages Overview. The outputs of the process followed to conduct the search is depicted in Figure 2.1. It compounds 4 sequential stages, which are described as:

Stage 1 - Retrieve automatically results from the digital libraries - The referred libraries were searched using the specific syntax of each database. The search was configured in each repository to select only papers carried out within the prescribed period. The automatic search was later complemented by a manual search, according to the guidelines of Wohlin [212], followed by backward snowball to complete the list of studies.

Stage 2 - Read titles and abstracts to identify potentially relevant studies - Identification of potentially relevant studies based on the analysis of title and abstract. Studies that are clearly irrelevant to the search and duplicates were discarded across the digital libraries. If there was

any doubt about whether a study should be included or not, it was included for consideration in a later stage.

Stage 3 - Apply inclusion and exclusion criteria on reading the introduction, methods and conclusion - Selected studies in previous stages were reviewed, by reading the introduction, methodology section and conclusion. Afterwards, exclusion and inclusion criteria were applied as defined in Table 2.1. At this stage, in case of doubt preventing a conclusion, the study was read in its entirety.

Stage 4 - Obtain primary studies and assess them - A list of primary studies was obtained and later submitted to critical examination using the 13 quality assessment criteria which is set out in Table 2.2.

Table 2.1: Exclusion and Inclusion Criteria applied at Stage 3.

Criterion	Description
	Exclusion Criteria(EC)
EC1	Studies published before 2010.
EC2	Studies not written in English.
EC3	Studies not related to the software development process.
EC4	Studies not supported by data collected on any well designed experiment or did not use empirical data from a third party.
EC5	Studies merely theoretical or based on expert opinion without locating a specific experience, as well as editorials, prefaces, summaries of articles, interviews, news, analysis/reviews, readers' letters, summaries of tutorials, workshops, panels, round tables, keynotes and poster sessions.
EC6	Studies aiming only at describing new development tools or works with the goal of simply assessing and/or validating new analytical methods without a clear statement to the benefits they may provide for the SDLC.
	Inclusion Criteria(IC)
IC1	Publications should be "journal" or "conference" or "workshop" or "book".
IC2	Works that put validated analytical methods into practice with the goal of understanding and/or improving the software development process.
IC3	Articles that clearly addressed any of the analytics depth (RQ7) and provided benefits for the SDLC on any dimension identified in RQ8.

2.3.1.4 Quality Assessment

The strategy to evaluate the quality of the studies is based on a checklist with thirteen criteria. The criteria were based on good practices for conducting empirical research [114] and in the Critical Appraisal Skills Programme (CASP) used in different types of publications [61].

The criteria developed to assess quality covered four main quality issues considered necessary when evaluating primary papers:

- **Reporting.** Three criteria (QC1-QC3) assess if the rationale, goals and context have been clearly stated.

- **Rigor.** Five criteria (QC4-QC8) evaluate if a meticulous and convenient approach have been applied.
- **Credibility.** Two criteria (QC9-QC10) check if the findings are well presented and the gathered insights plausible and/or credible.
- **Relevance.** The remain criteria (QC11-QC13) are related with the relevancy of the study for the SDLC, stakeholders and the research community.

Selection of primary studies. The quality of each publication should be assessed by the authors after the selection process in Stage 3. The checklist presented in Table 2.2 was used to assess the credibility and thoroughness of the selected publications. The steps that guided the selection of primary studies to reach the final results, are presented in Figure 2.1.

Table 2.2: Quality Criteria.

Criterion	Description
QC1	Is the paper based on research (or merely a "lessons learned" report based on expert opinion)?
QC2	Is there a clear statement of the aims of the research?
QC3	Is there an adequate description of the context in which the research was carried out?
QC4	Was the research design appropriate to address the aims of the research?
QC5	Was the recruitment strategy appropriate to the aims of the research?
QC6	Was there a control group with which to compare treatments?
QC7	Was the data collected in a way that addressed the research issue?
QC8	Was the data analysis sufficiently rigorous?
QC9	Has the relationship between researcher and participants been adequately considered?
QC10	Are the datasets available to the public, thus allowing replication ?
QC11	Is there a clear statement of findings?
QC12	Is the study of value for research or practice?
QC13	Did the study identified any clear benefits for the SDLC according to RQ8?

Each question was marked as "**Yes**", "**Partially**"or "**No**". We considered a question answered as "**Partially**"in cases where we could derive relevant contents from the text, even if the details were not clearly reported. These answers were scored as follows: "**Yes**"=**1**, "**Partially**"=**0.5**, and "**No**"=**0**. For each selected study, its quality score was computed by summing up the scores of the answers to all the quality criteria questions, being the minimum value admissible "**0**"and the maximum "**13**", in case all the questions were marked with a "**1**".

2.3.1.5 Data Extraction

To gather standard information regarding the papers under analysis, we created a data collection form as represented in Table B.1 in B.1. This data collection form helped us to identify the date, venue and authors of the publications and also how each of them addressed the topics of our research questions.

2.3.1.6 Data Synthesis

The synthesis aimed at grouping findings from the studies in order to: identify the answers to the **RQs** presented earlier in section 2.3.1 and were organized in a spreadsheet form. This data extraction process was manually conducted by the main author. The spreadsheet was loaded and analyzed using the R statistical engine[2] and has now been disclosed[3]. Obtained results, plots and findings are presented in section 2.3.2.

2.3.2 Conducting the Review

This phase is responsible for executing the actions defined in section 2.3.1.

2.3.2.1 Execute Search

We started the review with an automatic search followed by a manual search and afterwards applied the inclusion/exclusion criteria. The search as detailed in section 2.3.1.2, was performed in mid July, 2019, with the search string syntax being adapted to support the different search engines. Initially we identified 2769 articles, and upon reading their titles and abstracts, the dataset was reduced to 611 articles. Following, we filtered them with the inclusion and exclusion criteria. The complete workflow and results of the initial search and subsequent filtering phases, is depicted in Figure 2.1.

2.3.2.2 Apply Quality Assessment Criteria

The selection criteria was based on exclusions and inclusions. Table 2.1, defined, in section 2.3.1.3 those criteria used to assess remaining works in *Stage 3*. In case of any doubt, the study was kept for analysis at a later stage. *Stage 3* provided as inputs for *Stage 4*, 153 articles, which were then assessed in their quality dimension. At *Stage 4*, we applied the quality criteria described in section 2.3.1.4, resulting in 32 articles to further extract data and to answer the eight research questions.

[2] https://www.r-project.org, https://rstudio.com
[3] doi:10.17632/d3wdzgz88s.2

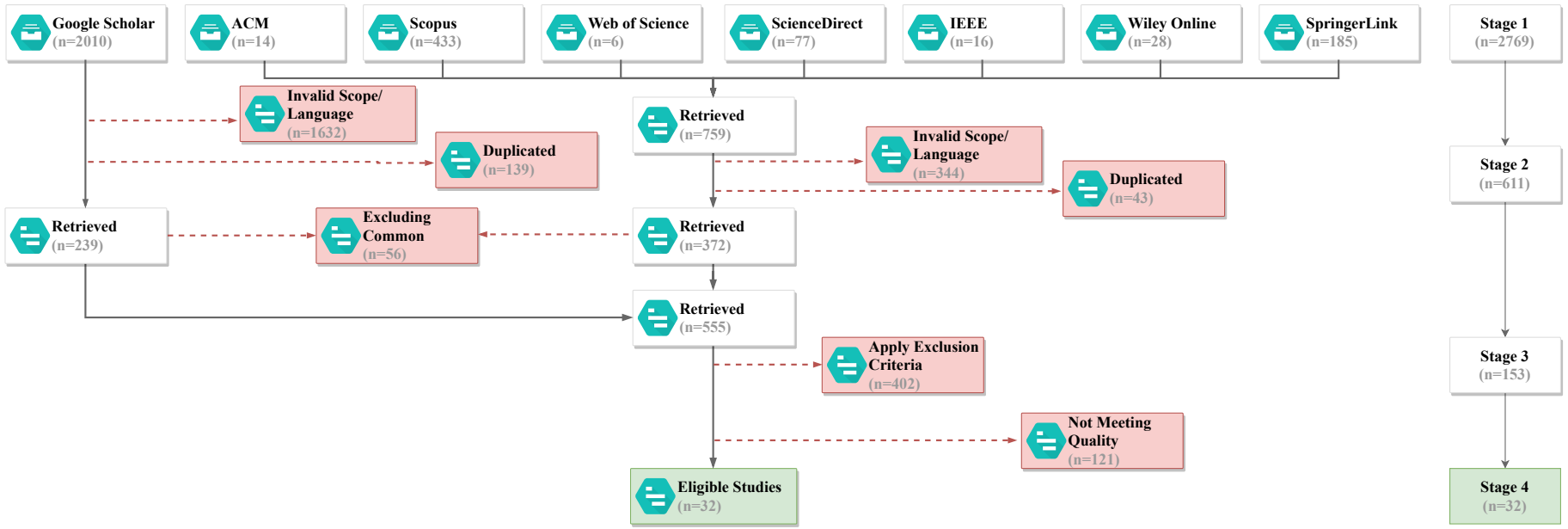

Figure 2.1: Study Selection Process Stages

We classified the studies quality level by plotting their descriptive statistics and analyzing the correspondent quartiles:

- **Min**:6, **1st Qu**:8.5, **Median**:9.0, **Mean**:9.007, **3rd Qu**:9.5, **Max**:12

As seen above, the third quartile is at score 9.5, therefore, we selected only the studies scoring above that mark. Based on the high level of quality, 32 studies were selected for final data extraction. Figure 2.2 shows the distribution of all studies per Year right after the quality assessment scoring task.

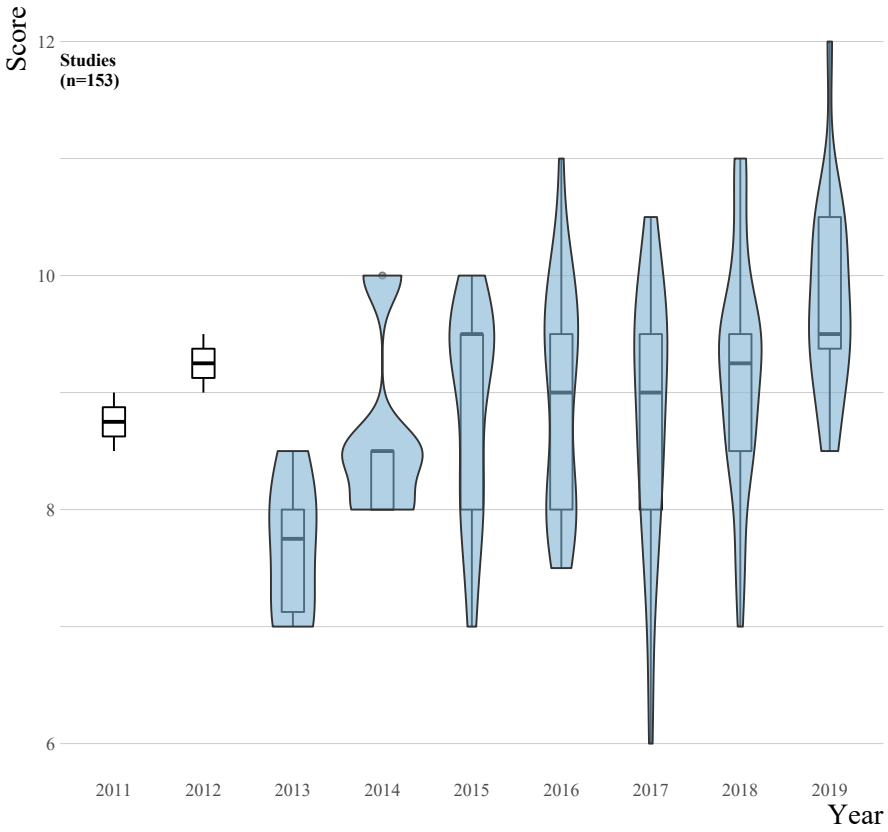

Figure 2.2: Studies score per Year at Stage 4

2.4 Document the Review

All selected studies and the details to support the statistics we show in section 2.4.1, are presented in Table B.2 in B.2. In section 2.4.2, we present the main findings, comments and answers to each of the research questions.

2.4.1 Demographics

Figure 2.3 shows clearly that the majority of the selected studies were published in journals. An increasing trend in these publications is also present.

The remaining articles were published in conferences with the exception of one which comes from a workshop. As it is possible to observe, only studies published between 2014 and

Figure 2.3: Number of studies per Venue Type per Year

2019 made the final stage of this SLR, and 75% of them were published in the last 3 years. This provides some indication that, not only SDA is a relatively new practice, but also, that it is becoming mature only in the very last few years of this decade.

Looking in-depth to the publication where the studies appeared, we easily find that the Empirical Software Engineering Journal has a strong dominance among all the others. The distribution of studies per Publication over the Years is presented in Figure 2.4. Here we can observe that only the Software Quality Journal and the Journal of Systems and Software have more than one study published within our final set of articles.

Regarding authorship, which we present the details in Figure B.1, we found that only 3 main authors appear with 2 studies in the selected papers. All the remaining authors are present with only one publication and none of them appear with more than one study per year. This may resonate the difficulty that is to setup, document and publish such type of studies. Figure 2.5 present the frequency of contributions regarding continents, countries and institutions involved, either as primary or secondary authors, on all studies.

2.4.2 Analysis and Findings

It is widely accepted that we lack experimentation in Software Engineering in general. This phenomenon is even more acute on what concerns experimentation related with analytics in practice for software development. Even if this work is scarce, we should look at it collectively to try to draw some picture of the current state-of-the-art. For that purpose, a summary table with the complete information extracted to answer all the **RQs**, is presented in Table B.6 in B.5.

Figure 2.4: Frequencies of studies per Publisher over the Years

In this section we present each research question and the correspondent dimension findings and their frequencies[4].

RQ1. What type of empirical studies have been conducted?

According to the type of empirical studies provided by [223], from the total number of publications, more than half, 53.12%, are Exploratory Case Studies. Quasi-Experiments and Exploratory Case Studies combined account for 90.62%. This is probably not a surprise, since the remaining study types are, quite often, harder to setup due to technical limitations in the data collection process or blocked by data privacy concerns raised by the involved entities.

One publication, [S13], combines three study types: Exploratory Case Study, Quasi-Experiment and a Survey. Having two types of empirical studies presented, we find [S31] and [S23] which combine a Exploratory Case Study and a Survey. Having a Quasi-Experiment and a Survey we have [S6] and [S24]. The remaining publications have only one empirical study type given. Study Types found and the plot of their distribution per Year is shown on Figure 2.6.

No Controlled Experiment, Meta-Analysis, Experience Report or Discussion had quality to reach the final stage of this SLR. Particularly for the Controlled Experiment studies absence,

[4]The sum of frequencies might be bigger than the total number of selected studies(n=32) because some publications have more than one Study Type, Data Source, SDLC Activity, Stakeholder, Mining Method and/or Analytics Scope.

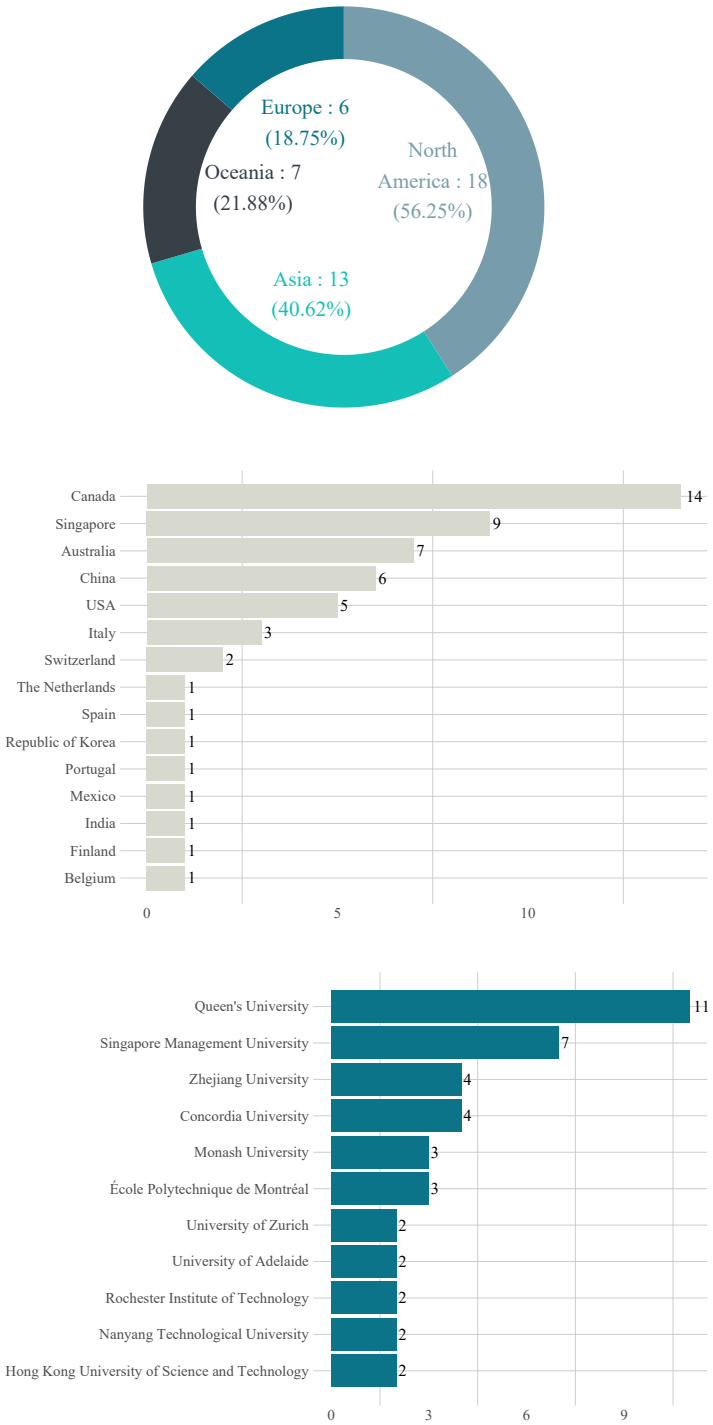

Figure 2.5: Number of studies per Continent, Country and Institution(> 1 study only)

its worth elaborate that a controlled experiment is one in which all factors are held constant except for one: the independent variable. It is common to compare a control group against an experimental group where all factors are identical between the two groups except for the factor being tested. This approach has the advantage that is easier to eliminate uncertainty about the significance of the results, however, it also has a considerable drawback: the effort needed to design and execute such experiments.

We believe that sufficient conditions needed to conduct such experiments are not yet being

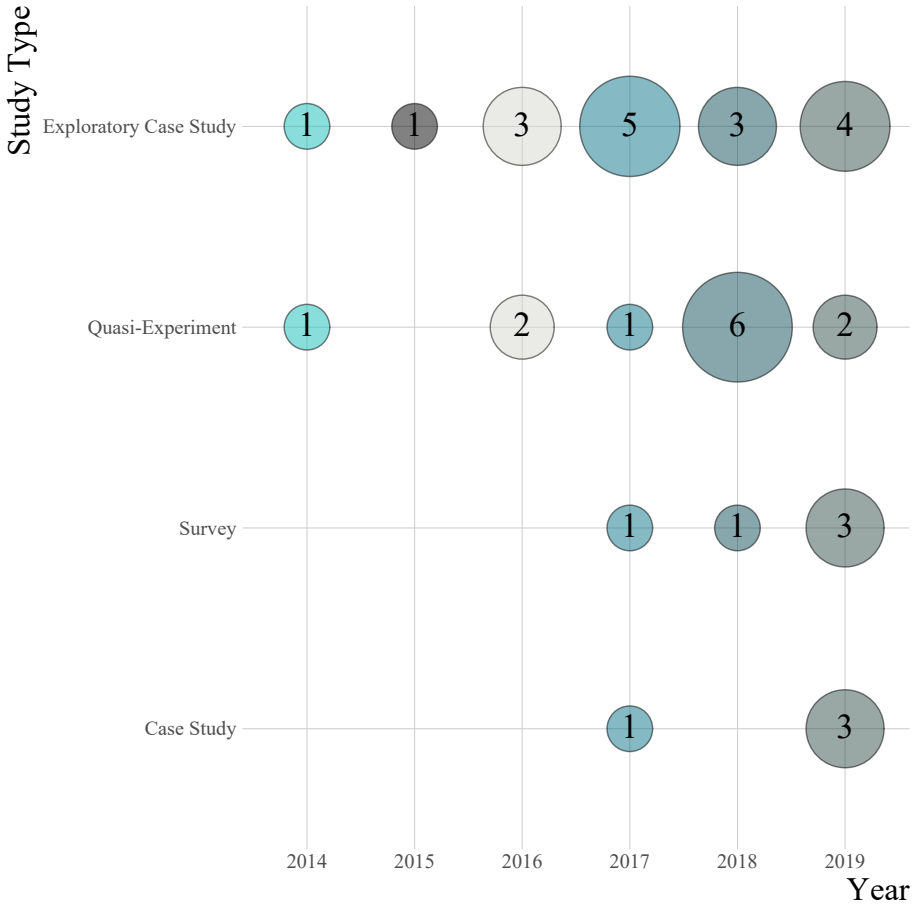

Figure 2.6: Frequencies of Study Types per Year

met in software development organizations. Experiments where treatments are applied to some factors in order to later evaluate the outcomes are almost non-existent in real live scenarios. This may reveal that, due to revenue generation pressure, costs control and/or time restrictions, organizations are not willing to spend time and resources to test and experiment novel approaches on analytics even when they promise potential benefits.

RQ1. Summary

i) Controlled Experiment studies look neglected by the community.
ii) 84.3% (27/32) of works pertain to only one study type (Table B.6).
iii) Evidences suggest an increasing trend in the publications quality.

RQ2. What are the main data sources used for software development related studies?

The top four data sources: Github Repositories, Google Play Store, Git Repositories and BugZilla combined are the data sources for more than 80% of the studies. This was somehow expected as they are generally under the public domain and contain the code, issue reports and product compilations of the most used open source projects, which are, very often used in empirical

studies. This provides some evidence that the community is probably studying the most what is possible to study, simply because the datasets are under the public domain.

Interesting to mention is the high number of publications using datasets from App Stores such as Google Play Store. This might be a relevant indicator that the researchers' focus, the profile of the end-user and the developers' characteristics are quickly and fundamentally changing.

Figure 2.7 plots the frequencies of all studies regarding **RQ2**. It is proper to highlight that, from all the data sources used in more than one study, 4 are related with software configuration management systems, 2 with App Stores and each of the remaining 3 with: Bug/Issue Tracking Systems, a Q&A Service and an Online Survey.

RQ2. Summary

i) Code management and bug/issue tracking systems are used frequently.
ii) App Stores, Q&A services, Wikis and Forums are promising sources.
iii) Repositories containing developers' project interactions are scarce.

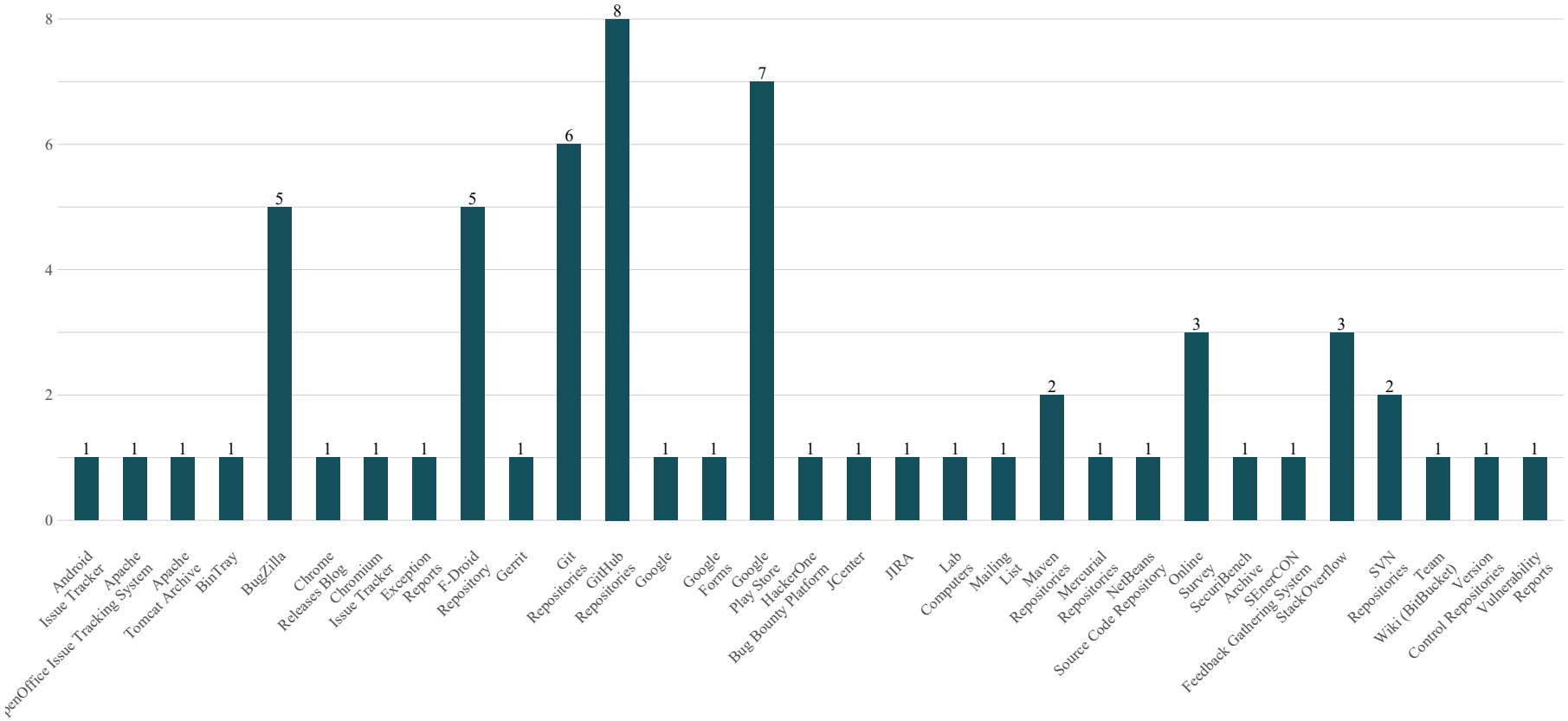

Figure 2.7: Frequencies of studies for Data Sources

RQ3. What type of process/project perspective analysis was conducted?

We found that all the studies were focused on a Post-Mortem approach, meaning the study was not designed to help the product/project managers take any corrective measures on a timely manner to the artifact under study. As such, any insights gathered could only impact future developments. A Post-Mortem approach provides benefits for the next product release or project, but usually, not for the one being studied as it brings no added value when proactive corrective actions are desired.

RQ3. Summary

i) Ineffective approach to improve project under study.
ii) Real-time development operational support is missing.
iii) Worthless approach if project actions recommendation is needed.

RQ4. What are the SDLC activities mostly studied?

According to [95], in Table 2.3 we summarize which activities of the SDLC, are being researched the most. Our findings show that 93.75% and 65.62% of the studies were targeting the Implementation and Maintenance phases, respectively. Regarding Testing, we found only 3 studies, [S01], [S24], [S30], focused on that activity. These results, which confirm that some phases are under-researched, require the attention of practitioners and eventually the opening of new streams of investigation on the SDLC. Software under operation was the focus of 5 studies and those were mainly related with software deployed to App Stores. Figure 2.9 present the statistics about all the activities studied.

Table 2.3: SDLC Activities Findings

Activity	Freq.	Perc.	Ref.
Implementation	30	93.75%	[S01], [S02], [S04], [S06], [S07], [S08], [S09], [S10], [S11], [S12], [S13], [S14], [S15], [S16], [S17], [S18], [S19], [S20], [S21], [S22], [S23], [S24], [S25], [S26], [S27], [S28], [S29], [S30], [S31], [S32]
Maintenance	21	65.62%	[S07], [S08], [S09], [S10], [S11], [S12], [S13], [S14], [S17], [S18], [S20], [S21], [S22], [S23], [S24], [S25], [S26], [S27], [S28], [S29], [S30]
Debugging	6	18.75%	[S07], [S08], [S09], [S10], [S11], [S12]
Operations	5	15.62%	[S03], [S05], [S18], [S20], [S28]
Testing	3	9.38%	[S01], [S24], [S30]

RQ4. Summary

i) Around 93.75% of articles focus on coding/programming activities.
ii) Analytics for Testing tasks appears less than on Debugging practices.
iii) Requirements Engineering and Design activities are not studied.

RQ5. Who were the target stakeholders of these studies?

All the studies targeted the Developers, and 7 were addressing Product Managers concerns. Only 5 publications could bring any value to Testers: [S01], [S24], Educators: [S29], End-Users: [S20] and Requirements Engineers: [S18]. These findings are aligned with the results found in previous SLRs mentioned in section 2.2.1. We are predisposed to think that these results are related with the data sources also identified previously. When the majority of data sources used are product code related, it is somehow plausible that the stakeholder for that study is a developer. On summarizing the data about the individuals that could benefit from each study, we argue that the proper insights are not reaching all those who need support on their daily activities, namely Project Managers, Testers and Requirements Engineers. Figure 2.9 supports our comments by plotting the frequencies of all stakeholders targeted.

RQ5. Summary

i) Developers keep being the main target stakeholder for SDA.
ii) SDA for Testers are less frequent than expected.
iii) High-Level management needs are not being addressed.

RQ6. What are the main mining methods being used?

All articles, as expected, present descriptive statistics about the domain under study. We know that, very often, research starts with just exploratory actions. However, understanding “What happened"is a reduced perspective for what analytics can do for software development. It is also not surprising that the following most frequent methods used are approaches which target the extraction of knowledge, either by correlating factors or by classifying or grouping subjects. Hypothesis testing appears less frequently as one would expect. This may be related with the fact that all studies have, as mentioned earlier, a post-mortem approach and any results obtained are not to be used immediately to perform any corrections in the studied project. If used properly, that is what hypothesis testing may bring in advanced forms of analtyics.

Being software development a process, one would expect to find Process Mining methods often in the assessed studies. Looking deep into the data, we can confirm that it does not hold true, which may reveal that practitioners are studying processes without the proper plethora of methods and tools. Figure 2.8 provide evidences for the most used mining methods.

RQ6. Summary

i) Few studies try to make any predictions.
ii) Hypothesis Testing appear in only 7(21.88%) of the studies.
iii) Only 1 study (3.12%) used Process Mining methods and tools.

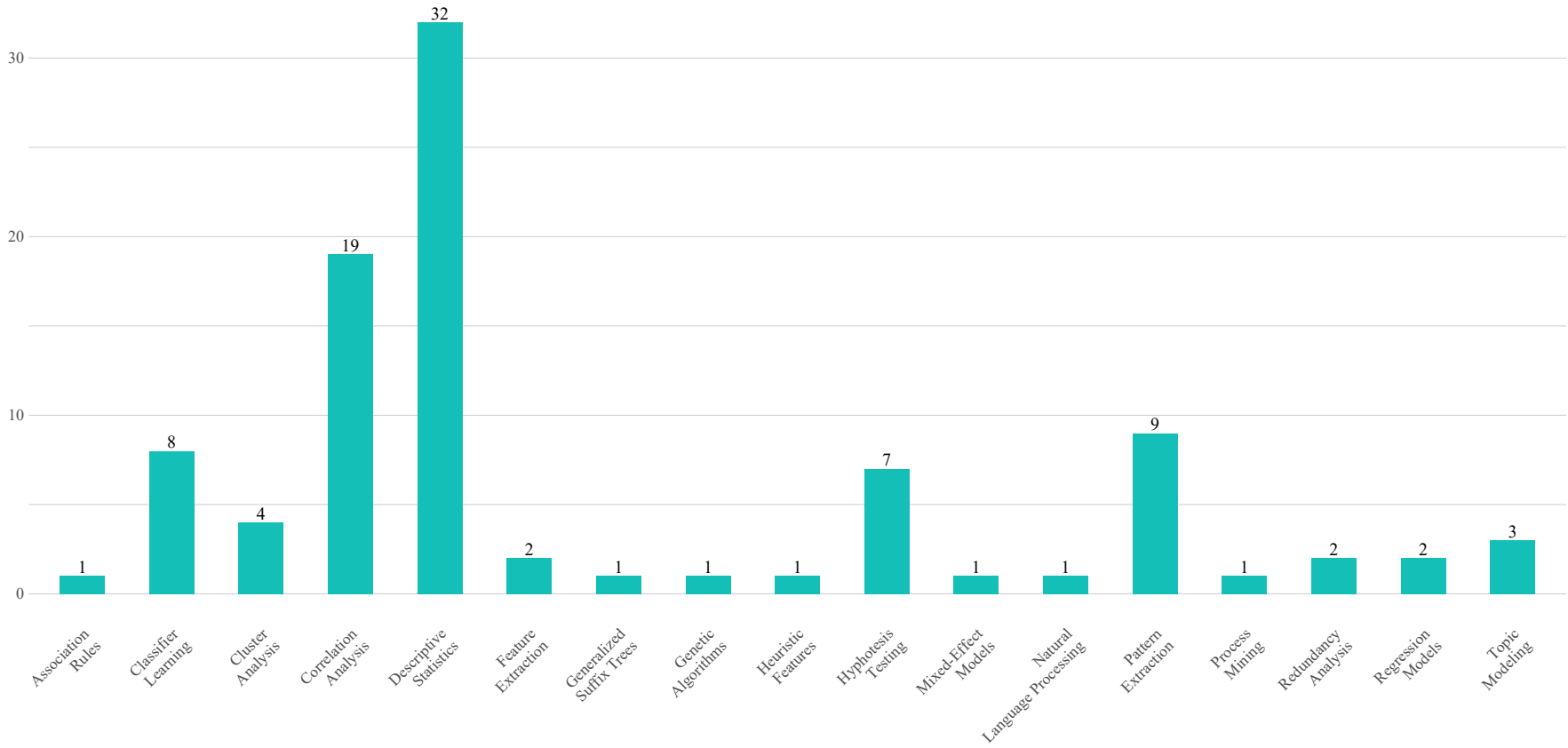

Figure 2.8: Frequencies of studies for Mining Methods

RQ7. Which type/form of analytics was applied?

Following the rationale in **RQ6**, we found all studies used Descriptive and Diagnostics Analytics together. It makes sense that understanding “hat happened"is complemented with “Why it happened". However, this observation is not fully aligned with the results mentioned in previous SLRs, namely in [154]. Although 28.12% of the studies had some sort of prediction as a goal, that is not reflected in the prescriptive domain, where only 1 study, [S30] aims at suggesting stakeholders actions to improve or correct a development activity. Figure 2.9 complements the analysis to this **RQ**.

RQ7. Summary

i) Descriptive and Diagnostics Analytics seems to be found together.
ii) An increasing trend exists in predictive studies (Tables B.2 & B.6).
iii) Management actions recommendation is not a common practice.

RQ8. What were the relevant contributions to the SDLC?

Technical Debt. All the studies had some sort of contribution to the quality dimension of software and no study was found to be classified with **“Absent"** under this realm. With **“Moderate"** contributions we find [S03], [S22], [S23], [S26], [S28], [S31]. Having a **“Strong"** impact we identify [S01], [S02], [S04], [S05], [S06], [S07], [S08], [S09], [S10], [S11], [S12], [S13], [S14], [S15], [S16], [S17], [S18], [S19], [S20], [S24], [S25], [S30], [S32]. Very few studies have **“Weak"** benefits identified.

Time Management. The management of project times looks forgotten since around 65% of the studies provide no contribution under this dimension. We identify only 3 studies, [S15], [S21], [S26] with **“Moderate"** contributions to manage the duration of product/project development. **“Weak"** benefits are present in 8 (25%) studies.

Costs Control. The same scenario happens with the control of costs as only 6 studies, [S01], [S02], [S04], [S08], [S11], [S21], provide contributions to this dimension and they are **“Weak"**.

Risk Assessment. Despite the fact that risk is cross-cut to all other dimensions identified in **RQ8**, we found only one study, [S01], concerned exactly with the risk associated with the security of software. The contribution given was **“Weak"** though.

Security Analysis. Regarding software security implementation and operations, we found very few studies where their main contributions were around this domain. We found 4 studies, [S27], [S29], [S01] and [S30]. Only the latter has a **“Strong"** classification regarding this contribution.

Most of the works focus on the software quality dimension and other features are barely touched by practitioners. Improving or understanding better a project costs, risks and security aspects are contributions rare to find. Only one study, [S1], provides contributions across all the dimensions we assessed and 3 of those 5 dimensions have **"Weak"** contributions. We got 5 studies providing contributions in 3 dimensions and 8 have 2 contributions. The remaining studies contribute to only one dimension. No study was classified as **"Complete"** on any of the contribution areas identified for the SDLC.

RQ8. Summary

i) The software quality dimension consume most research resources.
ii) Time and Costs concerns are not being addressed sufficiently.
iii) Security and Risks matters need extra and aligned effort to evolve.

Figure 2.9, which supports our answers to **RQ1, RQ4, RQ5, RQ7**, plots the frequencies of studies related with the analytics depth, study types, stakeholders and SDLC activities studied.

Figure 2.10 renders the evaluation off all studies across the five dimensions used to answer **RQ8**. As it is clear from the plots, Technical Debt and Time are the dimensions mostly studied. A list of all studies with a short summary, their context, methods and results are presented in B.2. A holistic perspective of all the **RQs** findings is presented in B.5.

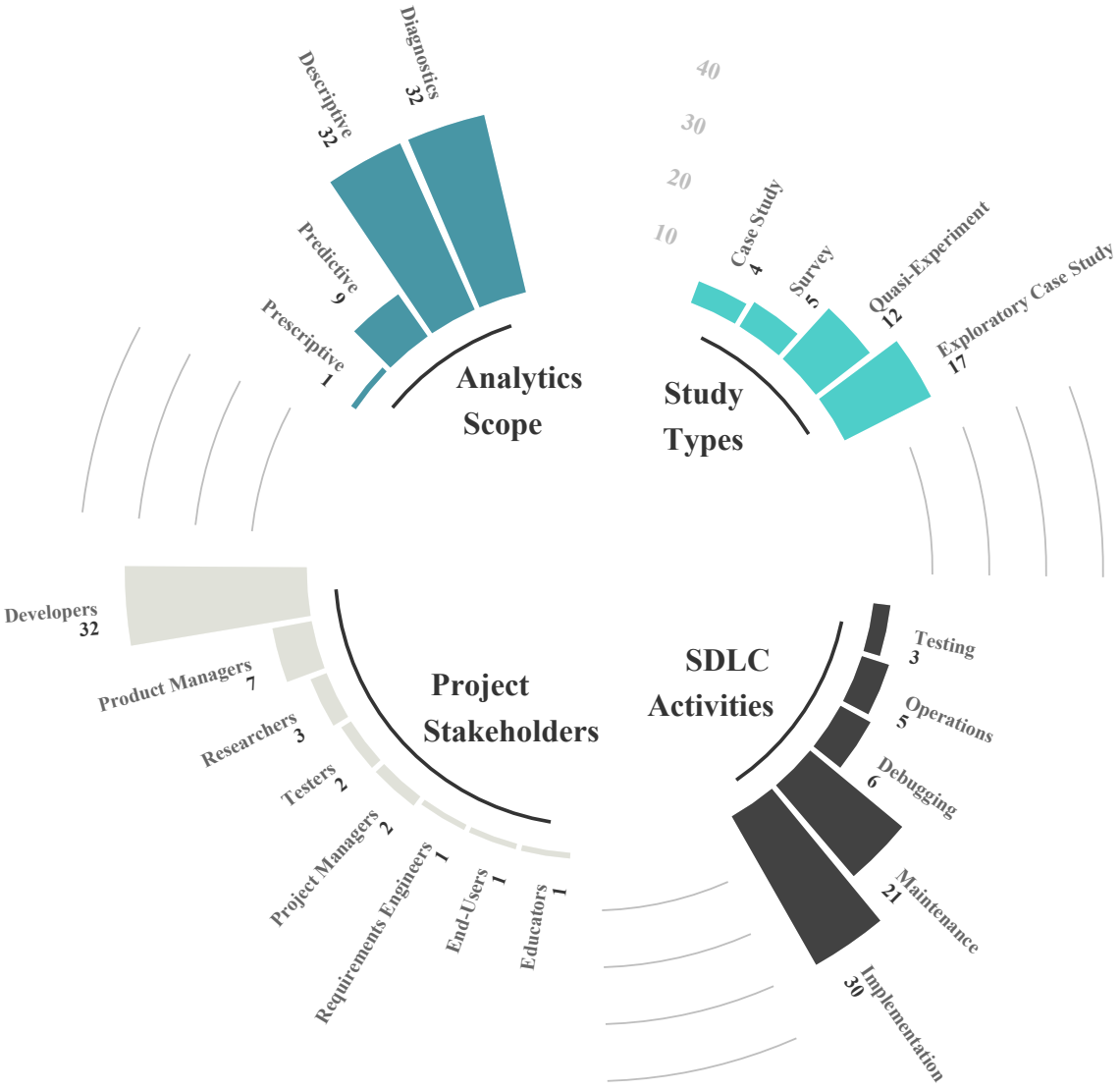

Figure 2.9: Frequencies of studies combining multiple RQs in the SLR

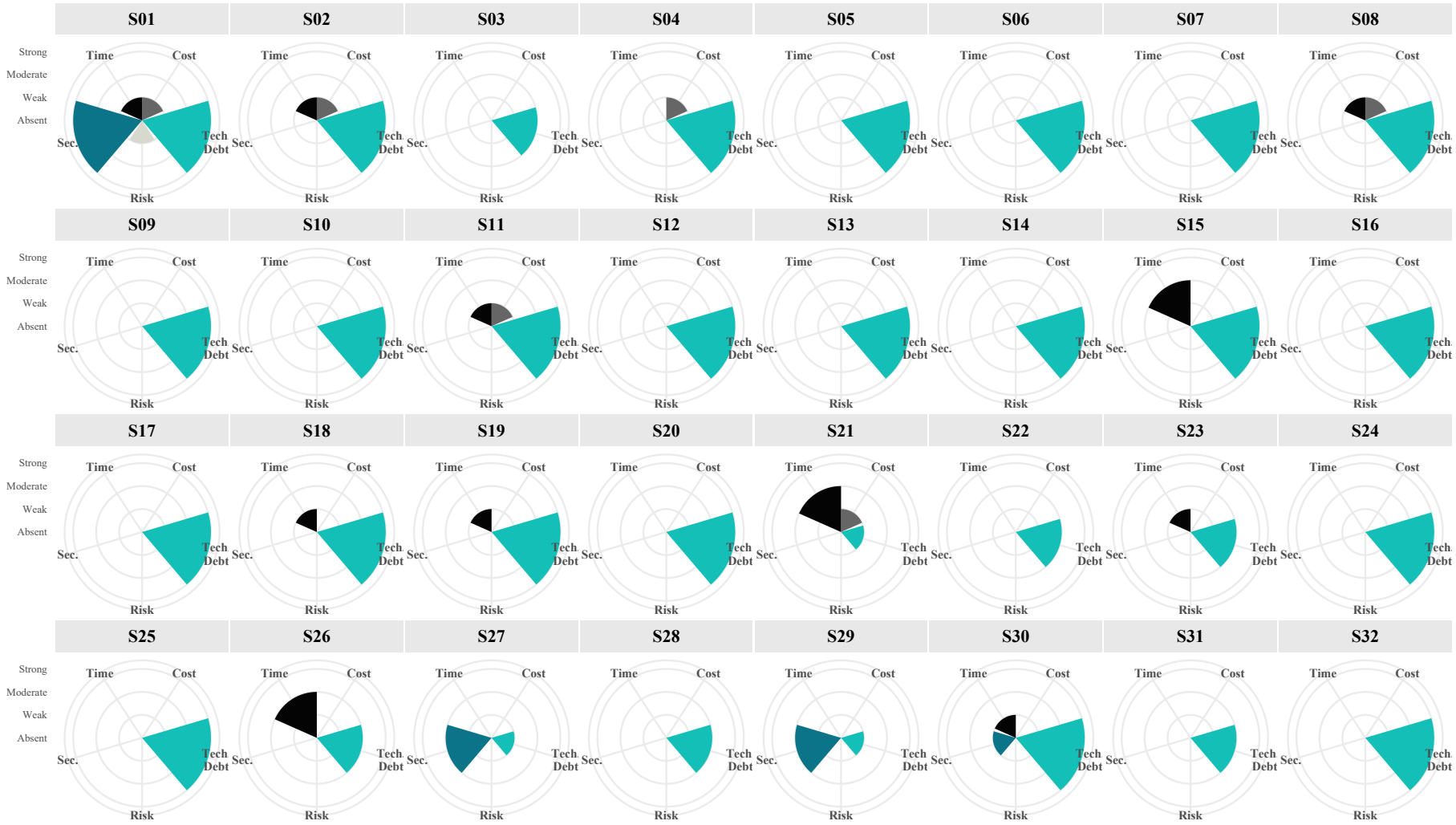

Figure 2.10: Classification combining all 5 contribution dimensions to SDLC(RQ8)

2.4.3 Threats to Validity

The following types of validity issues were considered when interpreting the results from this review.

2.4.3.1 Construct Validity

The studies identified from the systematic review were accumulated from multiple literature databases covering relevant journals, proceedings and books. One possible threat is bias in the selection of publications. This is addressed through specifying a research protocol that defines the objectives of the study, the research questions, the search strategy and search strings used. Inclusion, exclusion criteria and blueprint for data extraction and quality assessment complements the approach to mitigate such bias.

Although supported by important literature under the software engineering domain, we followed a self-defined classification criteria for some **RQs**, specifically for **RQ8**. This method is somehow subjective as someone else might have chosen any other classification categories.

Our dataset contains studies published until mid July, 2019. There are some evidences pointing to an increasing trend in the publishing of studies in the SDA domain, however, articles published in the second-half of 2019 which might also had good quality, were not included in this review. We excluded works where their goal was only to propose new algorithms and/or methods to analyze software development. Some of these studies had also validation experiments, however, their conclusions were related with the quality of the methods and not with any benefits potentially provided by them for the software development process. Some of those studies had also interesting approaches to improve analytics as a practice, however, they are not present in this review.

2.4.3.2 Internal Validity

One possible threat is the selection bias and we addressed it during the selection step of the review, i.e. the studies included in this review were identified through a thorough selection process which comprises of multiple stages. We were aiming to find high quality studies, therefore, a quality assessment was introduced and a final selection for studies ranking above the third quartile was conducted. This approach may have excluded studies with very important contributions on any of the dimensions we assessed in **RQ8** or other dimensions not covered by this review. We used an ordinal/categorical taxonomy to assess the studies regarding **RQ8**. This classification method is still subjective and depends on the authors' contents interpretation.

2.4.3.3 External Validity

There may exist other valid studies on other digital libraries which we did not search. However, we tried to reduce this limitation by exploiting the most relevant software engineering literature repositories. Studies written not in English were excluded which can also have excluded important work which otherwise would have been also mentioned.

2.4.3.4 Conclusion Validity

There may be bias in the data extraction phase, however, this was addressed through defining a data extraction form to ensure consistent extraction of proper data to answer the research questions. We should also refer that, the findings and further comments are based on this extracted data. Despite the fact that high levels of validation were applied in the statistics computation of this study, there is always a small chance that any figures might be inaccurate. For this reason, we publish our final dataset to enable replication and thus allowing for further validation.

2.5 Conclusions

We conducted a Systematic Literature Review on SDA in practice, covering a time span between 2010 and mid 2019. From an initial population of 2,769 papers, we kept 32 of them for appraisal.

It targeted eight specific aspects related with the goals, sources, methods used and contributions provided in certain areas of the SDLC. Our goal was to extract the most relevant dimensions associated with software development practices and highlight where and what were the potential contributions given by those works to the SDLC. From a quality assessment perspective, our aim was also to classify the benefits provided by those studies to significant software development concerns such as: quality/technical debt, time, costs, risks and security, therefore, a taxonomy was created to evaluate them.

Source code repositories, such as GitHub and Git, and App stores like Google Play Store are the most common data sources used in SDA. The most frequent study type is Experimental Case Study and the most common stakeholder of those studies are the developers. Product and Project Managers are also often present but in a less prevalence that one would expect. Mining methods have evolved in the last few years and that is reflected in the long list we got. Not surprisingly we found that descriptive statistics are the most usual method followed by correlation analysis. Being software development an important process on every organization, it was remarkable to find that process mining is present in only one study. Most contributions for the SDLC were given in the software quality dimension. Time Management and Costs Control were softly debated. The Security Analysis aspect, although with a weak evidence, leads us to think that this is an increasing topic of concern for organizations and researchers. However, we were expecting more work in this area because security is persistently in the forefront of concerns in the field. Risk Management contributions are almost non existent in the literature we evaluated.

Our analysis highlighted a number of limitations and shortcomings on the SDA practice and bring the focus to open issues that need to be addressed by future research. It is our understanding, that our work may provide a baseline for conducting future research and the findings presented here will lead to higher quality research in this domain.

2.5.1 Call for Action

As a final remark and to trigger a call for action in the research community, the following issues should be addressed:

- **Repository Diversity.** We suggest researchers to explore different and non trivial software development related repositories, such as the IDE or other archives containing development events(eg: decisions, fine grain actions executed, etc). More and distinct datasets are expected to expand the analytics coverage on software development.
- **Keep working on the needs of different stakeholders.** We have evidences that the practitioners who benefit most from the current SDA studies are the developers and many other profiles are left behind. We suggest to increase the focus on the real needs of requirements engineers, project, product and portfolio managers and higher level executives.
- **Aim at Software Development Operational Support.** No studies were found providing clear evidences that the outcome of that study could benefit on a timely manner the ongoing project or product versions. If organizations want to focus effectively on detecting, predicting and recommending corrective actions on a timely manner, meaning, any insights gathered will have impact on current project and not solely on the next project or product version, researchers and practitioners should focus on designing advanced tools and methods to address software development operational support.
- **Software Development Process Mining.** Despite the fact that Process Mining is now a mature topic, almost no software process related studies uses it. We suggest its techniques and tools, to study deeper the interaction of software development stakeholders and to complement the effectiveness of assessing certain software development tasks, such as, project effort prediction, code maintenance activities and/or bug detection methods.
- **Project Time and Costs.** We suggest more and deeper studies covering the Time and Costs of software projects. These are dimensions barely addressed by the studies we evaluated. The aforementioned topics are extremely relevant to forecast resource allocation for future projects.
- **Address Security and Risks holistically.** Due to the unceasing digital transformation present nowadays in the society, the security of information systems will be even more critical to any organization. We now have robust methods to assess security vulnerabilities in software code. However, very little is known about the developers behaviour during the Implementation and Maintenance phases, just to name a few. Even if, in the last years, security in general became quickly a pertinent topic, the security around development processes and the involved resources are still not clearly addressed. This is a topic with increasing relevance and deserves the rapid and focused attention from the practitioners.
- **Blockchain.** One of the most interesting, promising and relevant technological contributions to the society, was created roughly ten years ago - the birth of *bitcoin* [153]. Although *bitcoin* is an implementation of electronic money, it is supported by something very powerful, which can be used for many other use cases, called - *blockchain* [197]. The *blockchain*

is a mechanism which is able to keep a book of data records immutable and distributed across a multi-node network of servers. It is virtually indestructible since it has no central authority controlling it and preserves data integrity by potentially not allowing rollback on any past transactions. Additionally, if required, it guarantees that only the data owners are able to view or change their personal records and yet permit third-parties to be granted view only privileges to a selected dataset. This technology may be used embedded in SDA to anonymize and grant privacy to organizations sharing data without spoil the context associated with the development process under study.

Part II.

Software Process Immersion

PART I : FUNDAMENTALS

Introduction
Chapter 1

State-of-the-Art
Chapter 2

PART II : SOFTWARE PROCESS IMMERSION

Assessing Teams' Efficiency
Chapter 3

Unveiling Process Insights
Chapter 4

PART III : TOWARDS THE PRESCRIPTIVE COMMITMENT

Practices and Fingerprints
Chapter 5

PART IV : CONCLUSION

Conclusions and Future Work
Chapter 6

This Part present initial work done to experiment Process Mining methods and tools as a way to mine the Software Development Process.

Chapter 3.

Assessing Teams' Efficiency

Contents

This chapter presents a preliminary study about the usage of process mining techniques, evaluates the quality of process models discovered from mining software development sessions during a code smells detection activity and assess teams' efficiency on the tasks performed.

Companion Soundtrack: Time - Hans Zimmer (Inception Soundtrack)

"If you torture the data long enough, it will confess."
—Ronald Coase(1910-2013)[1]

3.1 Introduction

Inaccurate planning and/or project plan deviations cause substantial financial losses on software development projects [145]. Further, constant inaccuracies and losses may degrade the reputation of development teams as they become perceived as non-compliant to organizational plans and budget forecasts.

Critical success factors have always been at the forefront of the research related with software development projects [7, 29, 45, 195]. The existence of a vast literature about this topic, either about successes [145] or failures [62, 132], reveals the concerns and doubts that still haunt software development practitioners regarding the efficiency and effectiveness of their own projects.

It is frequently suggested that software projects can be assessed across four perspectives: quality, scope, time and cost [45], which are related with the planning and execution of the project's main activities. Each perspective has its own critical success and failure factors, that can be grouped into five different dimensions: organizational, people, process, technical, and project [195]. In this work, we will be mainly concerned with the effect of the human factor in process variability.

To start a software development project from scratch is a complex activity on its own [95], specially in the absence of a formalized process or methodology [160] that acts as a referential. Evidences found suggest that in addition to the initial project planning, the way people are organized, the tools they use and the processes they follow are key features for the success or failure of any software project [160]. As for software development, although prescribed process models may exist, projects often do not comply with them, both because each developer or team usually has some freedom to interpret the process and because its compliance is not verified on the run, since it is mainly intangible. As a result, it has been noted that process executions (i.e. projects) often deviate from what was planned [120]. In this work we bring further evidence that the human factor is a very important source of process variability and the latter will have an impact on process efficiency and effectiveness.

To understand how the process was actually performed by its practitioners, we used process mining techniques. Our approach, initially proposed in [36], captures events due to practitioners activities executed in the IDE, as well as records which artifacts were used and when, plus additional details on the ecosystem of components supporting the process. This new perspective on software development analytics, that uses process mining, allows the discovery of the actual processes practitioners are following, as well as deviations from those they were supposed to comply to, without the complexity and workload of collecting and merging information from different information systems, such as, source code systems, configuration management repositories or bug tracking tools. As we will show later in this article, we were able to identify

[1] British economist, author and Nobel Prize in Economics in 1991.

the most and less efficient teams, and the ones that drifted less from the same process when executed by an expert.

3.2 Background

3.2.1 Software Development and the IDE

Nowadays, most software practitioners develop their work upon an IDE, such as `Eclipse`, `IntelliJ IDEA`, `Netbeans` or `Visual Studio Code`. To a greater or lesser extent, those IDEs support different software development life cycle activities, such as requirements elicitation, producing analysis and design models, programming, testing, configuration management, dependencies management or continuous integration. In this work we will consider `Eclipse`, which owes its wide adoption to the vast plethora of plugins available in its marketplace. `Eclipse` distributions are customized for specific users / purposes, such as for modellers, programmers, testers, integrators or language engineers. Herein, we will consider the standard distribution, which is particularly suited to programmers.

An IDE, in addition to the artifacts it handles, contains metadata about the developers' activities that may reveal the reasons why some individuals and teams are more efficient than others. Moreover, it may have hidden in its usage, parts of the logic why some projects are successful and others fail. Those development activities can be identified by mining the large amount of events created during the execution of the IDE core components and the installed plugins.

3.2.2 Process Mining within the IDE

Process Mining is now a mature discipline with validated techniques producing accurate outcomes on several business domains [165, 208]. A process mining project, if best practices are followed [126], should use goals and event logs as inputs, and produces actions to implement as outputs. The goals may consist of improving some performance indicators, such as time, risks and costs associated to a specific process, or simply to maximize a service level. Actions may be the redesign of a specific project, adjust a current process or, if there is a fluctuation in case volume, one may want to include more resources.

Our short-term goal, whose fulfillment we will describe in this work, was to assess teams' efficiency by mining the software development process flow and variability that occurs due to the human factor. Our medium-term goal is to provide operational support to software developers, systematically and continuously using current event data to recommend the best activity, adequate resource or action to execute now or in the future. In both cases we will take as input the events emerging from using the IDE. Those events convey a spaghetti-like process [206] mainly because there is a very large number of possible commands/tasks to execute within any IDE that will grow exponentially with the number of installed plugins and, as a consequence, so grows the potential complexity of any mined process.

3.2.3 Related Work

This work is in the crossroads of software development practices and process mining techniques. Much have been said in literature about software development processes [72, 133] and process mining separately [127]. However, elaborating about works combining these two disciplines requires a careful approach, mainly because their intersection is vague in some cases and not fully explained in others. Going back almost a decade, [165] have mined software repositories to extract knowledge about the underlying software processes, and [180, 181] have learned about user behavior from software at runtime. Recently, [96] was able to extract events from `Eclipse` and have discovered, using a process mining tool, basic developers' workflows. Some statistics were computed based on the activities executed and artifacts edited. In [18], the authors have extracted development activities from non-instrumented applications and used machine learning algorithms to infer a set of basic development tasks, but no process mining techniques were used to discover any pattern of application usage. [52] used a semi-automatic approach for analyzing a large dataset of IDE interactions by using cluster analysis [52] to extract usage smells. More recently, [119] used process mining to gain knowledge on software in operation by analyzing the hierarchical events produced by application calls(eg: execution of methods within classes) at runtime. The studies mentioned above, extracted data from several different sources and have used a multitude of statistics methods, machine learning and process mining techniques. However, to the best of our knowledge, none of these works combine data from the IDE utilization with process mining methods with the aim of measuring individuals or teams efficiency. Even in the case of [96], where the approach is similar to ours, nothing was done related to conformance checking on the processes followed by developers, as there was no existing reference model to compare with. Our work introduces a valid approach for this purpose, and bring a new perspective to software development analytics by filling this gap.

3.3 Study Setup

We analyzed several teams performing independently the same well-defined task on software quality assurance. To block additional confounding factors in our analysis, all teams had similar backgrounds and performed the same task upon the same software system. To provide authenticity, the task targeted a real-world (large) open-source Java system, the `Jasml (Java Assembling Language)`[2].

To understand what happened in each team, we mined the corresponding process model based on its events (process discovery phase). Then, we compared each discovered process with a reference model (process conformance checking phase), to assess the overall similarities and processes' quality.

3.3.1 Research Questions

The following research questions emerged from our previously stated research goals:

[2] http://jasml.sourceforge.net/

- **RQ1:** To what extent can process mining discover accurate models representing developers' behavior?
- **RQ2:** Can we assess the efficiency of software development teams by using process mining techniques ?
- **RQ3:** The assessment of teams' proficiency, performed by a process expert, is reflected in the quality of the produced models?

3.3.2 Experimental Setup

3.3.2.1 Subjects

Subjects were finalists (3rd year) of a BSc degree on computer science at the Iscte university, attending a compulsory software engineering course. By this time they had been trained across the same set of almost 30 courses and therefore had similar backgrounds. They worked in teams up to 4 members each and were requested to complete a code-smells detection assignment, aiming at identifying refactoring opportunities, using the `JDeodorant` tool[3]. This tool allowed the detection of four different types of code smells: `Long Method`, `God Class`, `Feature Envy` and `Type Checking` [20]. Once they have detected the occurrences of those code smells, they were required to apply `JDeodorant`'s automatic refactoring features to the critical ones.

3.3.2.2 Data Collection Instrument

The `Eclipse` IDE has an internal event bus accessed by the interface `IEventBroker`[4] which is instantiated once the application starts. It contains a publishing service to put data in the bus, whilst the subscriber service reads what's in that bus. This allows a subscriber to read all or part of the events being managed within the IDE. Using this feature we developed an `Eclipse` plugin[5] capable of listening to the actions developers were executing. Before the experiment, the plugin was installed on each subject work environment, and later, all received a unique *username/key* pair as credentials. This method was useful to unlock all the plugin features and allowed us to identify each subject and the corresponding team.

3.3.2.3 Collected Data

A sample event instance collected with our plugin is represented in listing 3.1 in `JavaScript Object Notation (JSON)` format. The field tags are self explanatory.

[3] https://marketplace.eclipse.org/content/jdeodorant
[4] https://wiki.eclipse.org/Eclipse4/RCP/Event_Model
[5] https://github.com/jcaldeir/iscte-analytics-plugins-repository

Listing 3.1: Sample Eclipse Event Instance

```
{
"team" : "T-01",
"session" : "a5d63j-jdi3-ikd912",
"timestamp_begin" : "2018-05-07 16:53:52.144",
"timestamp_end" : "2018-05-07 16:54:04.468",
"fullname" : "Ana Sample",
"username" : "ana",
"workspacename" : "Workspace1",
"projectname" : "/jgrapht-core",
"filename" : "/jgrapht-core/AncestorTest.java",
"extension" : "java",
"categoryName": "Eclipse Editor",
"commandName": "File Editing",
"categoryID": "org.eclipse.ui.internal.EditorReference",
"commandID": "iscte.plugin.eclipse.commands.file.edit",
"platform_branch": "Eclipse Oxygen",
"platform_version": "4.7.3.M20180330-0640",
"java": "1.8.0_171-b11",
....
}
```

3.3.2.4 Data Storage

Collected data was stored locally in a Comma Separated Values (CSV) file. Whenever Internet connection was available, the same data was stored in the cloud[6]. This storage replication allowed offline and online collection. The final dataset, combining the two different sources, was then loaded into a MySQL database table where the username and event timestamps that formed the table's unique key were used for merging duplicated data. Figure 3.1 presents a schema of the data collection workflow.

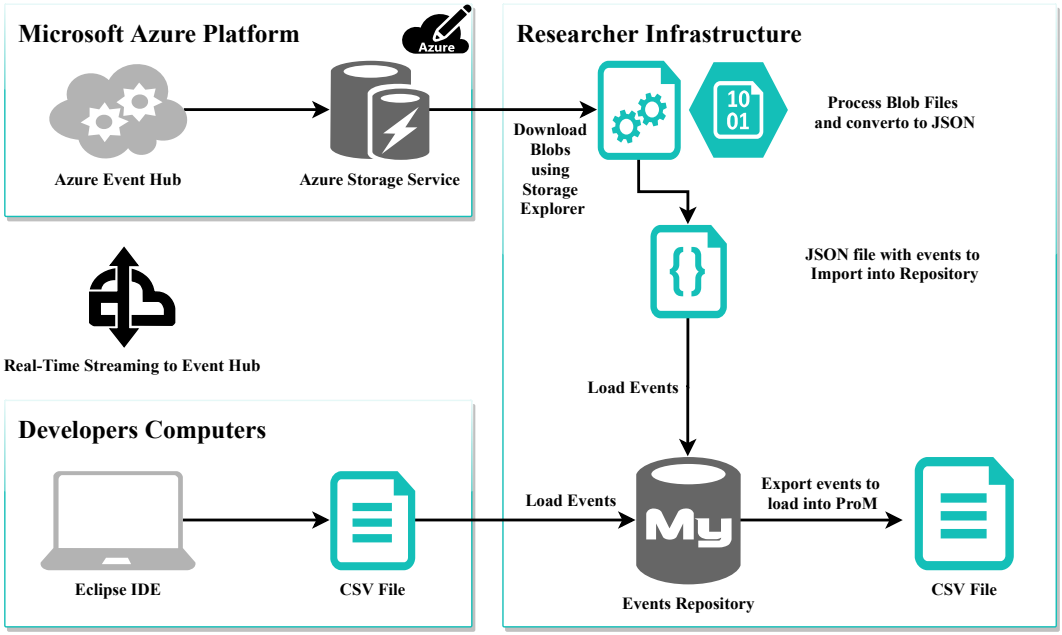

Figure 3.1: Experiment Data Collection Workflow

[6]https://azure.microsoft.com/en-us/services/event-hubs/

3.3.2.5 Data Preparation

When the software quality task ended, all events stored in the database were converted to the IEEE eXtensible Event Stream (XES) standard format [80] and imported into ProM process mining tool[7]. The following event properties were mapped when converting to XES format:

- *team* was used as CaseID since we were interested to look into process instances of teams, not of individual programmers.
- Properties *categoryName* and *commandName* forming a hierarchical structure were used as the activity in the process.
- The *timestamp_begin* and *timestamp_end* were both used as activity timestamps.
- Other properties were used as a resource in the process.

3.3.2.6 Data Demographics

As previously mentioned, we only analyzed data collected on the same software system, to block confounding factors. The chosen system was `Jasml (Java Assembling Language)`[8]. s

Table 3.1: Collected Events Statistics

Team	TM	UCC	UCA	UEA	PE (#/%)	GE (#/%)	TE (#)
T-43	4	10	38	39	790 / 85.13%	138 / 14.87%	928
T-41	2	10	37	40	615 / 77.75%	176 / 22.25%	791
T-02	3	12	41	24	552 / 74.80%	186 / 25.20%	738
T-26	2	8	28	22	360 / 77.25%	106 / 22.75%	466
T-23	1	9	23	22	276 / 93.24%	20 / 6.76%	296
T-21	1	9	27	23	272 / 77.71%	78 / 22.29%	350
T-24	1	8	26	13	181 / 89.60%	21 / 10.40%	202
T-01	4	13	45	16	105 / 29.49%	251 / 70.51%	356
REF.	1	4	12	20	134 / 97.10%	4 / 2.90%	138

TM - Team members, **UCC** - Unique Command Categories
UCA - Unique Command Actions, **UEA** - Unique Edited Artifacts
PE - Project related Events, **GE** - Generic Eclipse Events, **TE** - Total events

The plugin collected two types of events: events within a project context**(PE)** and generic events**(GE)** at the `Eclipse` global context. The former summarizes events for which we have associated project and file names. This information expresses actions done by each developer in the project where `JDeodorant` features, such as, detecting a `God Class`, `Long Method`, `File Open`, `File Edit`, `Refactoring`, `Delete Resources`, were applied. The latter represents events captured from `Eclipse` command actions not associated with any project (e.g. `Update Eclipse Software`, `Install New Software`, `Open Eclipse View Task List`, etc).

We present their statistics in Table 3.1. Project events should be seen as fundamental events for the task programmers were requested to execute, and, in a certain way represent the focus their are putting into that work. Generic events are seen as collateral actions not

[7] Version 6.8, available at http://www.promtools.org
[8] http://jasml.sourceforge.net/

mandatory for the task in hand, but that programmers may need or want to execute to prepare their environment. These generic events somehow convey a lack of focus on the task developers were supposed to execute.

The REFERENCE(also identified as REF.) team, corresponds to the professor that proposed the task itself. Being the main expert, he executed it in one of the most efficient ways. The full dataset, that includes data on all teams with fine grained details that is not addressed in this work, is publicly available.[9]

3.3.3 Data Analysis

3.3.3.1 Context

Several approaches have been proposed to evaluate the quality of discovered process models. Software quality metrics were mapped to process metrics in [209]. Groups of metrics were also used in [178, 179] to evaluate several dimensions in a process model and, more recently, artifacts were created to support process quality evaluation and perform process variants comparisons [27, 119]. All of these fit within the well defined [206] and generally accepted four dimensions to assess the quality of a model: *fitness*, *precision*, *simplicity* and *generalization*.

3.3.3.2 Process Discovery

Several well known algorithms exist to discover process models, such as, the α-algorithm, the heuristics, genetic and fuzzy miner. However, our need to discover and visualize the processes in multiple ways lead us to choose the ProM's StateChart Workbench plugin [119]. This plugin, besides supporting process model discovery using multiple hierarchies and classifiers, also allows to visualize the model as a Sequence Diagram and use notations such as Petri Nets and Process Trees. This plugin is particularly suitable for mining software logs, where an event structure is supposed to exist, but it also supports mining of other so-called generic logs.

Events collected from software in operation (e.g. Java programs) reveals the presence of a hierarchical structure, where methods reside within classes, and classes within packages [118]. The same applies to IDE usage actions, since identified menu options and executed commands belong to a specific category of command options built-in the Eclipse framework. Supported by this evidence, we used the Software log Hierarchical discovery method with a Structured Names heuristic, to discover the models based on the fact that the events were using a *category*|*command* structure (e.g. Eclipse Editor|File Open). Several perspectives can be used to discover and analyze a business process and the most commonly used are: Control-Flow, Organizational, Social and Performance. We have mainly focused on the Control-Flow perspective in this experiment. It defines an approach that consists in analyzing how each task/activity follows each other in an event log, and infer a possible model for the behavior captured in the observed process.

[9] doi:10.17632/8dmdwpgdy4.1

3.3.3.3 Process Variant Comparison

Our goal was also to compare the behaviour among the teams involved in the experiment against the "best practice"process, as performed by the expert, and identify the ones with less differences. For this purpose, we used the `Process Comparator` plugin [27], which is a tool that compares a collection of event logs, using a directed flow graph. It uses transition systems to model behavior and to highlight differences. Transition systems are annotated with measurements, and used to compare the behavior in the different variants. The annotations of each variant are compared using statistical significance tests, in order to detect relevant differences.

3.4 Study Results

Control-Flow Perspective.

Figure 3.2 presents team T-26 process variant, showing the code smells detection activities, and the correspondent statistics about the process followed to execute the requested task. It is clear, based on the different levels of blue in the activities performed, that they executed more often the activities related with the code smells detection and correction. We confirm this by observing the `Eclipse Editor | File Editing` activity which was executed more than any other activity.

Globally, our attention went to the evaluation of the Simplicity (or Complexity) of the models discovered. Simplicity allude to the rule that the simplest model that can describe the behavior found in a log, is indeed the best model.

Software artifacts with higher cyclomatic complexity tend to be harder to maintain. It has been claimed that the same rationale is applicable to process models [39]. Based on this, we were looking for the teams with less complexity in their processes. As shown, teams T-26, T-24 and T-41 are the ones with less `Cyclomatic Complexity` (as represented by different levels of green), therefore closer to the complexity of the REFERENCE model. That is also reflected by the number of `Simple` and `Composite States`, and `Activities` discovered in each of those models. Team T-26 modelled behavior was also the one discovered with best precision (45%) among these 3 teams.

On the opposite pole (as represented by different levels of black) with an unique characterization, we have team T-01, with four members, which did not delivered the results of the requested task. Its proficiency was insufficient and careful review of the process revealed this team produced more generic events than project related events, as shown in Table 3.1. From Figure 3.4 we can also learn this team used more unique command actions and respective categories than any other team, and that did not increase the number of edited files, as one would have expected. This leads us to think its members did not understand or follow the process at all, since many of their actions in the IDE apparently were not aligned with the required task. The high values of complexity, activities, number of transitions and composite states metrics observed in Table 3.2 complements this assumption.

We can, therefore, state the following: T-01 was an "expensive"team and the one that presented more risks from a project management perspective. When compared with other teams, this team had a similar process duration (see table 3.4) in executing the task, but did

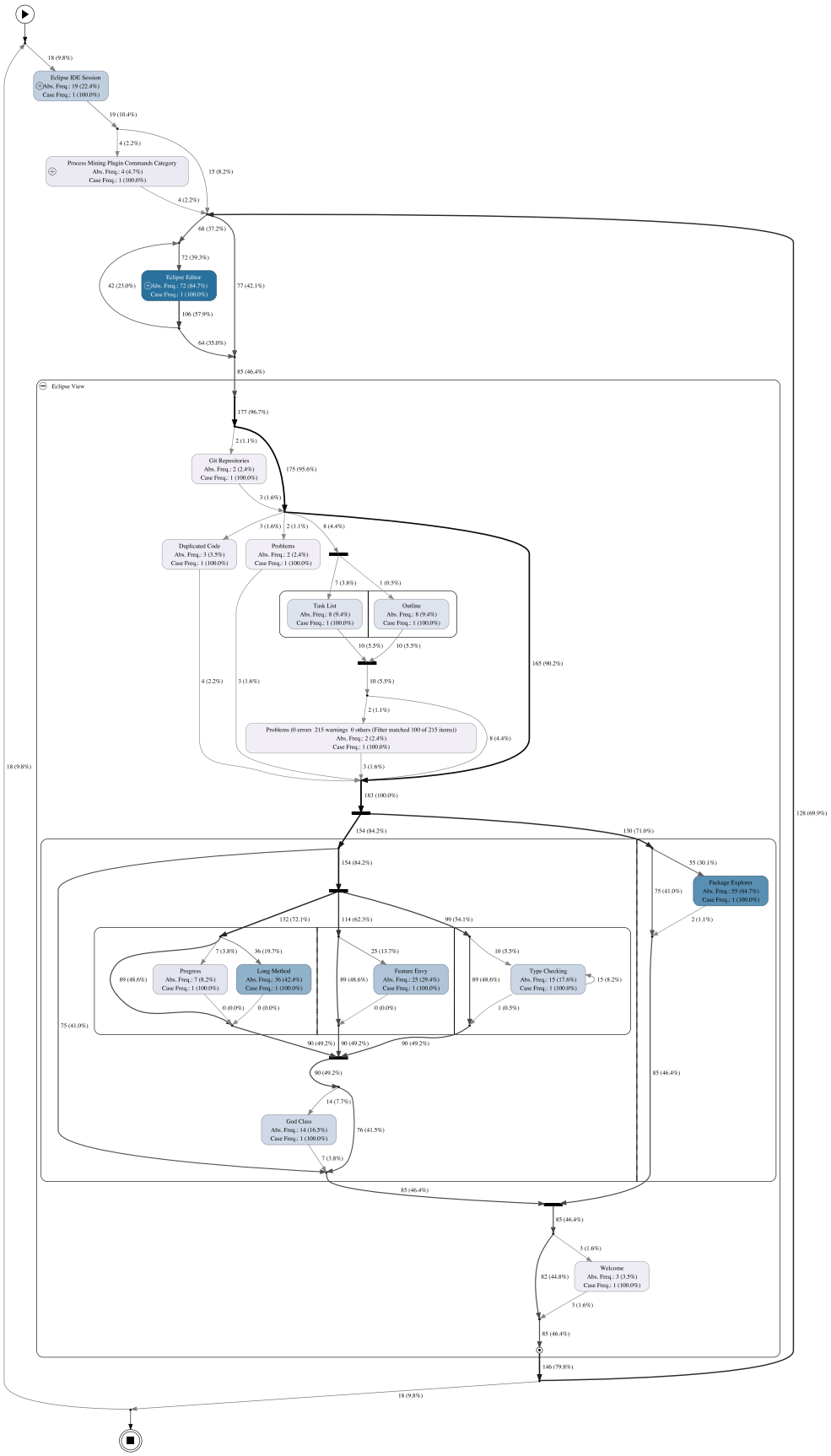

Figure 3.2: Team T-26 Process Variant: Activity Frequency View

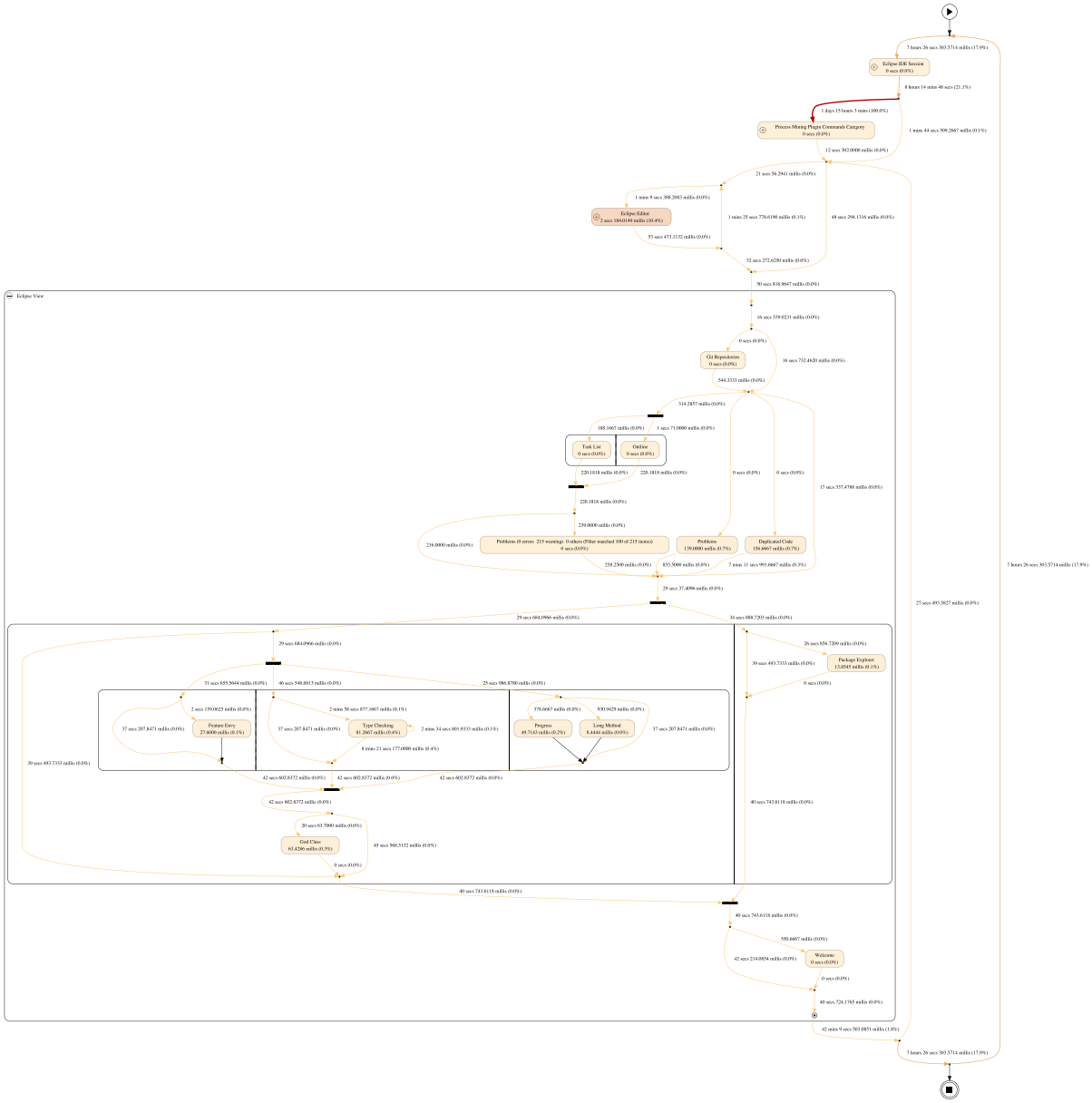

Figure 3.3: Team T-26 Process Variant: Activity Duration View

not deliver the expected outcomes at all. This team was not only non effective, but also showed major inefficiencies in whatever they tried to produce.

An interesting case to study deeper is team T-02 which had a good proficiency in the task, as seen in Table 3.4, but showed high levels of complexity in the model. This means we are dealing with a case where the team was effective, because they achieved the task with success, although without being efficient. This is confirmed by the high number of different commands executed showed in Table 3.1.

We also compared the behaviour between the 3 teams with less complex models against the reference model. The level of Control-Flow differences based on activity frequencies, as calculated with the Process Comparator plugin, is plotted in table 3.3. Team T-24 was the one with less differences when compared with the reference model, followed very closely by T-26. Based on the complexity measurements, control-flow differences and team size, we advocate

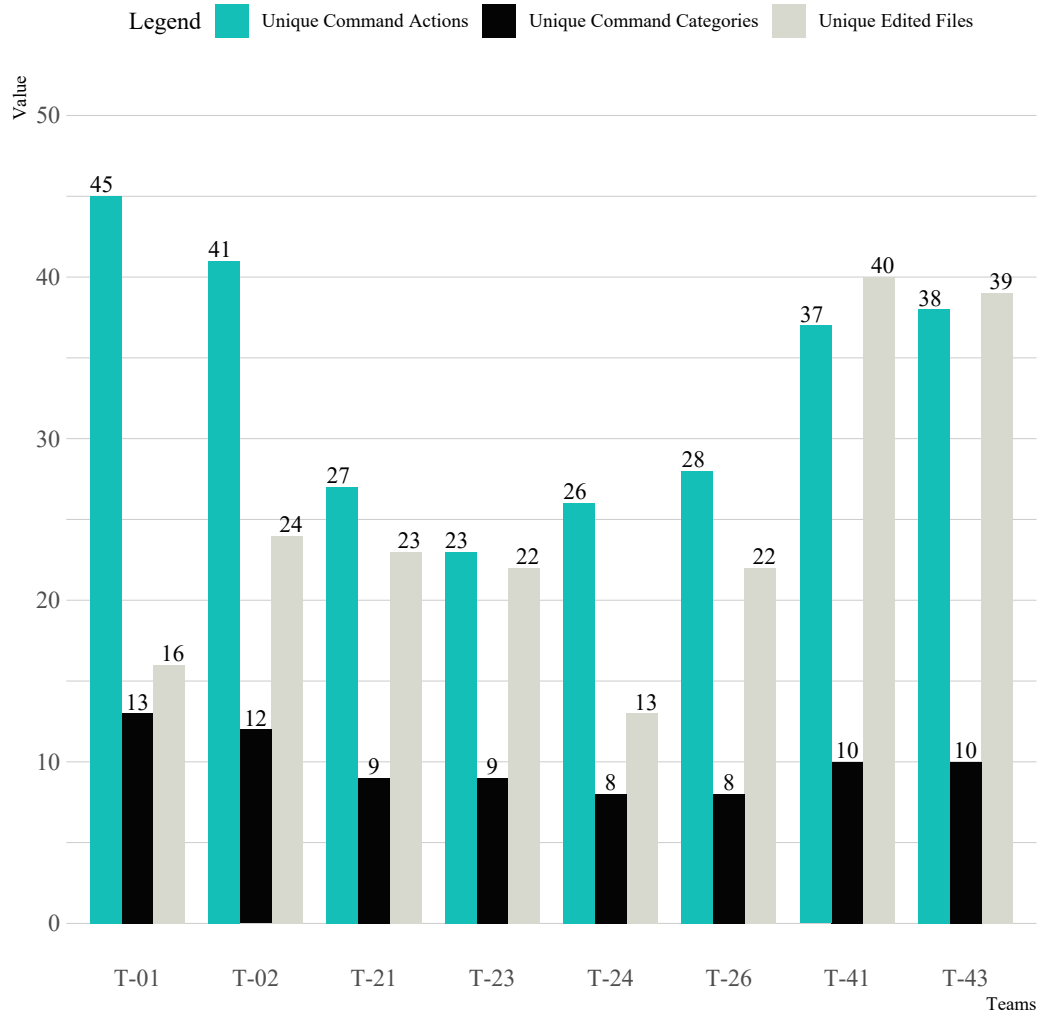

Figure 3.4: Collected Events Statistics

that T-26 had accomplished the task with the best overall efficiency and effectiveness. In fact, that is also reflected in the proficiency mark given by the professor (that acted as the task expert), as shown in table 3.4. This raises a set of other research questions, such as: can process mining be used to assess the proficiency of developers in general, or just for specific kinds of tasks?

Table 3.3: Behavior Differences Comparison

Ref. Log	Team	Control-Flow Differences(%)
REFERENCE	T-41	87.60 %
	T-26	85.11 %
	T-24	85.04 %

Organizational Perspective. Figures 3.5 and 3.6 provide an overview of the handover and collaborative tasks performed by team T-26. From here, one can easily know the peers who have worked more time on the tasks and which were the most common transitions of work

Table 3.2: Models Discovered - Metrics Summarization

Team	F(%)	P(%)	A	HD	SS	CS	T	PCC
T-43	85.8%	39.9%	37	2	93	12	130	35
T-41	74.2%	43.9%	38	2	88	11	121	31
T-02	81.6%	33.8%	47	2	109	11	159	48
T-26	80.1%	45.0%	25	2	60	6	85	23
T-23	79.7%	32.3%	31	2	104	16	141	35
T-21	94.2%	46.5%	36	2	93	12	131	36
T-24	94.4%	35.9%	30	2	74	8	103	27
T-01	91.7%	43.0%	52	2	147	18	209	60
REFERENCE	85.1%	53.7%	16	2	47	6	64	15

F-Fitness, **P**-Precision, **A**-Activities, **HD**-Hierarchical Depth
SS-Simple States, **CS**-Composite States, **T**-Transitions
PCC-Process Cyclomatic Complexity

within this team.

Table 3.4: Assignment Duration

Team	Proficiency	Process Duration
T-43	0	23h:49m
T-41	0.73	18d:3h
T-02	0.75	11d:22h
T-26	0.75	12d:16m
T-23	0.72	12d:13m
T-21	0.02	8d:12h
T-24	0.64	47m:14s
T-01	0	10d:7h
REFERENCE	–	23m:05

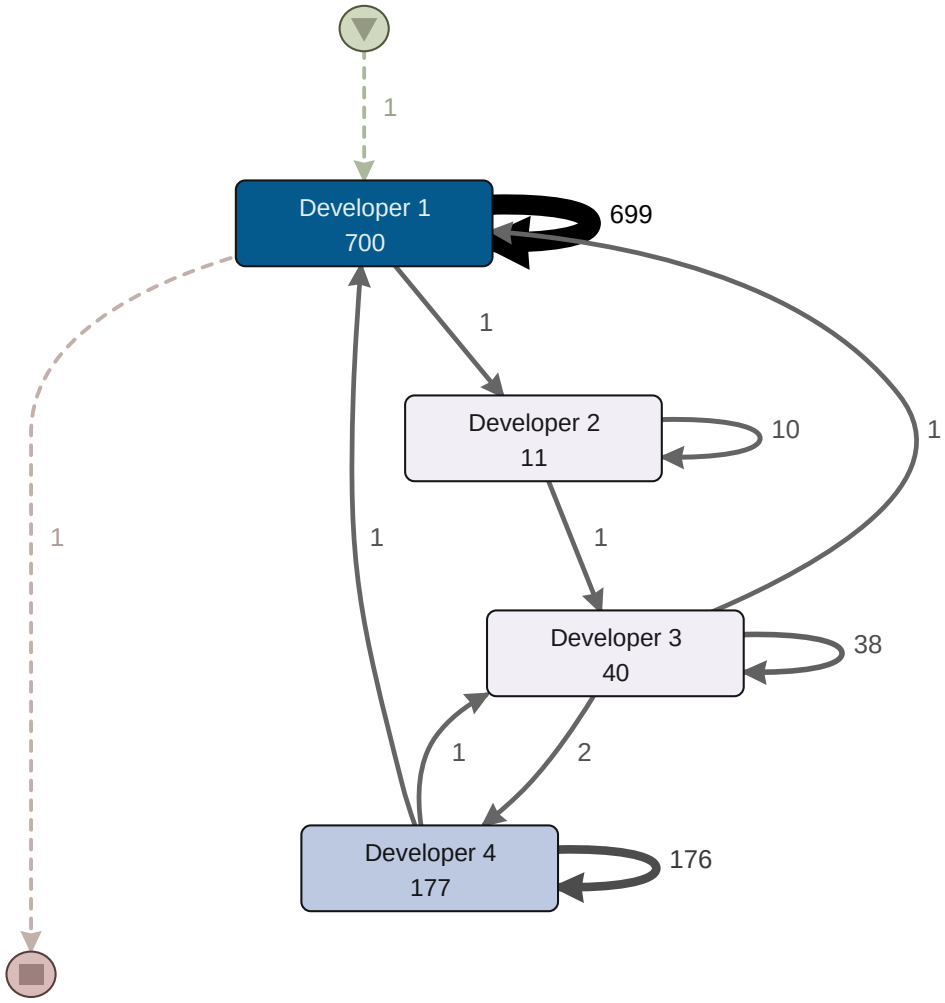

Figure 3.5: Handover between peers - Developers' Activities Frequency

3.5 Threats to Validity

Internal Validity. Since some teams worked in shared laboratories at the university campus, different team members may have used, in the same computer, the same user/key pair to activate the collection plugin. This may be a source of non accuracy in collected data.

Some users have stopped the collection mechanism which makes it impossible to understand what they were doing during that period. We also found that a few teams have made a pause in the task, causing it to express more execution time than what was really needed. The mined processes reflects these times, but indeed, that was idle time. Nevertheless, other reasons may exist for these delays, and therefore, their model is in fact accurate, because it plots what really happened.

External Validity. Since we wanted to block some factors such as the degree of previous experience (background) in the proposed process, and repeat the data collection process in a

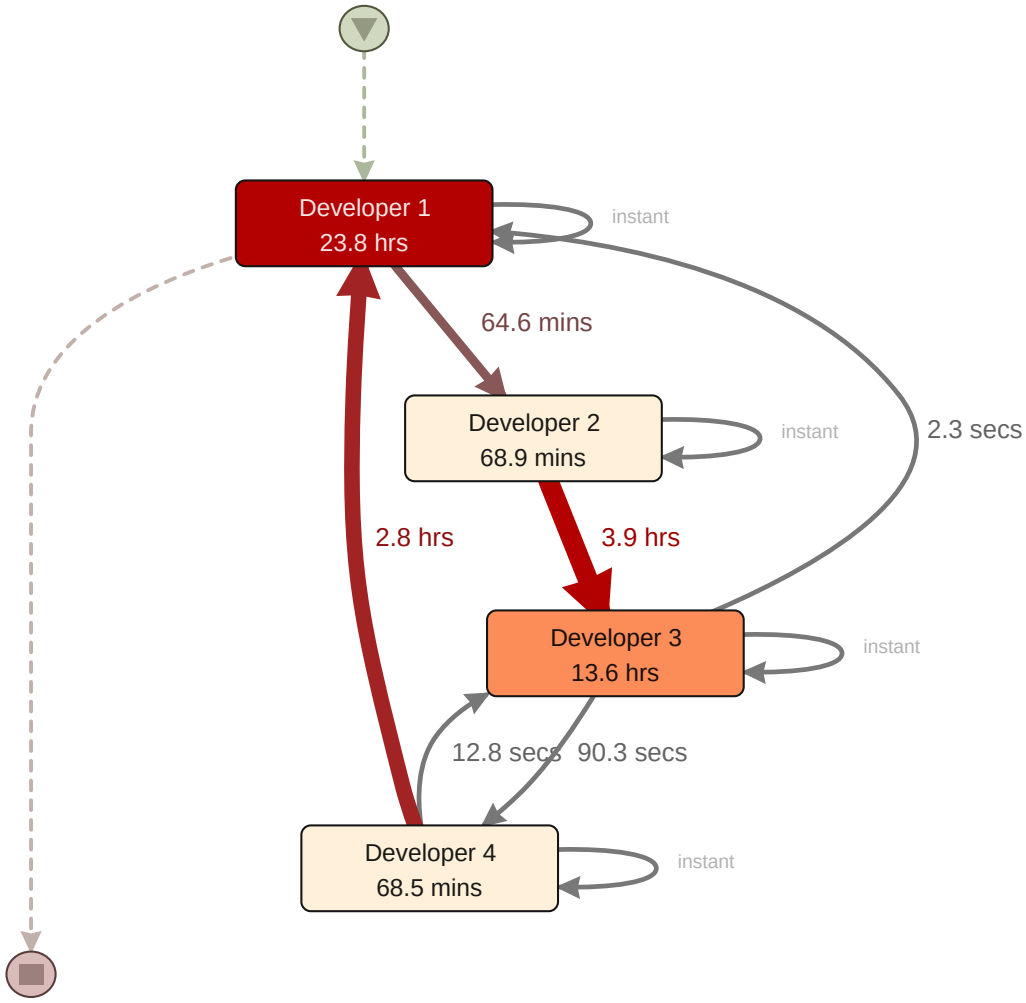

Figure 3.6: Handover between peers - Developers' Activities Duration

between groups design, to avoid the learning effects of paired designs, the only feasible solution was to use students as subjects, as referred in subsection 3.3.2.1. We cannot claim that these students are adequate surrogates for professional software developers.

3.6 Summary

Improving the efficiency and effectiveness of software development projects implies understanding their actual process. Given the same requirements specification, different software development teams may follow different strategies and that may lead to inappropriate use of tools or non-optimized allocation of effort on spurious activities, non-aligned with the desired goals. However, due to its intangibility, the actual process followed by each developer or team is often a black box.

The overall goal of this initial study was to improve the knowledge on how to measure

efficiency in development teams where a great deal of variability may exist due to the human-factor. The main focus was on the discovery of the underlying processes and compare them in terms of efficiency and effectiveness. By doing so, we were expecting to reveal potentially hidden costs and risks, so that corrective actions may take place on a timely manner during the software project life cycle.

Several independent teams of Java programmers, using the Eclipse IDE, were assigned the same software quality task, related to code smells detection for identifying refactoring opportunities and the quality of the outcomes were assessed by independent experts. On the events collected from the IDE, we used process mining techniques to discover development process models, evaluate their quality and compare variants against a reference model used as "best practice".

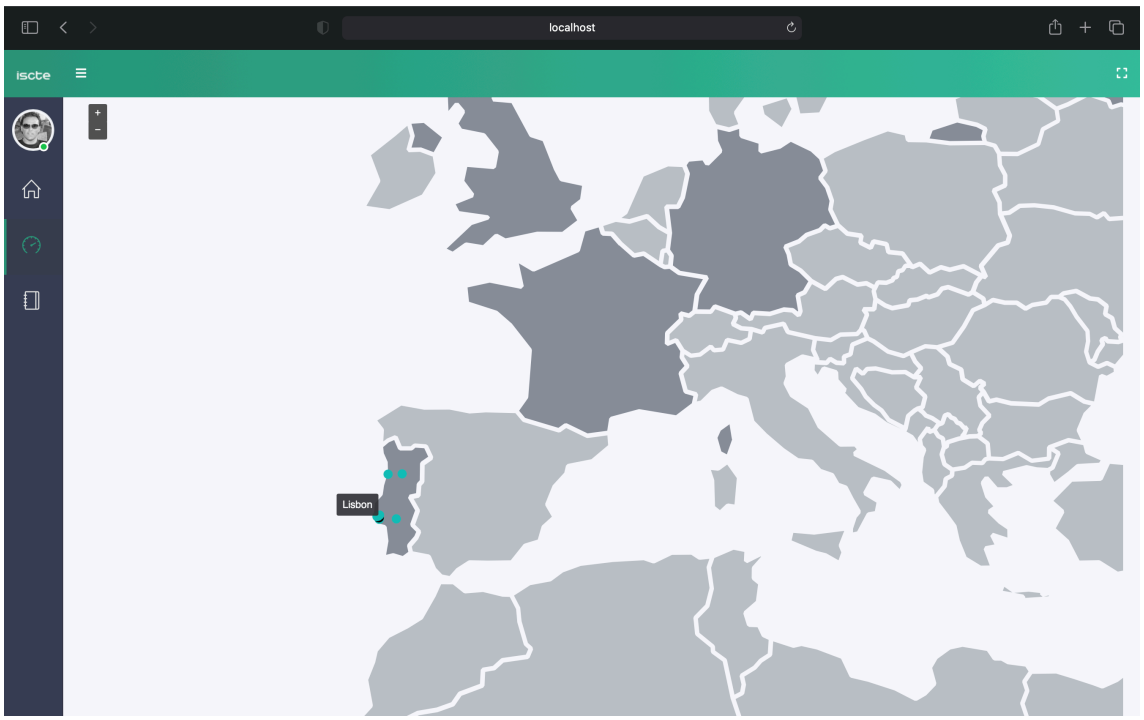

Figure 3.7: SDPM Team Dashboard - Geographic View

The main findings of this work also gave birth to a dashboard tool that will ingest event data and provide near real-time software development process insights, at the individual or team level, such as in the Personal Software Process (PSP) [93], or Team Software Process (TSP) approaches [94], but in an automated fashion. Figure 3.7 presents from a geographic perspective the developers' location, whilst Figure 3.8 provide project details for a specific developer. On Appendix A, we present the plugin to capture the events in section A.2 and the dashboard tool[10], installation guidelines and extra screenshots are shown in section A.4.

Regarding the main findings, teams whose process model was less complex, had the best outcomes and vice-versa. Comparing less complex process variants with the "best practice"process,

[10]Expected to be generally available as a Docker container soon

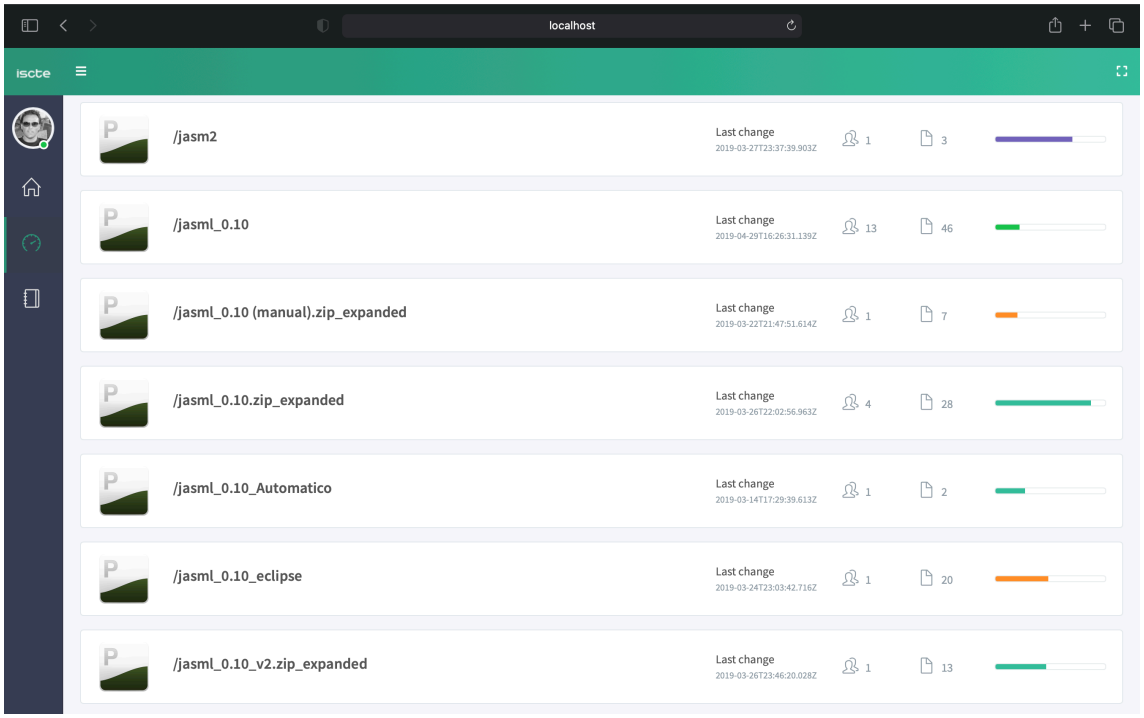

Figure 3.8: SDPM Personal Dashboard - Projects View

showed that they were also the ones with less differences in the control-flow perspective, based on activities frequencies. We have also determined which teams were most efficient through process analysis.

RQ1. We can not underestimate the fact that software development IDEs provide the users with a vast number of commands and menus to execute from, as seen in 3.2.3. Trying to model these, is indeed a challenge, and, most times, a spaghetti-like process is the result of a successful process discovery. However, from an event log containing user actions, we were able to model teams' behavior with moderate-to-strong Fitness and Precision values and yet achieve readable models. We are however well aware that these values should be validated with more experiments and different data. The need to answer RQ1 was vital to understand if we could indeed discover processes followed by different people, that may be using different tools, in different locations but contributing to the same final outcome or product. The importance of understanding and measure teams' dynamics has grown with the current business trends that lead to Global Software Engineering (GSE) and Global Software Development (GSD). This is one of the main challenges faced by GSE and GSD, as in those kinds of projects the usual monitoring techniques are obsolete [105].

RQ2. We were able to discover and reconstruct process models representing the efficiency of software development teams, where, in some cases, members were working individually, each with their own IDE setup configurations. We confirmed that process mining may play a fundamental role in assessing the efficiency of software development teams and in potentially contributing to keep them focused on their tasks by checking and enforcing compliance to the prescribed processes. Every project manager wants to have in the projects he/she manages

the most efficient and/or adequate resources. As this is expected to increase productivity in the development, measuring which teams or individuals are more efficient is a step further for better planning future software development projects.

RQ3. By assessing the way a task is executed and the proficiency achieved, as we did to answer RQ3, we were looking if there was any relation between those on the software development realm. This study can contribute to extend the discussion for the fact that the quality of a software product may well be dependent on the complexity of the processes followed. This research was resumed in a study presented in Chapter 4.

In general, teams with less complexity in their models were among the most proficient in the task. This means that, they not only understood what was requested, but also had the maturity to deliver what was expected by following a simple process. They were not only effective, they were also efficient by being focused in the task.

On the contrary, teams with insufficient proficiency produced long and complex models or, in very short time, they created very fuzzy models with too many generic events. These teams were the ones where more risk aroused from a development project perspective due to their erratic behavior and uncertainty around the expected deliveries. Some of those teams did not perform very well and quality was impacted, and some others did not even deliver what was expected. In a real-world scenario, these teams would have been identified as the most expensive teams because their productivity was indeed very low.

This gives us some evidence that teams' proficiency can be inferred by analyzing mined process models representing their behavior. We don't see this as a coincidence, however, to sustain this evidence, we may need to replicate this experiment in other contexts and with a larger number of teams and developers.

No relevant performance or bottleneck patterns were identified in the processes, and the reason for this may be related with the type of task requested, which did not impose restrictions on times to work on any artifact, and/or the reduced schedule imposed on the task.

We confirmed that, even for a well-defined software development task, there may be a great deal of process variability due to the human factor. We were able to identify when developers were more or less focused in the essential tasks they were required to perform. Less focused teams had the more complex process models, due to the spurious / non-essential actions that were carried out. In other words, they were less efficient. Experts' opinion confirmed that those teams also were less effective in their expected delivery. We therefore concluded that a self-awareness of the performed process rendered by our approach, may be used to identify corrective actions that will improve process efficiency (less wasted effort) and may yield to better deliverables, i.e. improved process effectiveness.

Chapter 4

Unveiling Process Insights

Contents

This chapter describes an experiment made to assess the feasibility of using only process metrics in order to understand the liaison of software and process complexities on refactoring sessions.

Companion Soundtrack: Time - Hans Zimmer (Inception Soundtrack)

"Information is the resolution of uncertainty."
—Claude Shannon(1916-2001)[1]

4.1 Introduction

A process[2] is *"a series of actions taken in order to achieve a result"*. In many business areas, either on delivering products and/or services, the quality of the outcome is very often related with the process followed to build it [58, 98, 193]. This is expected to be no different in the software development domain. Therefore, to fully comprehend how software quality and improved maintainability are achieved, one should look carefully to the process perspective to complement any code related analysis [71].

Software development is intrinsically a process and, accordingly, it is a blend of activities performed by developers, often working from different locations and using a multitude of languages, tools and methodologies in order to create a new product or maintain an existing one [71]. Since the early days of software development, it was understood that programming is an inherently complex and error-prone process, and to fully understand it, we should mine, in a timely and proper manner, all facets of that process [206]. Any relevant insights one may obtain should therefore originate from the activities and/or artifacts recorded in software repositories during the development life cycle.

Studies on estimating the effort to develop a certain artifact, the identification of software defects, the prediction of time to solve bugs or on software comprehension, and the detection of refactoring opportunities, are amongst the most common use cases for those repositories [19, 108, 109, 128, 143, 150, 156].

Refactoring on its own is still a very challenging activity. The identification of components to refactor and the forecast of which methods to embrace continue to be relevant topics for research [8, 174, 204, 205]. These challenges emerge partially due to the significant functionality limitations software repositories contain and the type of data they use [158].

Some authors confirm that developers perform refactoring tasks manually more frequently than automatically [156]. Furthermore, it has been observed, in a real-life scenario, that refactoring can be harmful when done manually, using only IDE native features or simply driven by developers' skills, as it may introduce non-expected defects in the code [112].

On trying to comprehend software development processes, including refactoring practices, many data sources, methods, and tools have been used with validated outcomes, but some others are yet to be fully exploited [136]. For example, since VCS are widely used by developers, researchers get easy access to historical data of many projects and use file-based VCSs as the primary source of code evolution data [110]. Although it is often convenient to use such repositories, research-based on VCS data is imprecise and incomplete [158].

As such, answering questions that correlate code changes with other activities (e.g., test runs, refactoring) is often unfeasible. Several reasons may contribute to it, as for instance:

[1] American electrical engineer and mathematician, known as "the father of information theory", and was the founder of practical digital circuit design theory.

[2] Adapted from https://dictionary.cambridge.org/dictionary/english/process

- developers may not commit all their tests and/or refactorings;
- there are many ways to refactor one version of the code, therefore it is important to determine the refactoring activities sequences and frequencies;
- often we cannot distinguish if a refactoring was done manually or through a tool, just by comparing source code snapshots [151].

4.1.1 Code vs. Process Analysis

Most published work on software quality-related issues is based on source code metrics, especially on Java systems [43, 67, 121]. Tools for collecting those metrics upon other frequently used languages, such as JavaScript or Python, are often not available, which expose well the difficulties to reproduce the same research on projects having diverse languages. In case those metric collection tools exist, they often require to share the source code with third-party organizations [164], particularly on cloud-based platforms. Such scenarios raise privacy and ownership issues on sensitive data. Source code obfuscation does not mitigate this problem because developers need to keep code semantics for interpreting the metrics in context.

Instead, mining the developers' activities and behaviors, the same is to say, to mine their development process fragments, may be a more feasible approach since it is not specific to any programming language, geographic location or development methodology followed.

Event data can be obfuscated without losing the process structure and coherence, therefore, whoever is responsible to analyze the logs can apply algorithms to discover process models in very similar ways as if the logs were not obfuscated [64]. In other words, events from the development tools and support activities can be collected, transformed and aggregated with fewer privacy concerns and technical hurdles. As such, it has been pointed out that software development event logs can be used to complement, or even replace, source code data in software development analytics-related tasks [40].

4.1.2 Contributions

It is frequent to find software prediction models using source code and ownership metrics [8]. However, periodically this data is not easily accessible or has imprecisions. Nowadays, development teams use a diversity of languages, methodologies and tools, therefore, the collection and aggregation of data from software projects remains a challenge. Additionally, process metrics have been found to be good predictors for modeling software development tasks [171].

Thus, we proposed earlier [36] and are now evaluating deeper the use of process metrics gathered from the IDE, as a way to enhance existing models or eventually, build new ones.

Software product and process metrics have long been proposed, as well as techniques for their collection [2, 3, 30, 39, 136, 209, 210]. However, the association between product and process dimensions is only marginally discussed in the literature [137]. In order to improve our understanding on the liaison between the type of development activities executed and the resulting software product characteristics, namely to ascertain if developers' behavior has an impact on software product quality, we collected data during a software quality improvement

task (application of refactoring operations) given to 71 development teams. Regarding developers' behavior, we recorded all events corresponding to the activities/tasks/operations team members performed within their IDE and used those events to mine the underlying process and extract their metrics. Regarding software quality, we collected complexity metrics before and after the refactoring actions took place. The main objectives for this work are, therefore:

- to assess the use of software process metrics to facilitate and improve the analysis and predictions on refactoring tasks and/or other generic software activities;
- to evaluate a possible association between the complexity of the produced code and developers' practices in different refactoring tasks;
- to build classification models for refactoring practices using only process metrics and assess the prediction accuracy of such approach.

4.2 Background

Empirical software engineering and software analytics are now mature research areas with substantial contributions to the software development best practices [224]. The knowledge base created to support those achievements took a great advantage from the experience gathered on analyzing past software projects. Based on the maturity obtained, it was possible to derive several models to measure software complexity, effort and relationships.

4.2.1 Early Models

Lines of Code(*LOC*). The identification and quantification of software size/defect relationship did not happen overnight. The first known "size" law, saying the number of defects D was a function of the number of *LOC*; specifically, D = 4.86 + 0.018 * i, was the result of decades of experience and was presented by Fumio Akiyama [6].

Cyclomatic Complexity. One of the most relevant propositions to assess the difficulty to maintain software was introduced by Thomas McCabe when he stated that the complexity of the code was more important than the number of *LOC*. He argued that when his "cyclomatic complexity" metric was over 10, the code is more likely to be defective [130]. This metric, underpinned by graph theory, went through thorough validation scrutiny and then became the first software metric recognized by a standardization body, the NIST [210], what makes it even more relevant in the context of this journal.

Halstead Complexity. On trying to establish an empirical science of software development, Maurice Howard Halstead, introduced the Halstead complexity measures [82]. These metrics, which are computed statically from the code, assume that software measurement should reflect the implementation or expression of algorithms in different languages, but be independent of their execution on a specific platform. Halstead's metrics were used, among other things, to assess programmers' performance in software maintenance activities (measured by the time to

locate and successfully correct the bug) [50].

Effort Estimators. Later, Barry Boehm proposed an estimator for development effort that was exponential on program size: effort = $a * KLOC^b * EffortMultipliers$, where 2.4 ≤ a ≤ 3 and 1.05 ≤ b ≤ 1.2 [25].

Henry and Kafura Metrics. These two authors defined and validated a set of software metrics based on the measurement of information flow between system components. Specific metrics are defined for procedure complexity, software modules complexity, and module coupling [87].

The above models were the foundation knowledge for what is nowadays often categorized as Software Development Analytics [34]. However, current development methods, tools and data repositories are very different from the past. Back in those years, software developers were mainly using a text editor and a compiler. Software projects were essentially built employing a single programming language, following a fairly simple development methodology and the developers were rarely located in different geographies or across multiple time zones. These workspace conditions have changed.

4.2.2 Modern Days

In 2019, JetBrains[3] polled almost 7000 developers about their development ecosystem. Results show that more than 30 different programming languages are being used and confirmed that web back-end, web front-end and mobile applications are the type of applications mostly developed, with figures of 60%, 46% and 23%, respectively. It was unanimous the adherence of cross-platform development frameworks and 80% said they use any type of source code collaboration tool, 75% use a standalone IDE and 71% use a lightweight desktop editor. Almost 50% said they use Continuous Integration (CI) and Continuous Deployment (CD) and issue tracking tools. Less than 15% responded that they use any sort of static analysis, code review and in-cloud IDE tools. Table 4.1 presents the key takeaways from the mentioned survey.

In summary, currently, a software development ecosystem has to deal with at least the following facets:

- **Multi-Language Ecosystem.** According to a recent work about multi-language software development [129], the authors present evidences that non-trivial enterprise software systems are written in at least 7 programming languages and, a previous work showed that in the open source world alone, the average is 5 languages per project. Among these, one may find General-Purpose Languages (GPL) general-purpose languages(GPL) such as Java or C# and also domain-specific languages(DSL) like SQL and HTML, and cross-language links are also quite common, meaning some code artifacts are shared between languages. As a result, developers confirm they find more problems in activities such as implementing new requirements (78%) and in refactoring (71%).

[3] https://www.jetbrains.com/lp/devecosystem-2019/

Table 4.1: Survey Key Takeaways*

Findings	
Programming Languages Overall Results	
Java	The most popular primary programming language
JavaScript	The most used overall programming language
Go	The most promising language as 13% said they will adopt it
Python	Most studied language as 27% said they used it in the last 12 months
Languages used in last 12 months	
JavaScript(69%), HTML/CSS(61%), SQL(56%), Java(50%), Python(49%)	
Shell Scripting(40%), PHP(29%), TypeScript(25%), C#(24%), C++(20%)	
Development Environments(Operating Systems)	
Windows(57%), macOS(49%), Unix/Linux(48%), Other(1%)	
Type of Application Development	
Web Back-End(60%), Web Front-End(46%), Mobile(23%), Libraries(14%)	
Desktop(12%), Other Back-End(16%), Data Analysis(13%), Machine Learning(7%)	
Type of Tests Used	
Unitary(71%), Integration(47%), End-to-End(32%), Other(2%), Don't Test(16%)	
Targeted Mobile Operating Systems & Frameworks Used	
Android(83%), iOS(59%), Other(3%)	
React Native(42%), Flutter(30%), Cordova(29%), Ionic(28%), Xamarin(26%)	
Regularly Used Tools	
Source Code Collaboration Tool(80%), Standalone IDE(75%)	
Lightweight Desktop Editor(71%), CI/CD Tool(45%), Issue Tracker(44%)	
Static Analysis Tool(13%), Code Review Tool(10%)	

*All values(%) represent the percentage of affirmative responses to each item

- **IDE Evolution.** A substantial change was carried in the IDEs. Software development moved away from the early days of the code editor. As confirmed by the Jetbrains poll, developers now use powerful platforms and frameworks which allow them to be more productive on their jobs. This results from the combination of different software development life cycle activities, such as: requirements elicitation, producing analysis and design models, programming, testing, configuration management, dependencies management or continuous integration into one single tool such as `Eclipse`, `IntelliJ IDEA`, `Netbeans` or `Visual Studio Code`. These tools support the needs of different stakeholders, as they embed a myriad of plugins available in their marketplaces. These plugins are not just available, they are properly customized for specific users/purposes, such as for modellers, programmers, testers, integrators or language engineers.

- **Low Code and No Code Paradigms.** Modern software development practices make consistent use of both approaches. They enable faster development cycles requiring little

to no coding in order to build and deliver applications and processes. Low-code development platforms are seen as advanced IDEs which employ drag-and-drop software components and visual interfaces to replace extensive coding. With high-level visual modeling languages, they provide higher levels of abstraction that allow a major reduction in hand-coding to develop an application [86]. In the extreme case we have no-code development where, by definition, textual programming is banned, giving rise to the so-called *citizen developers*. The most notable examples are Online Application Generators (OAG)s that automate mobile and web app development, distribution, and maintenance, but this approach is claimed to be pledged with security vulnerabilities [161]. This paradigm shift in software development may also require a change in the way we assess critical properties of a software project, such as, quality, maintainability, and evolvability.

- **Global Software Development.** The aforementioned IDE platforms facilitated collaboration and the adoption of GSD. Nowadays, a single software project often has developers, testers and managers located in different time zones and distinct world regions or countries [160].

- **Continuous Integration and Continuous Deployment (CI/CD) and DevOps.** CI/CD have seen an incremental usage in the last few years. However, efficient CI/CD pipelines are rare, particularly in the mobile apps world where developers seem to prefer the execution of *ad hoc* tasks [49]. Whilst CI/CD focuses more on the automation of tools along a defined software life cycle, DevOps has major concerns with the responsiveness, responsibilities and processes within the development, the deployment and the operational phases of software projects. Keeping these intertwined processes compliant with organizational rules is therefore a persistent requirement.

- **Resource Coordination.** It is still one of the fundamental problems in software engineering [88] and it can be characterized as a socio-technical phenomenon. Understanding the dependencies between development tasks and discover teams' behaviours continues to be a challenge in resource allocation and coordination of modern software projects.

Software product repositories have many limitations in terms of the process data they handle. For example, these repositories usually deal only with source code and do not track the developers' geographic location, their workflows within the IDE nor the developers' environment characteristics. A complete repository of process related data with the communications, activities, decisions and actions taken by developers, testers and project managers, are, most of the time, if not always, neglected when the goal is to study a development process. Usually, even if the authors claim they are studying a process, they are often doing it using only artifact related data [137].

With the existing diversity of languages, methodologies, tools and the fact that resources are now distributed across the world and originate from multiple cultures with different skills, it is somewhat an anachronism to keep using old methods to assess, for example, complexity or build cross-cutting analytical models in current software projects. New approaches, supporting multi-languages, being multi-process aware, and keeping geography diversity transparent are called for, such as our pioneering approach for mining of software development processes based

on the IDE event logs. That approach, dubbed Software Development Process Mining [36], allows reversing engineer a complete software development process, just a process fragment or simply *ad hoc* activities performed by developers, by mining event logs taken from real software development activities.

4.2.3 Related Work

To address the incompleteness of data sources related with software repositories, we strongly believe that Software Development Process Mining based at least on the IDE(but not limited to) can play that role and Process Mining tools and methods can be the vehicles to achieve that goal. Many authors have followed similar paths, bringing not only evidences for its usefulness but also valid contributions to improve established methods.

A decade ago, [165] mined software repositories to extract knowledge about the underlying software processes, and [180, 181] have learned about user behavior from software at operations. [97] was able to extract events from `Eclipse` and have discovered, using a process mining tool, basic developers' workflows. Some statistics were computed based on the activities executed and artifacts edited.

[141] presented an application of mining three software repositories: team wiki (used during requirement engineering), version control system (development and maintenance) and issue tracking system (corrective and adaptive maintenance) in the context of an undergraduate Software Engineering course. Experimentation revealed that not only product but process quality varies significantly between student teams and mining process aspects can help the instructor in giving directed and specific feedback. However, in this case, IDE usage mining was not contemplated.

The working habits and challenges of mobile software developers with respect to testing were investigated by [49]. A key finding of this exhaustive study, using 1000 Android apps, demonstrates that mobile apps are still tested in a very ad hoc way, if tested at all. A another relevant finding of this study is that CI/CD pipelines are rare in the mobile apps world (only 26% of the apps are developed in projects employing CI/CD) - authors argue that one of the main reasons is due to the lack of exhaustive and automatic testing. Therefore, distinguishing during development sessions the type of tests being done can contribute to the overall software quality.

[217] explored if one can characterize and identify which commits will be reverted. An identification model (e.g., random forest) was built and evaluated on an empirical study on ten open source projects including a total of 125,241 commits. The findings show that the 'developer' is the most determinant dimension of features for the identification of reverted commits. This suggests that assessing developers behaviors can lead to better understand software products quality.

[85] studied the dialogue between users and developers of free apps in the Google Play Store. Evidences found, showed that it can be worthwhile for app owners to respond to reviews, as responding may lead to an increase in the given rating and that studying the dialogue between user and developer can provide valuable insights which may lead to improvements in the app store and the user support process. We believe the same rationale may be applied

to comprehend the workflows and dialogues between developers and project owners, and how that may impact software products.

Development activities were extracted by [18] from non-instrumented applications and used machine learning algorithms to infer a set of basic development tasks. However, in this case, no process mining techniques were used to discover any pattern of application usage. The extraction of usage smells was the focus of [52], where a semi-automatic approach was adopted to analyze a large dataset of IDE interactions using cluster analysis. Again, process mining techniques were not used. Process mining was indeed used by [119] to gain knowledge on software under operation (not under development) by analyzing the hierarchical events produced by application calls(eg: execution of methods within classes) at runtime.

[216] collected events from the IDE to measure program comprehension and evaluated the correlation between developers' activities and the time they spent on them. Despite the fact that a process was being studied, no evidence of using process mining methods was provided.

A few authors have also followed the route we suggested earlier and resumed in [37]. As such, we are witnessing more evidences that it is indeed a valid approach, therefore, [11] used process mining to evaluate developers' coding behavior in software development processes. Process models were discovered and used to classify the developers as low-performing and high-performing profiles. With a similar goal, in [12], a different miner algorithm was assessed to obtain complementary results and in [10], developers' profiling was achieved by mining event logs from a web-based cloud IDE.

Finally, [8] highlights the importance of having more fine-grained process metrics in prediction models and evaluated several machine learning algorithms in predicting software refactoring opportunities. This work focuses on deciding when, what and why to refactor, however, it does not address which refactor practice was indeed applied.

The studies mentioned above used a multitude of process mining techniques, statistics and machine learning methods. Different data source types have been used to extract the information needed to support them. However, to the best of our knowledge, none of these works combine process and product metrics with the aim of assessing potential correlations and/or impacts between the process and the product. Moreover, none uses only process metrics to discover work patterns or to predict development behaviors, particularly, refactoring practices.

4.3 Study Setup

We setup an environment where the same well-defined tasks on software quality assurance was performed independently by several teams.

Our research guaranteed that all teams had similar backgrounds and performed the same task upon the same software system. This approach was used to block additional confounding factors in our analysis. The task targeted a complex open-source Java system named `Jasml (Java Assembling Language)`[4].

To understand the work developed by each team in each task, we collected the corresponding IDE events for mining the underlying process. At the end of each task, we also collected the

[4]http://jasml.sourceforge.net/

modified Jasml project code for each team and obtained the corresponding product metrics.

4.3.1 Subject Selection

Our subjects were the finalists (3rd year) of a B.Sc. degree on computer science at the Iscte university, attending a compulsory software engineering course. They had similar backgrounds as they have been trained across the same set of courses along their academic path. Teams were assembled with up to 4 members each and were requested to complete a code smells detection assignment, aiming at identifying refactoring opportunities and then to apply them.

4.3.2 Data Collection

The participants were requested to perform the refactoring tasks in two different ways: **Automatically** and **Manually**.

The refactoring tasks had the following requirements:

- **Automatic Refactoring(AR).** Executed from March 1st to March 20th, using JDeodorant[5]. This tool suggests refactoring opportunities by detecting, among others, the following four well-known code smells: Long Method, God Class, Feature Envy and Type Checking [20]. Once participants have detected the occurrences of those code smells, they were required to apply JDeodorant's fully automated refactoring features to fix the critical ones.
- **Manual Refactoring(MR).** This task was pursued from March 21st to 28th and differed from the previous one because JDeodorant automatic refactoring capabilities were banned. Instead, subjects could use Eclipse's native interactive refactoring features or perform the refactorings manually.

The Eclipse IDE has an internal event bus accessed by the interface IEventBroker[6] which is instantiated once the application starts. It contains a publishing service to put data in the bus, whilst the subscriber service reads what's in that bus. Using this feature we developed an Eclipse plugin[7] capable of listening to the actions developers were executing. Before the experiment, the plugin was installed on each subject's IDE, and later, all subjects received an unique *username/key* pair as credentials.

A sample event instance collected with our plugin is presented in listing 4.1. The field tags are self explanatory.

[5]https://users.encs.concordia.ca/ nikolaos/jdeodorant/
[6]https://wiki.eclipse.org/Eclipse4/RCP/Event_Model
[7]https://github.com/jcaldeir/iscte-analytics-plugins-repository

Listing 4.1: Sample Eclipse Event Instance

```
{
"team" : "Team-10",
"session"  : "dkoep74-ajodje5-63j3k2",
"timestamp_begin" : "2019-05-03 16:53:52.144",
"timestamp_end" : "2019-05-03 16:54:04.468",
"fullname"  : "John User",
"username"  : "john",
"workspacename"  : "Workspace1",
"projectname"  : "/jasml_0.10",
"filename"  : "/jasml_0.10/src/jasml.java",
"extension"  : "java",
"categoryName": "Eclipse Editor",
"commandName": "File Editing",
"categoryID": "org.eclipse.ui.internal.EditorReference",
"commandID": "iscte.plugin.eclipse.commands.file.edit",
"platform_branch": "Eclipse Oxygen",
"platform_version": "4.7.3.M20180330-0640",
"java": "1.8.0_171-b11",
"continent": "Europe",
"country": "Portugal",
"city": "Lisbon",
....
"hash": "00b7c0ef94e02eb5138d33daf38054e3" //To detect event tampering
}
```

4.3.2.1 Data Storage

Collected data was stored locally on each subject's computer in a CSV file. Whenever Internet connection was available, the same data was stored in real-time in the cloud[8]. This storage replication mechanism allowed for offline and online collection[9]. The final dataset, combining the two different sources, was then loaded into a MySQL database table where the username and event timestamps that formed the table's unique key were used to detect and avoid duplicated data insertions. Figure 4.1 presents the complete schema for the data collection workflow. We use the Business Process Model and Notation (BPMN) standard process definition language for that purpose [44].

4.3.2.2 Data Preparation

When the software quality task ended, we collected from each team their projects' code together with the events files containing the actions performed during the aforementioned activities. As such, each team produced and delivered two new `Jasml` projects, one for the automatic and another for the manual refactoring. The events files would map events for the two different tasks, as they were done in different time frames.

All events stored in the database were imported into the ProM process mining tool[10] and converted to the IEEE XES standard format [80]. The following event properties were mapped

[8] https://azure.microsoft.com/en-us/services/event-hubs/

[9] The plugin currently supports the collection of events locally in CSV and JSON files; stream events to Azure Event Hub and Kafka remotely; and uses an integration with Trello to extract project activities which can be triggered as manual events by the developers. Kafka and Trello integrations were not used in this experiment.

[10] Version 6.8, available at http://www.promtools.org

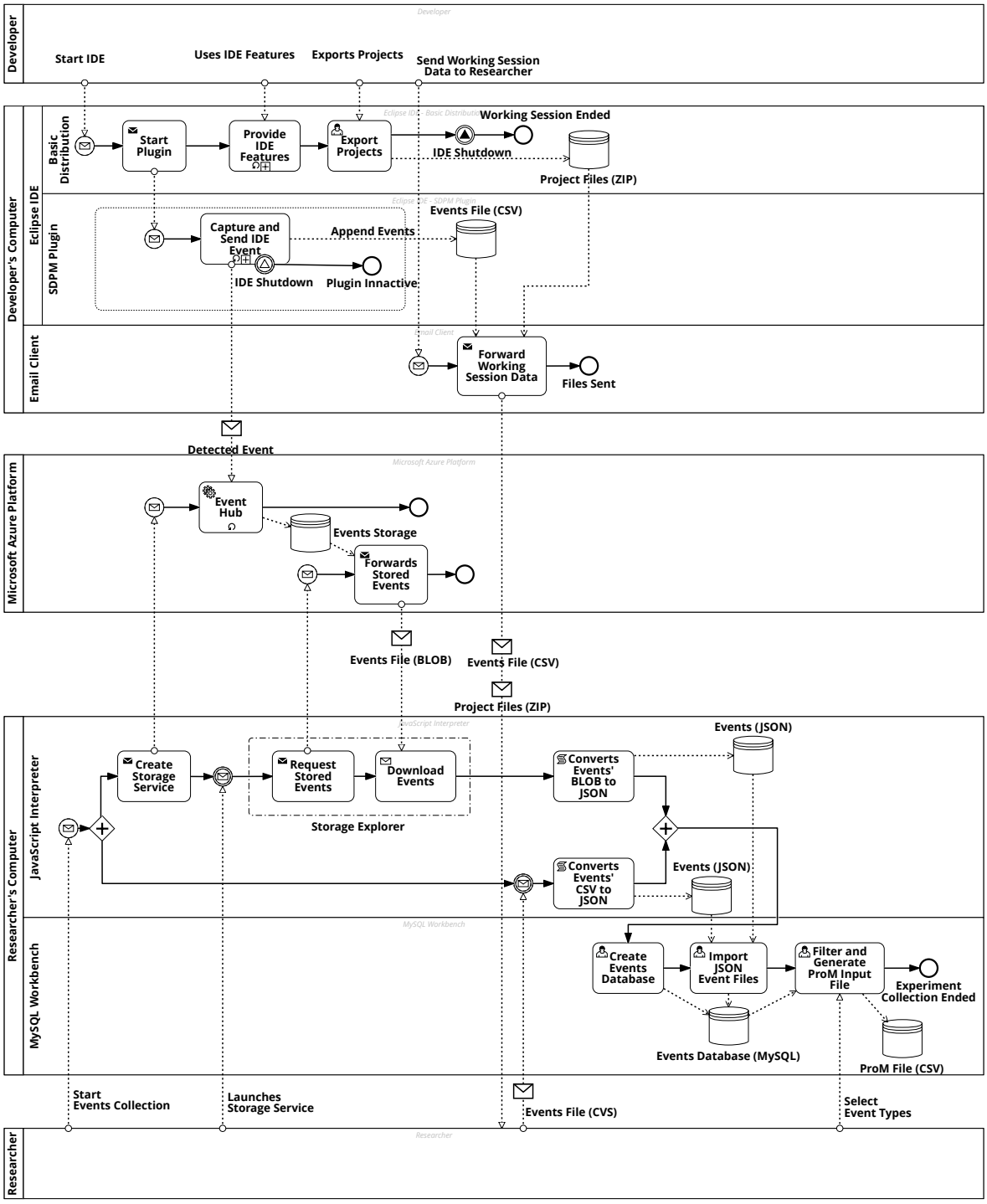

Figure 4.1: Experiment Data Collection Workflow

when converting to XES format:

- *team* and *session* were used as **CaseID** since we were interested to look into process instances of teams and their multiple development sessions, not of individual programmers.
- Properties *filename*, *categoryName* and *commandName* forming a hierarchical structure were used as the **Activity** in the process.
- The *timestamp_begin* and *timestamp_end* were both used as activity **Timestamps**.

- Other properties were not used in the process discovery phase, however, they were later used for metrics aggregation and context analysis.

4.3.3 Data Analysis

4.3.3.1 Context

All teams started with the same version of **Jasml 0.10**, therefore, we had two relevant moments to get measures from:

1. The initial moment ***(t0)***, when we extracted the metrics for the initial product version. However, we didn't know how it was built, therefore, we were missing[11] the process metrics.

2. The end of the task ***(t1)***, when we extracted again the product metrics for the changed **Jasml 0.10** project of each team as they stand after the refactoring sessions. In addition, we had also IDE usage events which provide evidences on how the product was changed.

Following data extraction, we computed, for each product metric defined in Table 4.2, their relative variance as shown by Equation 4.1. The relative variance variables were the ones we used in all **RQs**.

$$\Delta product\ metrics_{\mathbf{(t1\text{-}t0)}} = \frac{product\ metrics_{\mathbf{(t1)}} - product\ metrics_{\mathbf{(t0)}}}{product\ metrics_{\mathbf{(t0)}}} * 100 \tag{4.1}$$

The relative variance was used in order to generalize our approach, thus, making it applicable in scenarios where different teams work on distinct software projects.

Process metrics described in Table 4.3 were derived from the events dataset captured between moments ***(t0)*** and ***(t1)***, either by summing the events or using the method described in 4.3.3.3. These metrics may be seen as a representation of the effort done by each team during the refactoring practices.

The complete workflow followed in data pre-processing, aggregation and analysis is presented in Figure 4.2.

4.3.3.2 Product and Process Metrics

To extract software metrics we used the plugin built by Sauer[12]. Although having more than a decade of age, it is still one of the more proven and popular options regarding open source metrics plugins for Eclipse.

The plugin itself offers a simple interface and reporting capabilities with which users can define optimal ranges and issue warnings for certain metrics, as well as being able to export calculated metrics to XML files. The set of metrics obtained by this plugin are presented in Table 4.2.

[11] In reality we may consider all of them to be zero
[12] http://metrics.sourceforge.net

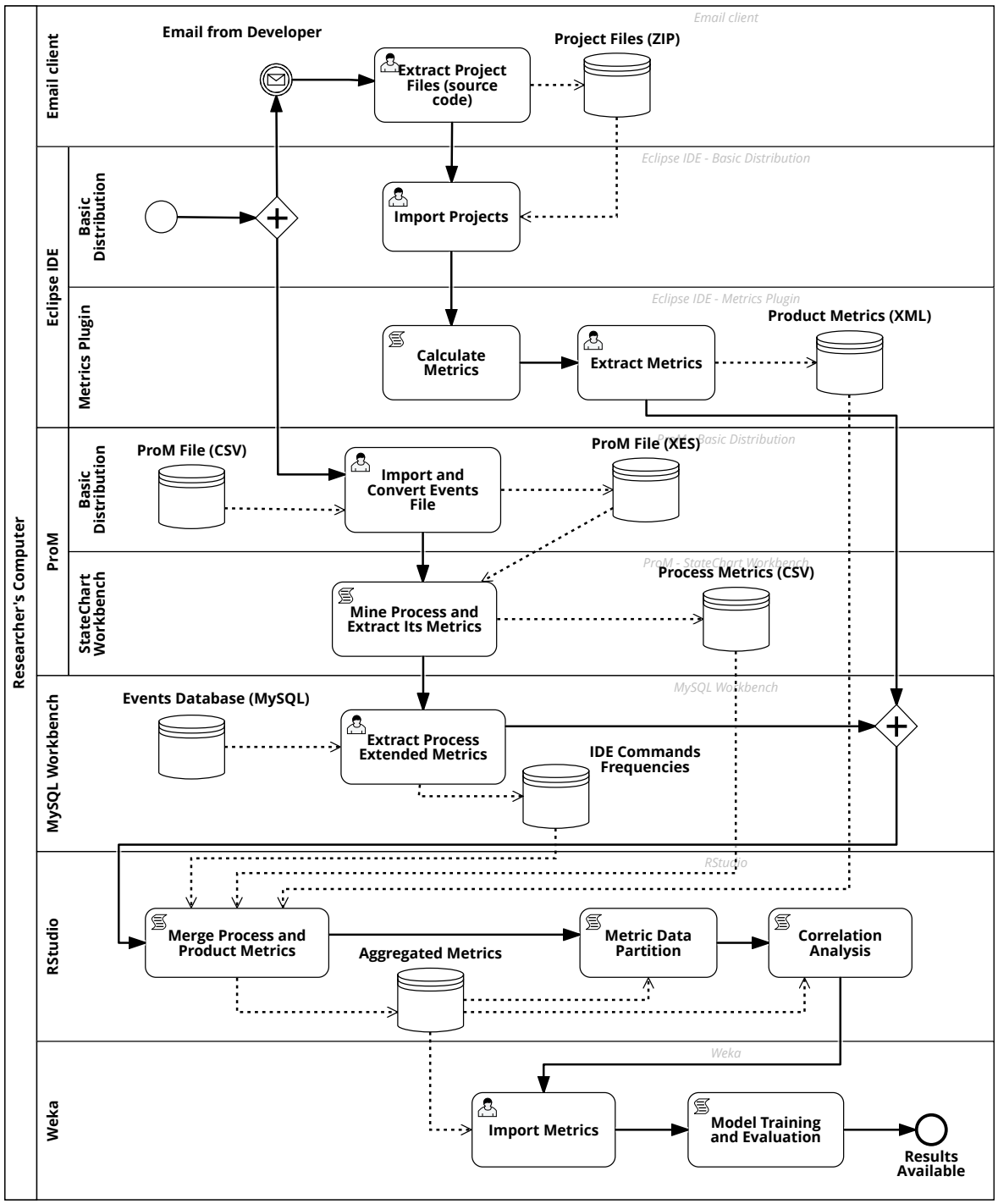

Figure 4.2: Study Computation and Analysis Process

4.3.3.3 Process Discovery

Several well known algorithms exist to discover process models, such as, the α-algorithm, the heuristics, genetic and the fuzzy miner amongst others [73, 127]. Our need to discover and visualize the processes in multiple ways lead us to choose the `ProM's StateChart Workbench` plugin [119]. This plugin, besides supporting process model discovery using multiple hierarchies and classifiers, also allows to visualize the model as a Sequence Diagram and use notations such as Petri Nets and Process Trees. This plugin is particularly suitable for mining software

Table 4.2: Product Metrics Description

Name	Description	Scale
VG	McCabe Cyclomatic Complexity (Avg. per Method)	Numeric
PAR	Number of Parameters (Avg. per Method)	Numeric
NBD	Nested Block Depth (Avg. per Method)	Numeric
CA	Afferent Coupling (Avg. per Package Fragment)	Numeric
CE	Efferent Coupling (Avg. per Package Fragment)	Numeric
RMI	Instability (Avg. per Package Fragment)	Numeric
RMA	Abstractness (Avg. per Package Fragment)	Numeric
RMD	Normalized Distance (Avg. per Package Fragment)	Numeric
DIT	Depth of Inheritance Tree (Avg. per Type)	Numeric
WMC	Weighted methods per Class (Avg. per Type)	Numeric
NSC	Number of Children (Avg. per Type)	Numeric
NORM	Number of Overridden Methods (Avg. per Type)	Numeric
LCOM	Lack of Cohesion of Methods (Avg. per Type)	Numeric
NOF	Number of Attributes (Avg. per Type)	Numeric
NSF	Number of Static Attributes (Avg. per Type)	Numeric
SIX	Specialization Index (Avg. per Type)	Numeric
NOP	Number of Packages	Numeric
NOC	Number of Classes (Avg. per Package Fragment)	Numeric
NOI	Number of Interfaces (Avg. per Package Fragment)	Numeric
NOM	Number of Methods (Avg. per Type)	Numeric
NSM	Number of Static Methods (Avg. per Type)	Numeric
MLOC	Method Lines of Code (Avg. per Method)	Numeric
TLOC	Total Lines of Code	Numeric
VG_LEVEL	Different levels of **ΔVG** (**LOW, MEDIUM, HIGH**)	Categorical

logs, where an event structure is supposed to exist, but it also supports the mining of other so-called generic logs.

Events collected from software in operation (e.g. Java programs) reveals the presence of a hierarchical structure, where methods reside within classes, and classes within packages [118]. The same applies to IDE usage actions where identified menu options and executed commands belong to a specific category of command options built-in the `Eclipse` framework. Supported by this evidence, we used the Software log Hierarchical discovery method with a Structured Names heuristic to discover the processes based on the fact that the events were using a *filename*|*category*|*command* structure (e.g. `/jasml0.10/src/jasml.java`|`Eclipse Editor`|`File Open`).

Several perspectives can be used to discover and analyze a business process. The commonly used are: `Control-Flow`, `Organizational`, `Social` and `Performance`. We have focused on the `Control-Flow` perspective in this work. It defines an approach that consists in analyzing how each task/activity follows each other in an event log, and infer a possible model for the behavior captured in the observed process.

Process metrics, shown in Tables 4.3 and 4.4 were obtained using the discovery method described in 4.3.3.3, and by running queries into the events database as presented in Figure 4.2.

Table 4.3: Process Metrics Description

Name	Description	Scale
DEV	Number of Developers	Numeric
SES	Number of User/Development Sessions	Numeric
EVTS	Number of Events Collected	Numeric
NFILES	Number of Unique Files Touched	Numeric
NCOM	Number of Unique Commands Issued in IDE	Numeric
PCCPF	Process Cyclomatic Complexity per File Touched	Numeric
EC	Number of Event Classes	Numeric
NOA	Number of Activities	Numeric
NSS	Number of Simple States	Numeric
NCS	Number of Composite States	Numeric
NOT	Number of Transitions	Numeric
PCC	Process Cyclomatic Complexity	Numeric
NVER	Number of Unique IDE Versions	Numeric
NCAT	Number of Unique Command Categories	Numeric
NPLA	Number of Unique IDE Platforms	Numeric
NISP	Number of Unique Geographic Locations	Numeric
NOS	Number of Unique Operating Systems	Numeric
NPER	Number of Unique Perspectives used in the IDE	Numeric
PCC_LEVEL	Different levels of **PCC** (**LOW, HIGH**)	Categorical

4.3.3.4 Data Partitioning

We used the **k-means** clustering algorithm to compute new variables based on the partition of the teams across different levels (clusters) of Process Cyclomatic Complexity *(PCC)* and McCabe Cyclomatic Complexity variance *(Δ VG)*. The decision of how many clusters to use **(k)** was supported by a detailed analysis of the **Elbow** and **Silhouette** methods:

- **Elbow Method.** It is frequently used to optimize the number of clusters in a data set. This heuristic, consists of rendering the explained variation as a function of the number of clusters, and picking the elbow of the curve as the optimal number of clusters to use. In cluster analysis, the elbow method runs **k-means** clustering on the dataset for a range of values for **k** (say from 2-10), and then, for each value of **k** computes an average score for all clusters. The distortion score is computed as the sum of square distances from each point to its assigned center [170].

- **Silhouette Method.** It is a commonly used approach of interpretation and validation of consistency within clusters of data. Provides a concise graphical representation of how well each object has been classified within the corresponding cluster. The Silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The silhouette can be calculated with any distance metric, such as the Euclidean distance or the Manhattan distance, and ranges from -1 to +1. A high value indicates that the object is well matched to its own cluster and poorly matched to neighboring clusters. The clustering configuration is appropriate if most objects have a high value. If many objects have a low or negative value, then the clustering configuration

Table 4.4: Process-Extended Metrics Description

Category	Name	Scale
Refactor	Java-Extract Method	Numeric
	Java-Move - Refactoring	Numeric
	Java-Extract Class...	Numeric
	Java-Rename - Refactoring	Numeric
	Delete Resources	Numeric
	Java-Encapsulate Field	Numeric
	Java-Change Method Signature	Numeric
	Java-Move Type to New File	Numeric
Eclipse Editor	File Open	Numeric
	File Editing	Numeric
	File Close	Numeric
Eclipse View	Project Explorer	Numeric
	Package Explorer	Numeric
	Long Method	Numeric
	God Class	Numeric
	Code Smell Visualization	Numeric
	Type Checking	Numeric
	Feature Envy	Numeric
	Duplicated Code	Numeric
Edit	Find and Replace	Numeric
	Copy	Numeric
	Paste	Numeric
	Cut	Numeric
	Delete	Numeric
	Undo	Numeric
	Redo	Numeric
File	Import	Numeric
	Refresh	Numeric
	Save	Numeric
	Save All	Numeric
Source Compare	Generate Getters and Setters	Numeric
	Select Next Change	Numeric
.....	//List is truncated on purpose	
.....	//List size is ≈250	
Text Editing	Delete Previous Word	Numeric

may have too many or too few clusters and, as such, requires further research before a decision on the optimal number of **k** clusters is made [107].

4.3.3.5 Model Selection with Hyperparameter Optimization

To build, tune model parameters as recommended [140, 215], train, evaluate and select the best-fit classification models presented in Tables 4.8 and 4.9, we used **Weka** and the **Auto-Weka** plugin. **Weka** (Waikato Environment for Knowledge Analysis) is a popular suite of machine learning software written in Java. It's workbench contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to this functionality [102]. **Auto-Weka** is a plugin that installs as a **Weka** package

and uses Bayesian optimization to automatically instantiate a highly optimized parametric machine learning framework with minimum user intervention [200].

4.3.3.6 Model Evaluation

Several evaluation metrics can be used to assess model quality in terms of false positives/negatives (FP/FN), and true classifications (TP/TN). However, commonly used measures, such as **Accuracy, Precision, Recall and F-Measure**, do not perform very well in case of an imbalanced dataset or they require the use of a minimum probability threshold to provide a definitive answer for predictions. For these reasons, we used the **ROC**[13], which is a threshold invariant measurement. Nevertheless, for general convenience, we kept present in Tables 4.8 and 4.9 all the evaluation metrics. **ROC** gives us a 2-D curve, which passes through (0, 0) and (1, 1). The best possible model would have the curve close to y = 1, with and area under the curve (**AUC**) close to 1.0. **AUC** always yields an area of 0.5 under random-guessing. This enables comparing a given model against random prediction, without worrying about arbitrary thresholds, or the proportion of subjects on each class to predict [171].

4.3.4 Research Questions

The research questions for this work are:

- **RQ4:** How different refactoring methods perform when the goal is to reduce complexity, future testing and maintainability efforts?.
  Methods Used. Process Mining Model Discovery, Descriptive statistics and Cluster Analysis.

- **RQ5:** Is there any association between software complexity and the underlying development activities in refactoring practices?
  Methods Used. Process Mining Model Discovery, Correlation Analysis using the Spearman's rank correlation.

- **RQ6:** Using only process metrics, are we able to predict with high accuracy different refactoring methods?
  Methods Used. Supervised and Unsupervised Learning Algorithms with Hyperparameter Optimization.

- **RQ7:** Using only process metrics, are we able to model accurately the expected level of complexity variance after a refactoring task?
  Methods Used. Supervised and Unsupervised Learning Algorithms with Hyperparameter Optimization.

[13] Receiver operating characteristic (**ROC**) is a curve that plots the true positive rates against the false positive rates for all possible thresholds between 0 and 1.

4.4 Study Results

In this section, we present the experiment results with respect to our research questions.

RQ4. How different refactoring methods perform when the goal is to reduce complexity, future testing and maintainability efforts?

In this **RQ**, we used as product metrics, the ones identified in section 4.3.3.2. Since IDE usage is a sequence of actions (it can be seen as a process, or at least, as a process fragment), we used as process metrics the ones identified in 4.3.3.3. Notice that both, product and process metrics, have been computed to obtain the Δ between **t1 and t0**.

Table 4.5: Teams' Statistics

Task Mode	Teams	Dev.	Ses.	Evts.	Δ VG	PCC
Automatic Refactoring	32	65	150	10443	7.81%	166.5
Manual Refactoring	39	52	170	22676	2.69%	300.3
Total	71	117	320	33119		

Dev - Developers, **Ses** - Sessions, **Evts** - Events,
Δ **VG** - McCabe Cyclomatic Complexity Reduction %(mean),
PCC - Process Cyclomatic Complexity(mean)

Table 4.6: Teams' Refactoring Results

Metric Name	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
Automatic Refactoring						
Δ **VG**	2.68%	5.87%	6.95%	7.81%	8.84%	16.77%
PCC	24.0	77.0	168.5	166.5	218.2	407.0
Manual Refactoring						
Δ **VG**	0.32%	0.62%	0.94%	2.69%	3.92%	13.98%
PCC	53.0	152.0	275.0	300.3	407.0	738.0
Data Partition						
VG_LEVEL	**LOW** = [0%, 4%]; **MEDIUM** = [4.1%, 9%]; **HIGH** = [>9%]					
PCC_LEVEL	**LOW** = [0, 285]; **HIGH** = [>285]					

Δ **VG** - McCabe Cyclomatic Complexity Reduction %,
PCC - Process Cyclomatic Complexity

We had 32 teams performing automatic refactoring using the *JDeodorant* plugin, and 39 doing manual refactoring supported only by the Eclipse native features and/or driven by the developers experience and skills. Table 4.5 shows the total number of developers and their activities, here referred as development sessions. In Table 4.6 we show measures of central tendency and measures of variability regarding the distribution of Δ*VG* and *PCC*, together with how both were partitioned.

Figure 4.3 provides evidence for selecting the optimal number of clusters to partition the data according to **LOW** or **HIGH** levels of process cyclomatic complexity used in Figure 4.4. The same clustering method was used to partition the different levels of software cyclomatic complexity as **LOW**, **MEDIUM** or **HIGH**.

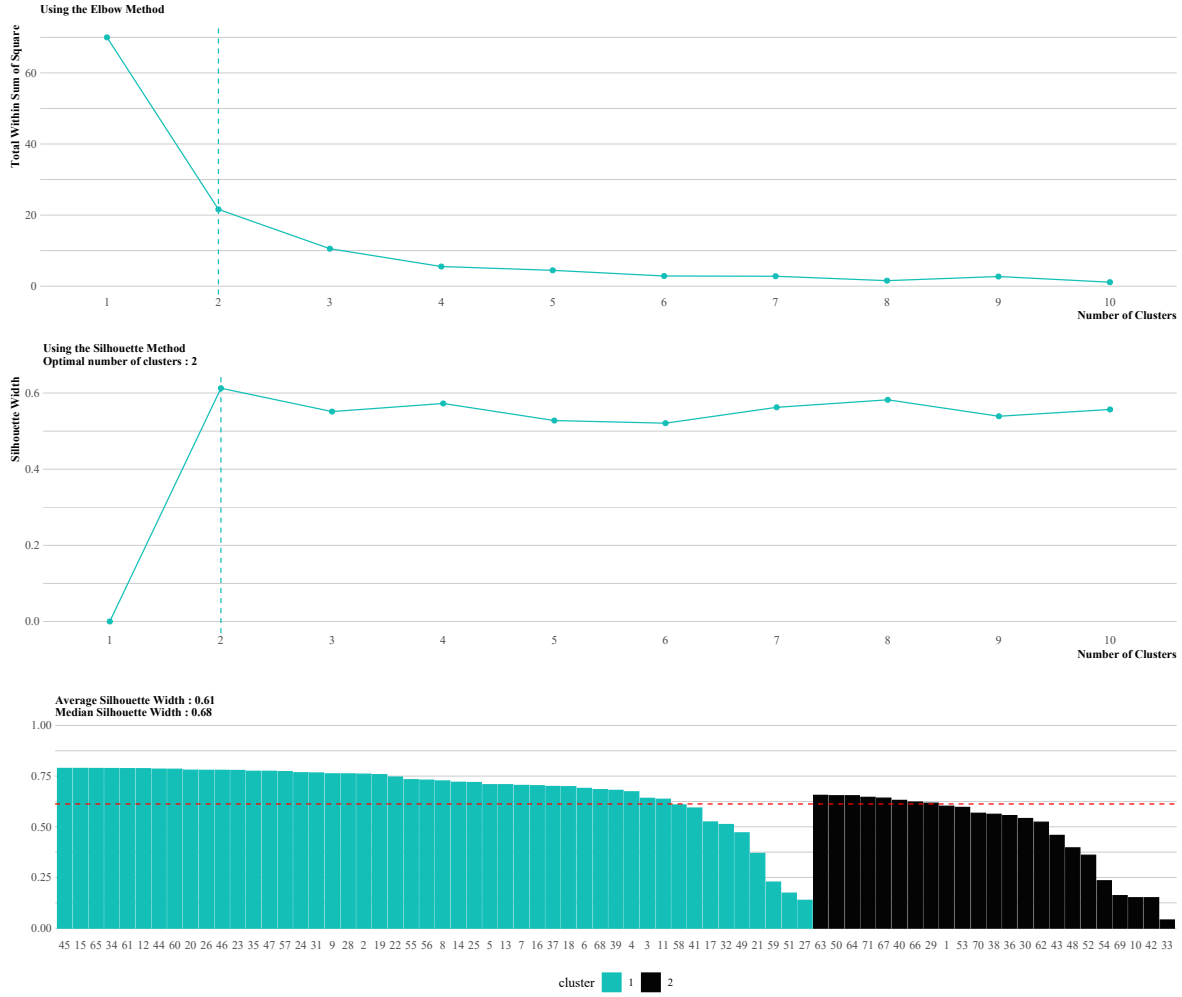

Figure 4.3: Detecting optimal partitions of PCC

Observation 1: Automatic Refactoring achieves higher levels of McCabe Cyclomatic complexity reduction. Consider relevant in Table 4.5, how the mean of code cyclomatic complexity reduction *(Δ VG)* for automatic refactoring is almost three times the reduction when doing manual refactoring. It is also relevant to mention, by looking at Figure 4.4, that only four teams had high complexity levels in their work sessions when doing refactoring using *JDeodorant*. Furthermore, from those, one team had the major software complexity reduction(16.77%), whilst other had near the lowest value of reduction(2.68%) within the automatic refactoring practice. The observation of such different results raised the doubt about the comprehension, focus and behaviour of those two teams in the given task. This demanded further investigation on their efficiency, for which, we provide some evidences later using Figures 4.6 and 4.7.

Observation 2: Manual refactoring practices have higher process cyclomatic complexity. We observe that teams doing manual refactoring almost double the mean of process cyclomatic complexity *(PCC)*, when compared with the ones using the automatic features of *JDeodorant*. Being deprived of the code smell detection plugin, these teams had to do more manual work to potentially achieve the same results as the ones doing automatic refactoring. This suggest that

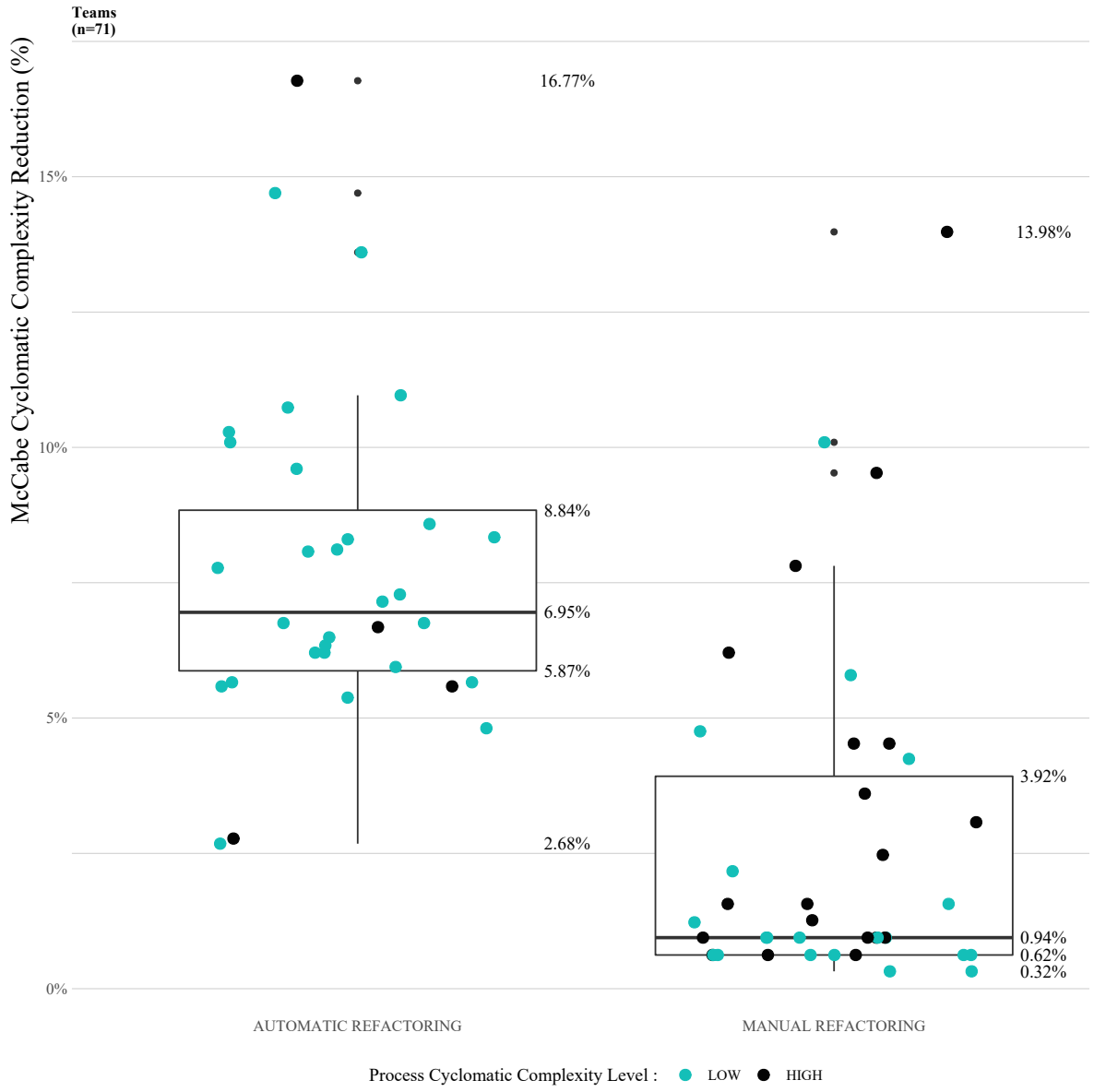

Figure 4.4: Refactoring Practices Comparison

the refactoring plugin was working as expected, thus reducing software complexity with less effort simply because several code snippets may have been introduced automatically.

On the contrary, teams doing the task manually needed to do more code, and therefore, more actions within the IDE to detect and correct the code smells. As shown earlier in section 4.1, manual refactoring tasks can introduce non expected defects in the code and is seen as a practice to avoid.

Figure 4.4 plot the percentage of McCabe Cyclomatic Complexity per method reduction obtained after both refactoring sessions. The different colors plot the different levels of process cyclomatic complexity as discovered from mining each team events log.

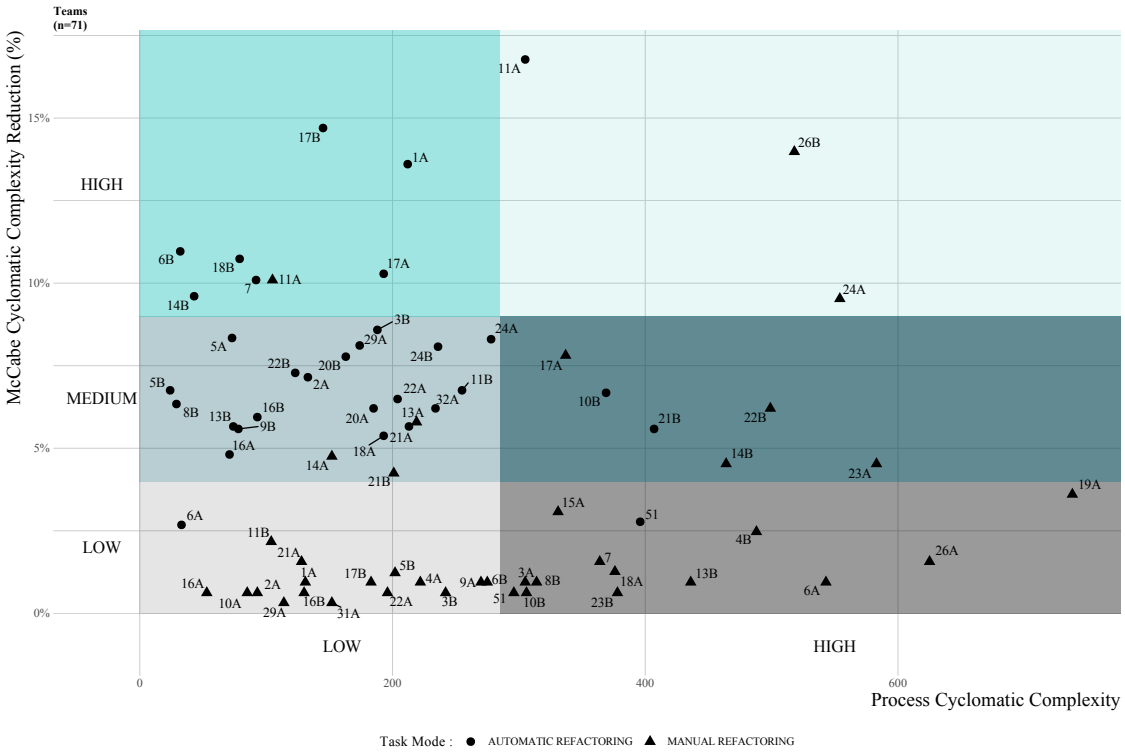

Figure 4.5: Plotting teams according to levels of software and process cyclomatic complexity

Observation 3: Even using JDeodorant, similar work efforts does not mean the same level of gains in software complexity reduction. If it is apparent that, when using *JDeodorant*, the processes tend to have lower levels of complexity and obtained globally more gains in product complexity reductions, the same cannot be said for teams doing manual refactoring. These teams have a more heterogeneous process behavior since they were free to apply refactoring functionalities without any guidelines in detection and correction from a dedicated plugin. Figure 4.5 identifies all teams and distributes them according to their levels of software and process complexity.

From Figure 4.4, we can also observe that the team (11A) with the highest reduction in code complexity ($\approx 16.77\%$), had also a high level of process complexity even if they were using the JDeodorant plugin. We can also identify a team(51) doing automatic refactoring with high levels of process complexity but having instead, very low gains in code cyclomatic complexity reduction ($\approx 2.68\%$). As such we investigated the activities of both teams in order to identify potential reasons for this substantial variation.

Figures 4.6 and 4.7, represent the process flow views of both individual teams regarding the files browsed and/or changed during the refactoring practice[14]. Based on the same values for the activities and paths, we can clearly identify that the team with high gains in *VG* reduction worked in less files (number of nodes) and was focused evenly on all of them (dark blue nodes means more actions on those files).

On the contrary, the team with low gains in *VG*, visited more files but worked frequently on

[14]We acknowledge that the labels in these two diagrams, produced by the Disco tool, are illegible in a printing version. However, since the figures are in vectorial format, they can be *zoomed in* easily if this work is read in its electronic version (pdf), the most probable access medium.

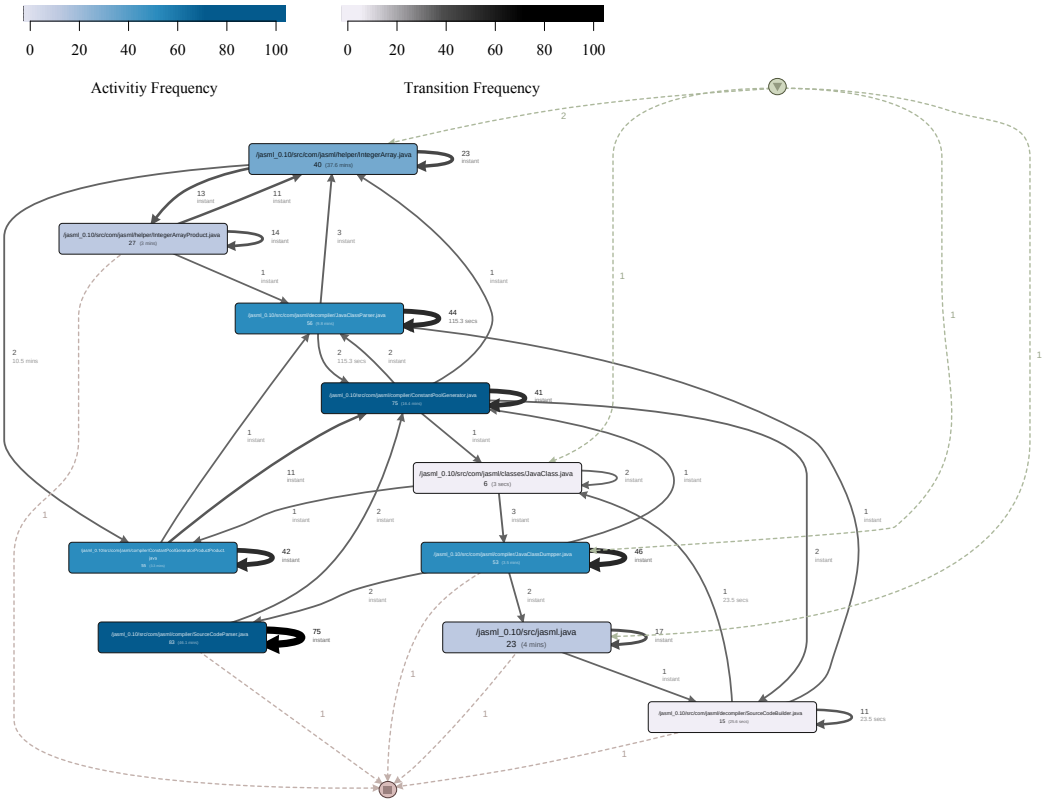

Figure 4.6: Team 11A : High PCC and High VG reduction (20% activities/files, 80% paths)

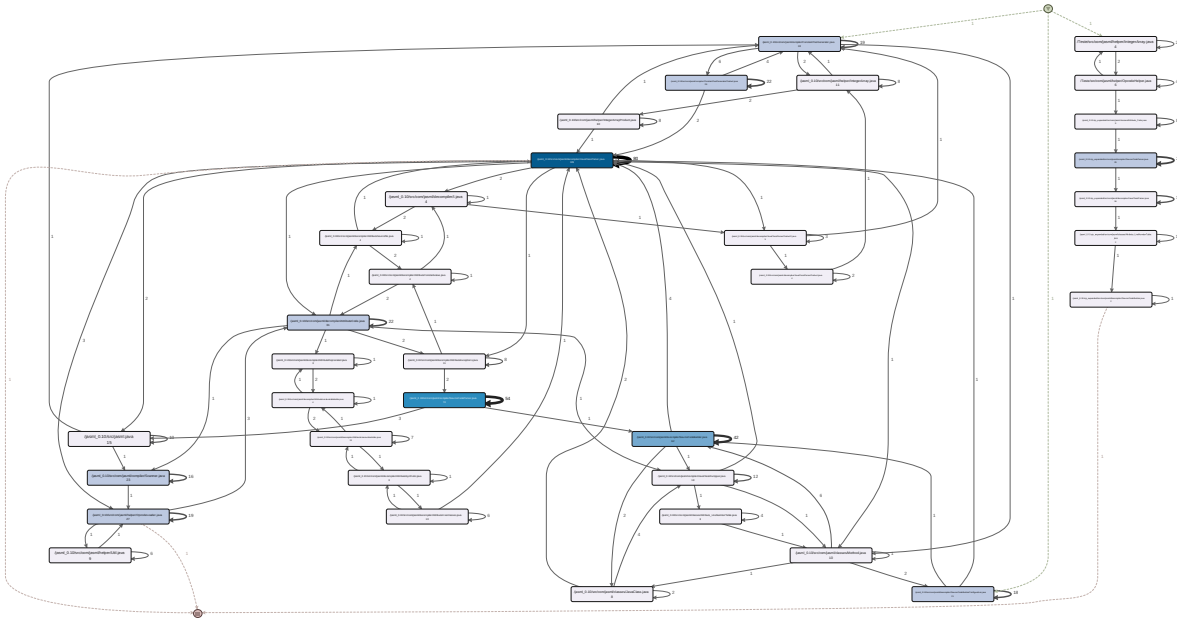

Figure 4.7: Team 51 : High PCC but Low VG reduction (20% activities/files, 80% paths)

only 3 of them. This fuzzy behavior suggests lack of focus and/or knowledge about the task to accomplish, and present a good way to measure efficiency on development teams or individual developers. That can be confirmed by comparing both teams statistics in Figure 4.8, where we present product metrics, process metrics and extended process metrics scaled to represent their position to the mean value of each action for both teams.

We highlight in the extended process metrics the fact that the team with bigger VG

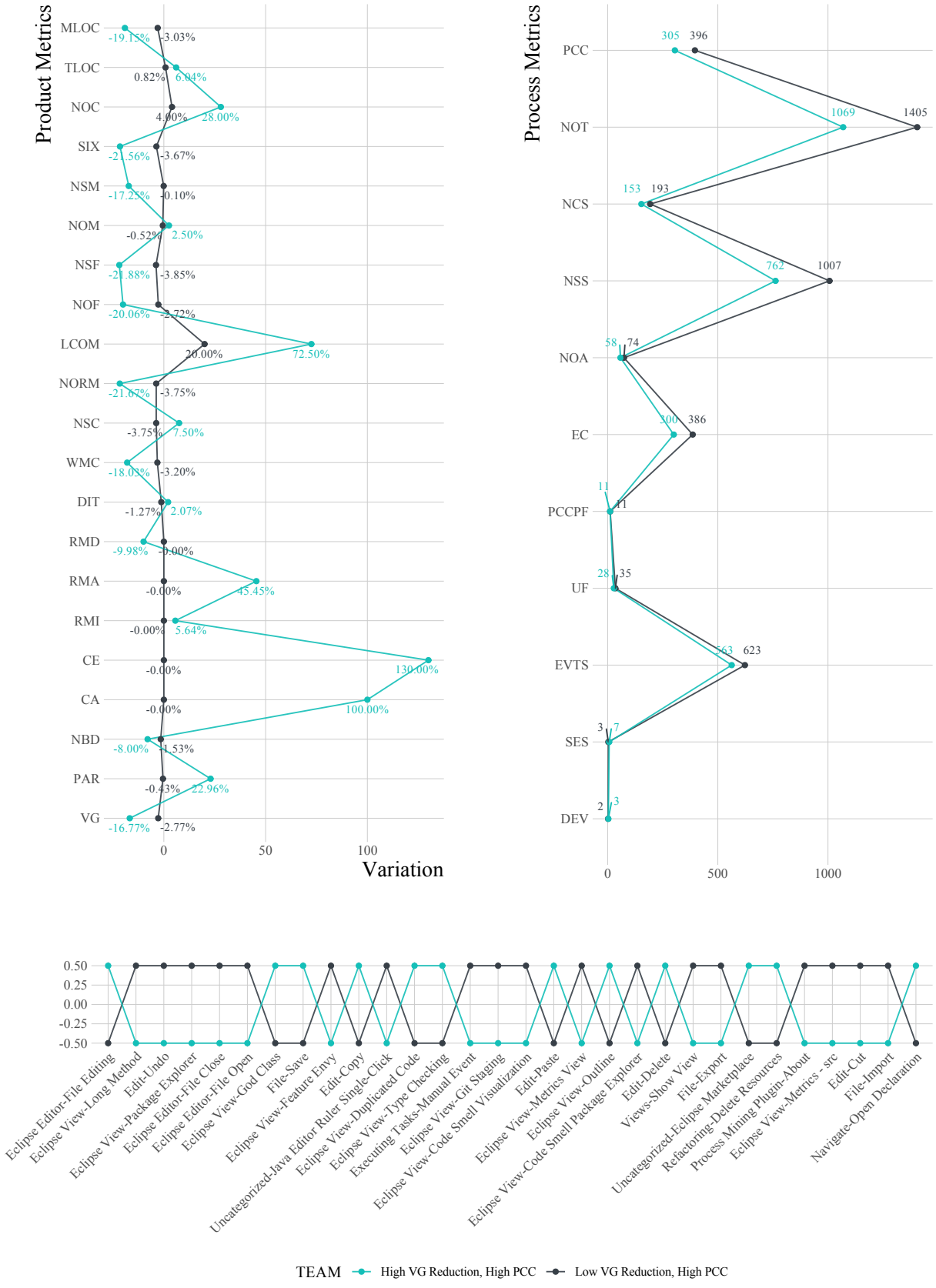

Figure 4.8: Teams(11A vs. 51) with distinct VG variance positioning but similar PCC levels

reduction was the one with less frequencies in commands such as : Undo, Cut, File Open, File Close plus other navigational and less productive actions. This team had also bigger

frequencies in commands to detect and fix code smells, such as: God Class, Duplicated Code and Type Checking. However, the gains in the *VG* reduction were achieved at the cost of increasing 28% the number of classes(NOC) and the lack of cohesion of methods(*LCOM*) by ≈72%. On the process side, despite the fact that this team had more work sessions(7), they touched less files, meaning their activities were less complex, and that is confirmed by the *UF*, *NOA* and *PCC* metrics.

RQ5. Is there any association between software complexity and the underlying development activities in refactoring practices? With the evidences shown in **RQ4** for the two distinct refactoring methods, one may question if the product complexity reduction gains are monotonically correlated with the development activities which originated them.

We used the Spearman correlation coefficient to measure the strength of correlation between metrics of these two dimensions, product and process complexities. This coefficient ranges from -1 to 1, where -1 and 1 correspond to perfect negative and positive relationships respectively, and 0 means that the variables are independent of each other.

To validate our results, we performed a significance test to decide whether based upon this sample there is any or no evidence to suggest that linear correlation is present in the population. As such, we tested the null hypothesis, $\mathbf{H_0}$, and the alternative hypothesis, $\mathbf{H_1}$, to gather indication of which of these opposing hypotheses was most likely to be true.

Let $\mathbf{p_s}$ be the Spearmans' population correlation coefficient both for automatic and manual refactoring, then we can thus express this test as:

$\mathbf{H_0}$: $p_s = 0$: No monotonic correlation is present in the practice.

$\mathbf{H_1}$: $p_s \neq 0$: A monotonic correlation is present in the practice.

Automatic Refactoring. After computing the Spearman correlation coefficient on the subset of teams doing automatic refactoring, and despite the fact that some correlations were slightly negative as we expected, we got no significant statistics on the correlation of *ΔVG* and *PCC* or any other pair of metrics, as shown by Spearmans' *rho* and *p-value* in Table 4.7.

Observation 4: No significant correlation was found between product and process metrics on automatic refactoring practices. Hence, we can say that we cannot reject the null hypothesis, $\mathbf{H_0}$, meaning that a monotonic correlation cannot said to be found between code cyclomatic complexity and process cyclomatic complexity or any other process metric.

Manual Refactoring. When analyzing the dataset with manual refactoring activities, we found that product complexity reduction was moderately correlated with the process cyclomatic complexity and several other metrics process related. Table 4.7 presents Spearmans' *rho* and *p-value*, highlighting the significant correlations[15].

Observation 5: A moderate correlation was found between product metrics and process metrics on manual refactoring tasks. It is relevant to highlight the presence of a moderate positive correlation between the product cyclomatic complexity reduction *(ΔVG)* and the overall

[15]Other product and process metrics were omitted due to the absence of significant correlations

Table 4.7: Spearmans' Correlation - Refactoring Tasks

	Automatic Refactoring ΔVG		Manual Refactoring ΔVG	
Factors	***Spearmans' rho***	***p-value***	***Spearmans' rho***	***p-value***
PCC	-0.02	0.9707	**0.43**	**0.0432***
UF	0.01	0.5218	0.32	0.3427
SES	0.15	0.7489	0.24	0.2814
DEV	-0.05	0.7342	0.03	0.8193
NPER	-0.19	0.4976	**0.32**	**0.0197***
NISP	-0.10	0.6875	**0.35**	**0.0120***
PCCPF	-0.01	0.7787	**0.45**	**0.0059***
NCAT	-0.11	0.6309	**0.39**	**0.0096***
NCOM	-0.05	0.6240	0.42	0.0712

**Statistically significant if p-value < 0.05*

process cyclomatic complexity*(PCC)* and per unique file touched*(PCCPF)*. This means that the more actions the teams have done within the IDE the bigger the gains obtained in complexity reduction.

Observation 6: Weak to moderate correlations were found between product complexity reduction and IDE command categories. Weak to moderate correlations emerge when we pair the product complexity reduction with the number of the IDE command categories*(NCAT)*, IDE perspectives activated*(NPER)* and the number of distinct physical locations from where the task was performed*(NISP)*. Based on the significance tests, we can reject $\mathbf{H_0}$, and accept $\mathbf{H_1}$, meaning that a monotonic correlation exists between code cyclomatic complexity and process cyclomatic complexity as well as with the other highlighted metrics.

Observation 7: No significant correlations were found between any process metrics and product metrics, except for ΔVG. All product and process metrics collected are shown in Tables 4.2 and 4.3.

Figure 4.9 plot only the significant correlations[16] among all those we studied. As expected, process metrics show strong correlations between themselves, however, we find this result obvious and not relevant withing the context of this work.

RQ6. Using only process metrics, are we able to predict with high accuracy different refactoring methods?.

Process metrics have been confirmed as suitable predictors for many software development prediction models. They were found not only suitable, they performed significantly better than code metrics across all learning techniques in several studies [125, 171].

Our goal was to use the process metrics described in Table 4.3, to predict if a refactoring task executed by a group of teams had been done automatically, using the *JDeodorant* features, or manually, using only the Eclipse native functionalities or driven by developers skills. Each subject in our dataset has the class to predict labelled as **AR** and **MR** for automatic and manual refactoring, respectively. In this case, we did not use metrics from Table 4.4 because that

[16]Blank squares means non significant values

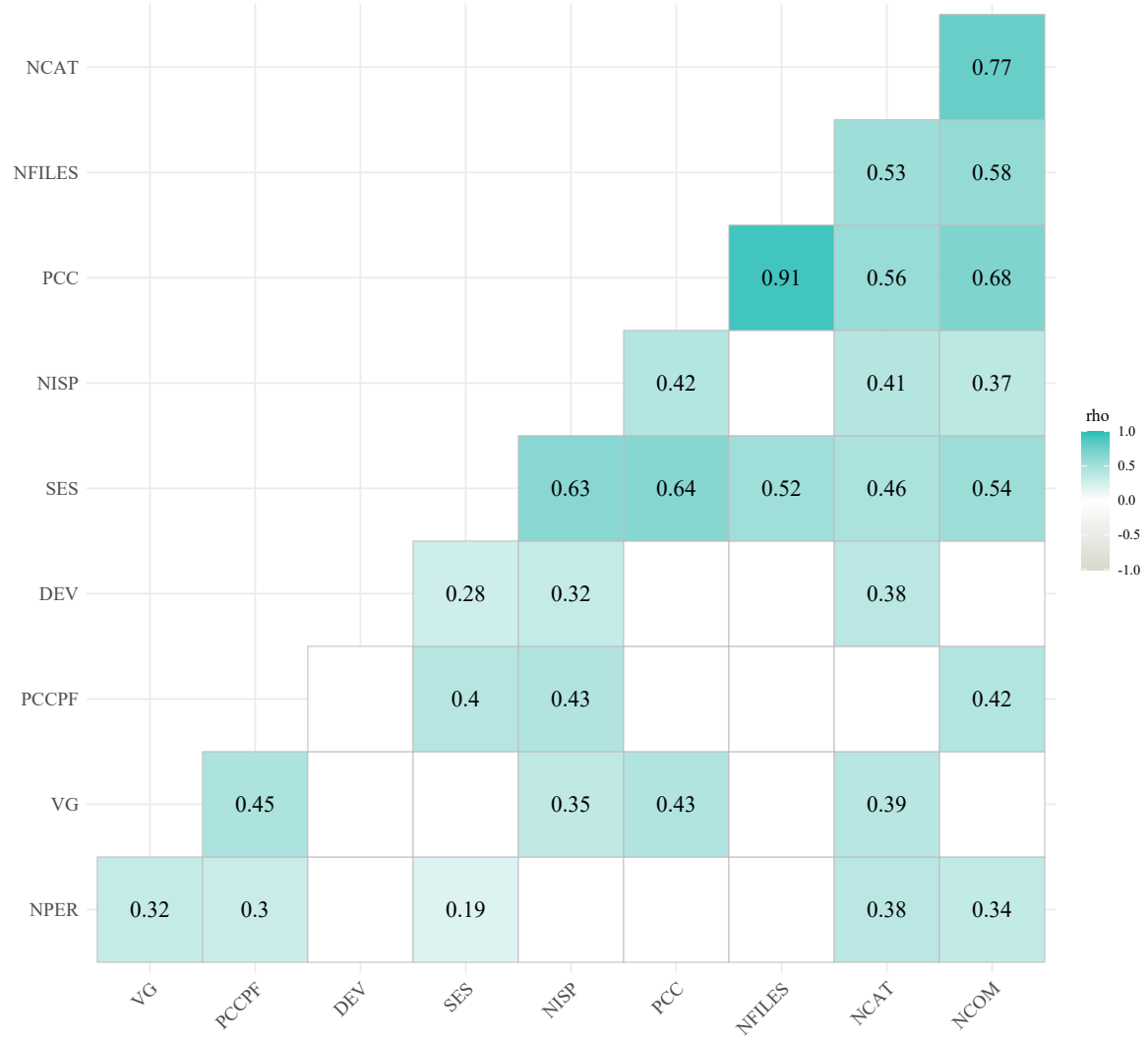

Figure 4.9: Manual Refactoring correlation results

would introduce bias in our models since the process extended metrics can easily be used to understand if developers used or not IDE built in features or their own skills during a refactoring practice.

Table 4.8 present the results for the 5 best models we got out of the ≈30,000 we evaluated on our research. In this context, the machine learning models used were built by assembling and testing supervised or unsupervised algorithms adjusted with feature selection and hyperparameter optimization. From the models built, the ones with higher ROC were chosen. A brief explanation of each algorithm can be found on section C.1 in Appendix C, and the code obtained from training Model 1 is presented in Listing 4.2.

Table 4.8: Detailed Model Evaluation

Model	TP	FP	Pre.	Rec.	F-M.	MCC	ROC	PRC
Model 1, RandomCommittee/RandomForest, Accuracy = 92.95%								
AR	0.906	0.051	0.935	0.906	0.921	0.858	0.983	0.980
MR	0.949	0.094	0.925	0.949	0.937	0.858	0.983	0.987
W. Avg.	0.930	0.075	0.930	0.930	0.929	0.858	0.983	0.984
Model 2, RandomCommittee/RandomForest, Accuracy = 90.14%								
AR	0.875	0.077	0.903	0.875	0.889	0.801	0.939	0.923
MR	0.923	0.125	0.900	0.923	0.911	0.801	0.939	0.948
W. Avg.	0.901	0.103	0.901	0.901	0.901	0.801	0.939	0.937
Model 3, Logistic Model Trees, Accuracy = 90.14%								
AR	0.906	0.103	0.879	0.906	0.892	0.802	0.945	0.938
MR	0.897	0.094	0.921	0.897	0.909	0.802	0.945	0.951
W. Avg.	0.901	0.098	0.902	0.901	0.902	0.802	0.945	0.945
Model 4, RandomSubSpace/REPTree, Accuracy = 88.73%								
AR	0.844	0.077	0.900	0.844	0.871	0.772	0.929	0.907
MR	0.923	0.156	0.878	0.923	0.900	0.772	0.929	0.935
W. Avg.	0.887	0.120	0.888	0.887	0.887	0.772	0.929	0.922
Model 5, Logistic Regression, Accuracy = 83.09%								
AR	0.750	0.103	0.857	0.750	0.800	0.659	0.939	0.940
MR	0.897	0.250	0.814	0.897	0.854	0.659	0.939	0.950
W. Avg.	0.831	0.184	0.833	0.831	0.829	0.659	0.939	0.945

TP-True Positive, **FP**-False Positive, **Pre**-Precision, **Rec**-Recall,
F-M-F-Measure, **MCC**-Matthews Correlation Coefficient,
ROC-Receiver Operating Characteristic, **PRC**-Precision-Recall Curve,
AR-Automatic Refactoring, **MR**-Manual Refactoring,
W. Avg-Weighted Average

Listing 4.2: Best-Fit Model Code for Refactoring Practice Detection

```
1  /** Java code to implement the best model found. */
2
3  /** Attribute Search **/
4  AttributeSelection as = new AttributeSelection();
5  ASSearch asSearch = ASSearch.forName("weka.attributeSelection.GreedyStepwise", new
       ↪ String[]{"-C", "-R"});
6  as.setSearch(asSearch);
7
8  /** Attribute Evaluation and Selection **/
9  ASEvaluation asEval = ASEvaluation.forName("weka.attributeSelection.CfsSubsetEval",
       ↪ new String[]{"-M", "-L"});
10 as.setEvaluator(asEval);
11 as.SelectAttributes(instances);
12
13 /** Reduce Dimensions **/
14 instances = as.reduceDimensionality(instances);
15
16 /** Build Classifier **/
17 Classifier classifier = AbstractClassifier.forName("weka.classifiers.meta.
       ↪ RandomCommittee", new String[]{"-I", "64", "-S", "1", "-W", "weka.classifiers.
       ↪ trees.RandomForest", "--", "-I", "29", "-K", "13", "-depth", "3"});
18 classifier.buildClassifier(instances);
```

Observation 8: Random Forest confirms its accuracy. Random Forest models were found to be the ones with higher accuracy in predicting refactoring opportunities in previous studies [8]. We observe the same behaviour. Random Forest shows twice in the top 5 of our best models, with a **ROC** value of 0.983 and 0.939 for Model 1 and 2, respectively. In both cases, the models were computed by a meta learner which builds an ensemble of randomizable base classifiers, the Random Committee.

Our dataset is not imbalanced, thus, we have almost the same number of subjects for each class, meaning we may use also the **Accuracy** metric to complement our analysis. Model 1 and 2 had respectively, an accuracy of 92.5% and 90.14%.

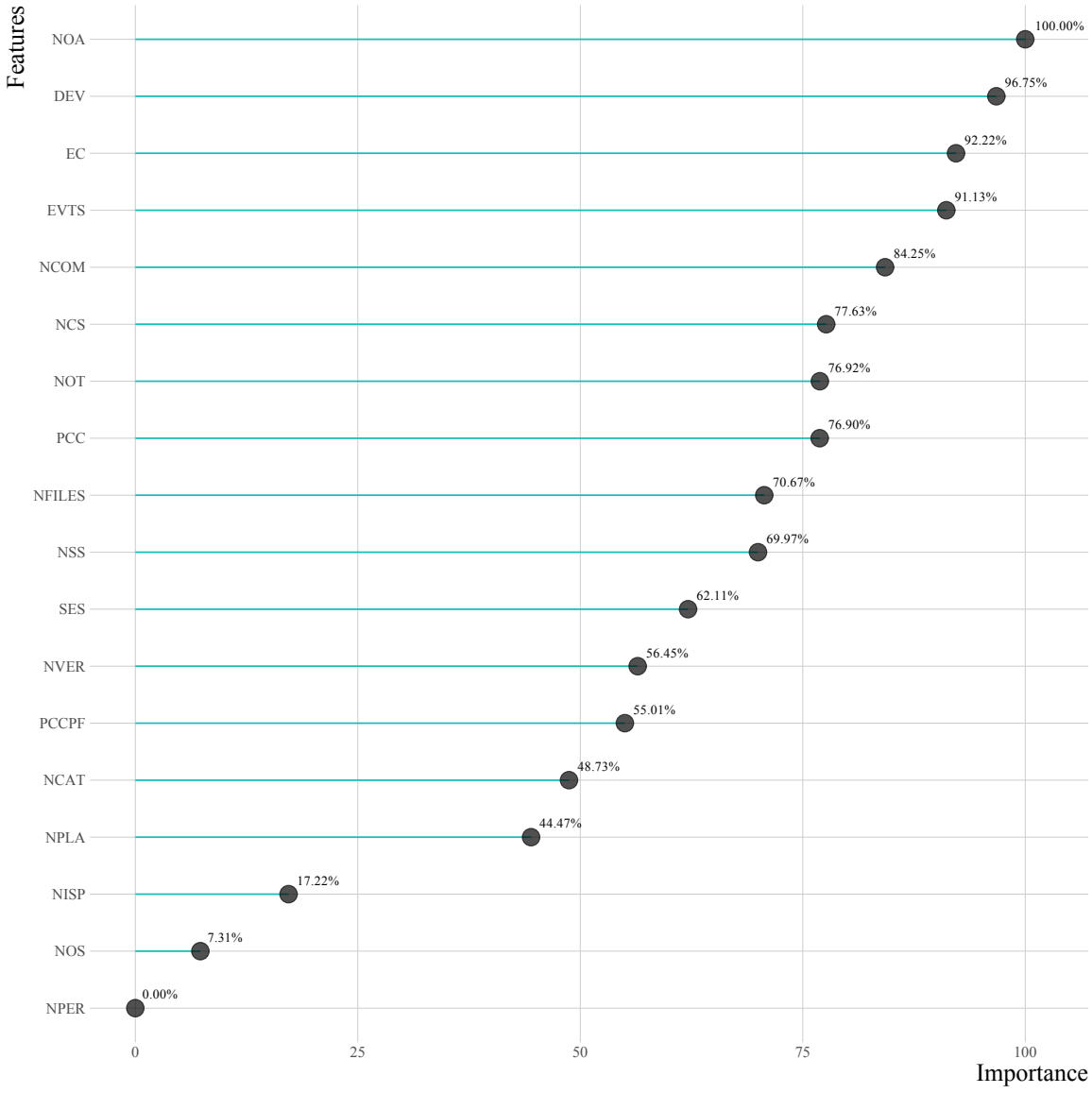

Figure 4.10: Feature importance for models on Table 4.8

During models computation phase, we also assessed which of the features were more or less important to predict the refactoring practices: automatic(**AR**) or manual(**MR**). Figure 4.10 shows their average importance.

Observation 9: Number of Activities, Developers and Commands are the most relevant model features. These features show among the ones with highest importance in the models we computed. We recall that the number of activities (*NOA*) is a composite metric obtained by the process mining extraction plugin using a hierarchical structure composed of the filename, command category and commands issued during the coding phase. Having a mid level importance we find the process cyclomatic complexity and the number of development sessions.

Observation 10: Distinct IDE Perspectives and Operating Systems have almost irrelevant importance. In our models, the different types operating system used by the developers, the different number of IDE perspectives and number of development locations (*NISP*) are irrelevant predictors in modeling the type of refactoring performed. We argue that, particularly the number of different locations from where the developers performed their work require additional research in order to get any generalized conclusions about this insight.

RQ7: Using only process metrics, are we able to model accurately the expected level of complexity variance after a refactoring task?

To answer this **RQ**, we used not only the metrics from Table 4.3, but also the ones from Table 4.4. During our analysis, it was clear that process extended metrics, representing the commands issued by each developer/team, added significant predictive power to the models computed. Therefore, to predict the expected software cyclomatic complexity we needed to include individual commands frequencies in addition to the process metrics used in previous **RQ**. By doing this we were able to achieve models with higher accuracy and good ROC values. However, in general, these models have lower accuracy than the ones in **RQ6**.

Table 4.9 shows the top five models computed to predict the complexity level gains obtained after a refactoring session, either using a dedicated plugin or simply by using Eclipse features.

Observation 11: Locally Weighted Learning combined with a Decision Table outperforms Random Forrest. Contrary to the previous RQ, in this case the best model is not based on a Random Forrest algorithm. However, the latter show as the second best model in terms of accuracy. The Locally Weighted Learning method uses an instance-based algorithm to assign instance weights which are then used by a specified weighted instances handler. It uses a stack of methods, initially a cluster like mechanism such as the LinearNNSearch and then a Decision Table to classify the outcome. This shows up at no surprise since Decision Tables use the simplest hypothesis spaces possible and usually outperform state-of-the-art classification algorithms.

A brief explanation of each algorithm can be found on section C.1 in Appendix C, and the code obtained from training Model 1 is presented in Listing 4.3.

Observation 12: Teams with LOW level of software complexity gains are frequently spotted with higher F-Measure and ROC values. Our models perform better in detecting subjects achieving low levels of complexity reduction. These are the most critical cases, as such, a software development project manager can quickly detect the teams or individuals responsible for those outcomes and implement actions to bring the project under acceptable quality parameters.

Observation 13: Process extended metrics have in general higher importance than

Table 4.9: Detailed Model Evaluation

Model	TP	FP	Pre.	Rec.	F-M.	MCC	ROC	PRC
Model 1, LWL/LinearNNSearch/DecisionTable, Accuracy = 94.36%								
LOW	0.968	0.000	1.000	0.968	0.984	0.972	0.991	0.992
MEDIUM	1.000	0.095	0.879	1.000	0.935	0.892	0.994	0.992
HIGH	0.727	0.000	1.000	0.727	0.842	0.832	0.992	0.967
Weighted Avg.	0.944	0.039	0.950	0.944	0.942	0.917	0.993	0.988
Model 2, Bagging/RandomForest, Accuracy = 83.09%								
LOW	0.839	0.075	0.897	0.839	0.867	0.771	0.938	0.94
MEDIUM	0.828	0.095	0.857	0.828	0.842	0.737	0.971	0.945
HIGH	0.818	0.083	0.643	0.818	0.720	0.668	0.971	0.827
Weighted Avg.	0.831	0.085	0.841	0.831	0.834	0.741	0.957	0.926
Model 3, KStar, Accuracy = 78.87%								
LOW	0.935	0.225	0.763	0.935	0.841	0.707	0.948	0.951
MEDIUM	0.862	0.143	0.806	0.862	0.833	0.713	0.945	0.915
HIGH	0.182	0.000	1.000	0.182	0.308	0.398	0.982	0.904
Weighted Avg.	0.789	0.157	0.818	0.789	0.755	0.661	0.952	0.929
Model 4, RandomCommittee/REPTree, Accuracy = 74.64%								
LOW	0.903	0.300	0.700	0.903	0.789	0.603	0.895	0.873
MEDIUM	0.759	0.143	0.786	0.759	0.772	0.619	0.886	0.847
HIGH	0.273	0.000	1.000	0.273	0.429	0.491	0.932	0.738
Weighted Avg.	0.746	0.189	0.781	0.746	0.726	0.592	0.897	0.842
Model 5, LWL/LinearNNSearch/DecisionTable, Accuracy = 71.83%								
LOW	0.871	0.300	0.692	0.871	0.771	0.569	0.843	0.803
MEDIUM	0.759	0.190	0.733	0.759	0.746	0.565	0.800	0.729
HIGH	0.182	0.000	1.000	0.182	0.308	0.398	0.823	0.541
Weighted Avg.	0.718	0.209	0.757	0.718	0.689	0.541	0.822	0.732

TP-True Positive, **FP**-False Positive, **Pre**-Precision, **Rec**-Recall,
F-M-F-Measure, **MCC**-Matthews Correlation Coefficient,
ROC-Receiver Operating Characteristic, **PRC**-Precision-Recall Curve,
LOW-Low level of Cyclomatic Complexity,
MEDIUM-Medium level of Cyclomatic Complexity,
HIGH-High level of Cyclomatic Complexity,
W. Avg-Weighted Average

process standard metrics. From Figure 4.11 we can understand that 18 out of 30 metrics are related with the commands issued by the developers. In general, these metrics have also higher importance in the models. It is not surprising to find methods and class extraction commands in the top of the list, with ≈86% and ≈56% importance, respectively. It was however unexpected to find project export actions being so relevant (≈70%).

Listing 4.3: Best-Fit Model Code for Expected Cyclomatic Complexity Level Setection

```
/** Java code to implement the best model found. */

/** Attribute Search **/
AttributeSelection as = new AttributeSelection();
ASSearch asSearch = ASSearch.forName("weka.attributeSelection.GreedyStepwise", new
    ↪ String[]{"-C", "-R"});
as.setSearch(asSearch);

/** Attribute Evaluation and Selection **/
ASEvaluation asEval = ASEvaluation.forName("weka.attributeSelection.CfsSubsetEval",
    ↪ new String[]{"-L"});
as.setEvaluator(asEval);
as.SelectAttributes(instances);

/** Reduce Dimensions **/
instances = as.reduceDimensionality(instances);

/** Build Classifier **/
Classifier classifier = AbstractClassifier.forName("weka.classifiers.lazy.LWL", new
    ↪ String[]{"-K", "60", "-A", "weka.core.neighboursearch.LinearNNSearch", "-W", "
    ↪ weka.classifiers.rules.DecisionTable", "--", "-E", "auc", "-S", "weka.
    ↪ attributeSelection.GreedyStepwise", "-X", "2"});
classifier.buildClassifier(instances);
```

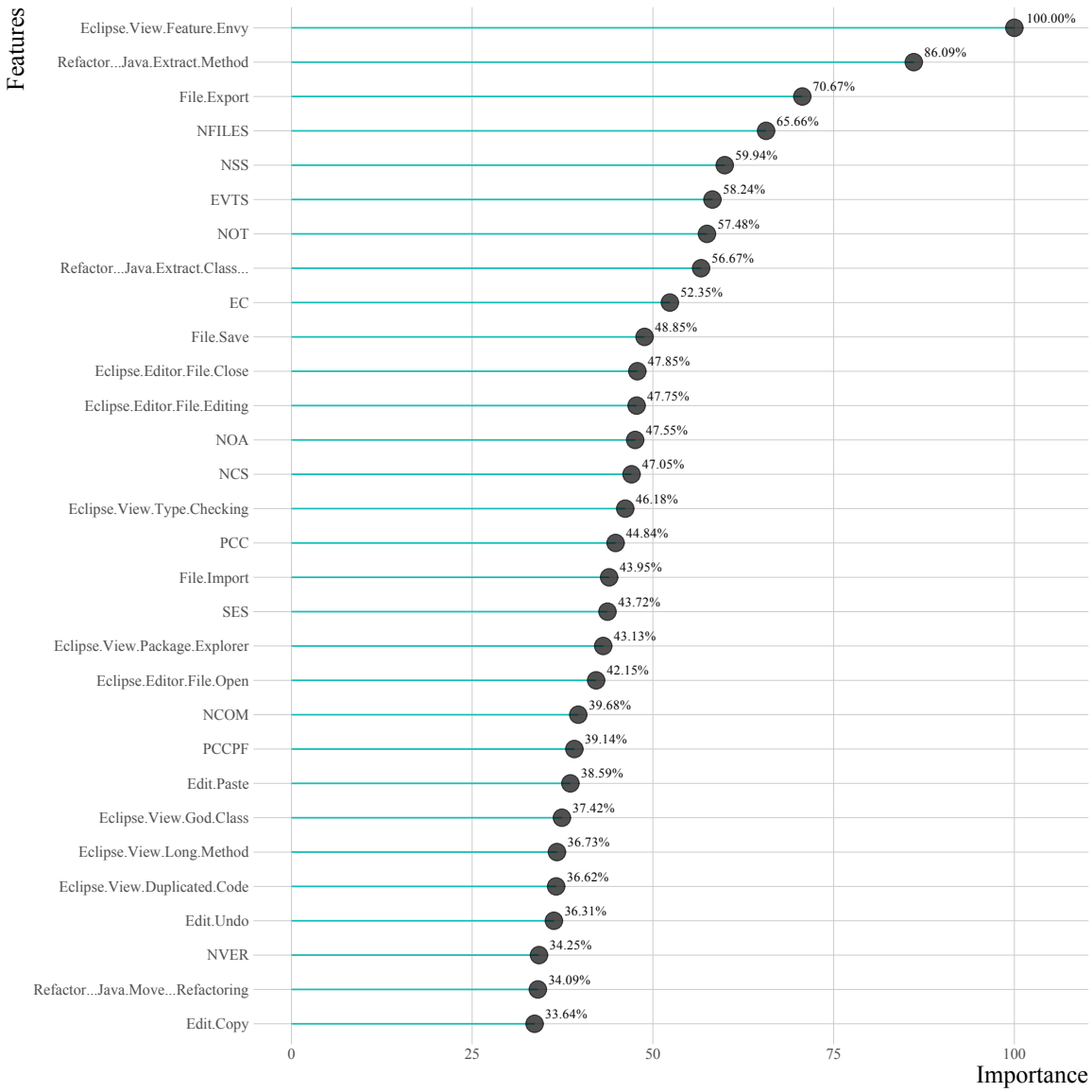

Figure 4.11: Feature importance for models on Table 4.9 (Top 30 only)

4.5 Threats to Validity

The following types of validity issues were considered when interpreting the results from this article.

Construct Validity. We acknowledge that this work was supported by an academic environment and using subjects with different maturity and skills which we did not assessed deeply upfront. Additionally, although some work has been done in this domain, we are still just scratching the surface in mining developers' activities using process mining tools. Some of these tools are not ready yet to automate the complete flow of: collecting data, discover processes, compute metrics and export results. As a consequence, the referred tasks were mostly done manually, thus, introducing margin for errors in the data metrics used in the experiment. To soften this, and to reduced the risk of having incoherent data, we implemented the validation of metric values from multiple perspectives. Another possible threat is related

with the event data pre-processing tasks before using the Process Mining tools to discover the processes and associated metrics. Events were stored initially in a database, and from there, queries were issued to filter, aggregate and select some process related metrics. We used all the best practices in filtering and querying the data. However, there is always a small chance for the existence of an imprecise query which may have produced incorrect results and therefore, impacted our data analysis.

Internal Validity. We used a cluster analysis technique supported by the Elbow and Silhouette methods. This was used to partition the subjects according to different software and process cyclomatic complexity levels. Even if this is a valid approach, other strategies could have been followed, thus, results could vary depending on the alternative methods used, since the models computed to address RQ6 and RQ7 make use of this data partition approach. As mentioned earlier, our population was not very large and we had to use it for training and test purposes. As such, our prediction models were all trained using k-fold cross validation and using feature selection methods.

External Validity. We understood from the beginning there was a real possibility that events collected and stored in CSV/JSON files on developers' devices could be manually changed. We tried to mitigate this threat of having data tampering by using a hash function on each event at the moment of their creation. As such, each event contain not only information about the IDE activities, but also a hash code introduced as a new property in the event for later comparison with original event data. For additional precautions regarding data losses, we implemented also real-time event streaming from the IDE to a cloud event hub.

Our initial dataset contains events collected from a group of teams when performing an academic exercise. Each user was provided with a *username* and *password* to enter in the Eclipse Plugin. With this approach, we can easily know which user was working on each part of the software and their role in the whole development process. However, we cannot guarantee that each developer used indeed their own *username*. This does not cause any invalid results in the number of activities for example, but may introduce some bias in the number of developers per team[17].

Conclusion Validity. We performed an experiment using data from 71 software teams executing well defined refactoring tasks. This involved 320 sessions of work from 117 developers. Since this is a moderate population size for this type of analysis, we acknowledge this may be a threat to generalize conclusions or make bold assertions. The Spearmmans' correlation, a nonparametric measure (therefore having less statistical power) of the strength and direction of association that exists between two variables, was done on 32 and 39 teams for automatic and manual refactoring tasks respectively. These figures, although valid, are close to the minimum admissible number of subjects for this type of analysis. Nevertheless, the insights we unveil in this work should be able to trigger additional research in order to confirm or invalidate our initial findings.

[17] A metric used on almost all **RQs** and identified as having a high importance

4.6 Summary

Software comprehension and maintenance activities, such as refactoring, are said to be negatively impacted by software complexity. The methods used to measure software product and processes complexity have been thoroughly debated in the literature. However, the discernment about the possible links between these two dimensions, particularly on the benefits of using the process perspective, has a long journey ahead.

In this work, we tried to understand deeper the liaison of process and software complexity. Moreover, we assessed if process driven metrics and IDE issued commands are suitable to build valid models to predict different refactoring methods and/or the expected levels of software cyclomatic complexity variance on development sessions.

We mined source code metrics from a software product after a quality improvement task was given in parallel to **(117)** software developers, organized in **(71)** teams. Simultaneously, we collected events from their IDE work sessions **(320)** and used process mining to model their processes and extract the correspondent metrics.

We found that teams using a plugin for refactoring ***(JDeodorant)*** reduced software complexity more effectively and with simpler processes than the ones that performed refactoring using only ***Eclipse*** native features. We were able to find moderate correlations **(≈43%)** between software cyclomatic complexity and process cyclomatic complexity. Using only process driven metrics, we computed ≈30,000 models aiming to predict the type of refactoring method (automatic or manual) teams had used and the expected level of software cyclomatic complexity reduction after their work sessions. The best models found for the refactoring method and cyclomatic complexity level predictions, had an accuracy of **92.95%** and **94.36%**, respectively.

Main conclusions. To the best of our knowledge, this is the first work where, using proven process mining methods, process metrics were gathered and combined with product metrics in order to understand deeper the liaison of product and process dimensions, particularly the cyclomatic complexities. Furthermore, it brings to the attention of researchers the possibility to adopt process metrics extracted from the IDE usage as a way to complement or even replace product metrics in modeling the development process.

We can't compare our work to any previous studies, however, with a small set of features, we were able to unveil important correlations between product and process dimensions and obtain good models in terms of accuracy and ROC when predicting the type of refactoring done or the expected level of cyclomatic complexity variance after multiple sessions of development. We used a refactoring task as our main use case, however, by taking a snapshot of product and process metrics in different moments in time, one can measure other development practices the same way.

We have also demonstrated the feasibility of an approach that allows building cross-cutting analytical models in software projects, such as the one we used for detecting manual or automatic refactoring practices. Events from the development tools and support activities can be collected, transformed, aggregated, and analyzed with fewer privacy concerns or technical constraints than source code-driven metrics. This makes our approach agnostic to programming languages, geographic location, or development practices, making it suitable for challenging

contexts such as in modern global software development projects. Initial findings are encouraging, and lead us to suggest practitioners may use our method in other development tasks, such as, defect analysis and unit or integration tests.

Relevance for practitioners. This approach can be particularly relevant in cases where product metrics are not available or are difficult to obtain. It can be also a valid approach to measure and monitor productivity within and between software teams. As we showed by analyzing the sessions complexity and the software cyclomatic complexity variance, non efficient teams can easily be detected. Our method easily support real-time data collection from individuals located in different geographic zones and with a multitude of development environments. Because the data collection is not dependent on code repositories and is decoupled from check-ins and/or commits, process and code analysis can be performed before repositories are updated. Development organizations can leverage this approach to apply conformance checking methods to verify the adherence of developers' practices with internally prescribed development processes. This facilitates mainly the detection of low performance practices and may trigger quick correction actions from project managers.

We also consider that the following aspects deserve further research efforts:

- **Software Repository Diversity.** Traditional software repositories have limitations and imprecisions. To expand the analytics coverage on the mining of software development processes, we should explore non trivially used repositories, such as the IDE. This is particularly interesting to drive studies aiming to combine development perspectives: i) product quality and ii) the underlying development process.

- **Software Development Process Mining Pipeline.** Many process mining tools are not ready for non-human intervention. Due to this reality, many metrics in this work had to be extracted semi-automatically, using a tool but not dispensing user interaction. This is a strong limitation in advancing research based on event data and current process mining methods. A microservices-based architecture seems to be a good alternative for building a coherent pipeline for software development process mining.

- **Data Sharing.** Research combining software product and process data is scarce and experiments in this area are difficult to design and execute. To mitigate this problem, we expect an increment in shared datasets containing this hybrid data, providing that privacy and/or anonymity on sensitive information is guaranteed.

Part III.

Towards the Prescriptive Commitment

PART I : FUNDAMENTALS

Introduction
Chapter 1

State-of-the-Art
Chapter 2

PART II : SOFTWARE PROCESS IMMERSION

Assessing Teams' Efficiency
Chapter 3

Unveiling Process Insights
Chapter 4

PART III : TOWARDS THE PRESCRIPTIVE COMMITMENT

Practices and Fingerprints
Chapter 5

PART IV : CONCLUSION

Conclusions and Future Work
Chapter 6

This Part demonstrate the benefits of mining software development sessions by combining process mining and text mining towards the search and allocation of optimal resources for modern software projects.

Chapter 5.

Practices and Fingerprints

Contents

This chapter evaluates our approach in a real development scenario in order to provide evidences that software development sessions are as regular as a natural language, thus allowing for the detection of developers' fingerprints and practices.

Companion Soundtrack: Time - Hans Zimmer (Inception Soundtrack)

"You can discover more about a person in an hour of play than in a year of conversation."

—Plato[1]

5.1 Introduction

5.1.1 Motivation

Software development can be characterized as a socio-technical phenomenon [71]. Understanding the actual dependencies between development tasks and teams' behaviors to fulfill them is a serious challenge for most software project managers concerned with the allocation and coordination of resources [88]. Being able to group developers with similar behaviors, for instance, based on the time they spent on each activity or working on a specific artifact, is a step forward in that understanding. This requires analyzing developers' traces (i.e. executed actions/commands) within the IDE.

Traditional process mining techniques come to the rescue of such concerns. However, within the software development context, the latter usually assumes a structured and noise-free input and produces spaghetti-like processes. As such, a lot of variances may mislead the results and correspondent interpretation [91]. Process variant analysis is a research stream within the process mining domain. In the last decade, several novel approaches to effectively mine process variants have been proposed [26, 28, 198]. The latter evolved to detect the existence of similarities and differences in behaviors within a common business process, which can be considered as "fingerprints" left by process instances [46, 207].

Applying process mining algorithms on large event logs, containing a significant number of cases and events, usually requires the use of powerful computational systems and, even then, may lead to long processing times. Process variant comparison techniques, in particular due to massive manipulation of vectors and matrices, are computationally heavy. Software development event logs, generated from IDE usage, often have events at the thousands, hundreds of different activities, hierarchical states, and many different resources associated with events. Therefore, the aforementioned performance problem is usually noticeable. Natural language techniques can mitigate it by performing initial filtering, aggregation of events and in finding local regularities. Even if event aggregation is not desired in mining processes, the trade-off between the practical aspects versus the accuracy of certain algorithms should be carefully evaluated in the software development realm.

In this paper, we propose an approach to profile developers using a stack of text mining to express developers' fingerprints, and process mining to discover, model and assist in hypothesis evaluation regarding their workflows. We used events collected from the IDE during development sessions as input for the unsupervised learning techniques and process mining algorithms. Additionally, since the process of coding can be represented as a grammar with a specific semantic [90], we find it useful to assess how similar this grammar is to a natural language, and in finding optimal parameters for the text mining algorithms.

[1] Athenian philosopher during the Classical period in Ancient Greece, founder of the Platonist school of thought, and the Academy, the first institution of higher learning in the Western world.

A development session executed by one developer at his/her IDE can be considered an instance of a process, where the goal is to produce a software product or maintain an existing one. Its workflow of activities depends on many factors, such as the development methodology, program design or individual experience [37]. Furthermore, developers are usually free to produce code without a referential model or guidelines on how to execute the coding tasks and, most often without any intelligent guidance from traditional development tools. This poses challenges when one wants to detect similarly or deviating programming profiles to assess productivity and optimize resource planning.

To validate our approach for profiling developers, while controlling for spurious effects, we performed a controlled experiment where, in the realm of a Python programming contest, a group of developers had the same well-defined set of requirements specifications and a well-defined sprint schedule. Events were collected from the PyCharm IDE, and from the Mooshak automatic judge where subjects checked-in their code stepwise.

5.1.2 Students as Surrogates for Professional Software Developers

In this study we use students as surrogates for professional software developers. Therefore it is worth reviewing the discussion in the literature on using students as surrogates for professionals.

Almost half a century ago, the practice of using students in research was already widespread, due to the convenience of their availability and usual willingness to participate in experiments. For instance, in consumer behavior (marketing) studies, researchers tested whether students could be used as consumer surrogates, but results were inconclusive [63, 81, 188]. Also since the seventies, as reported in [175], students have been used as surrogates for managers on Decision Support Systems (DSS). The same study reports that undergraduate students were more used than graduate students, which could be a validity hindrance in that case, since graduate students are more closer to managers in age, maturity and education.

Students have also been extensively used as surrogates in Software Engineering studies. For instance, a study carried out with students on detection methods for software requirements inspections [168] was replicated with similar results using professionals as subjects [167]. Another study on lead-time impact assessment for software development projects did not find significant differences between students and professionals [92]. In [182], the performance in PSP improvement tasks was compared between freshmen students, graduate students, and industry people, and again no significant differences were found between the three groups. Two separate studies in Requirements Engineering provided somehow complementary conclusions. While in [22] definitive conclusions about the suitability of students in projects could not be drawn, in [191] the authors argue that it may be possible to influence students to provide answers that are in line with industrial practice, although it was not clear under which conditions could that influence be exerted in empirical investigations. A systematic literature review on using students as surrogates for professionals can be found in [115]. The author concludes that many factors influence the results of experimental studies such as the number of subjects, nature of tasks and previous experience on that, motivation levels of subjects, training provided, and incentives for participation in the experiment. In other words, the appropriateness

of students as surrogates for professionals depends on current study conditions. In section 5.5, we argue why this may hold in our study.

5.1.3 Contributions

The main objectives for this work are the following:

- To evaluate if software development sessions can be mined as any natural language;
- To assess if coherent development fingerprints can be discovered from an event log containing developer's IDE interactions and submission of answers to several coding problems;
- To appraise the impact of individual behaviors in the outcome of a programming task given a group of developers.

The remainder of this paper is organized as follows: section 5.2 provides background related to the research area and emphasizes the need for the proposed approach. Subsequent sections, outline the related work in section 5.2.4, detail the methodology and experiment setup in section 5.3 and present the results, its corresponding analysis and implications in section 5.4. Threats to validity are presented in section 5.5 and the concluding comments and future work in section 5.6.

5.2 Background

5.2.1 Language Models

Natural languages (e.g., English, Portuguese, etc.) possess a rich vocabulary and therefore are complex and powerful. A programming language or a sequence of development actions in plain English, as seen in Figure 5.1[2], is an artificial language but is expected to follow the same principles of a natural language. The rationale is that although a given piece of software is written with an artificial language, it is a natural product of the human mind as prose or poetry in natural language [90]. As such, it is also amenable to statistical analysis like the ones performed in the area known as "text mining", where Natural Language Processing (NLP) algorithms and analytical methods are used.

We argue that development sessions viewed as a sequence of actions like those in Figure 5.1 and represented by a well-defined vocabulary, can be regarded from the same perspective. In this paper, we describe a novel method to detect different developers' profiles based on models built from development interactions using n-gram probabilistic language models [54]. Furthermore, we combine these unsupervised learning models, which present a good fit in capturing local regularities in text data, and process mining algorithms, which are known to perform well in the modelling of complex business processes.

[2]This word cloud, where the size of each word is proportional to its relative frequency, was generated from data collected during the validation experiment of our proposed approach.

Figure 5.1: Word cloud with example of frequent activities on interactions in PyCharm and submissions to Mooshak

5.2.2 Topic Modeling

Understanding unstructured data is a major challenge in software development, and having a predefined data model is not a common scenario when dealing with such type of data. Moreover, those data are typically text-heavy. As such, topic modeling has become one of the most used methods to mine software repositories [42].

Topic modeling is a method for unsupervised classification of documents, by modeling each document as a mixture of topics and each topic as a mixture of words [159]. Despite some limitations, such as the order of the documents, it is frequently used to build models from unstructured textual data, as it presents an effective means of data mining where subjects represent documents or even a textual representation of actions executed in certain contexts [5].

Within the most prevalent methods used to mine software repositories, we find algorithms such as LDA and many of its variations, Latent Semantic Indexing (LSI), Latent Semantic Analysis (LSA), Probabilistic Latent Semantic Indexing (PLSI) and Independent Component Analysis (ICA) [24, 43]. These algorithms are used to cluster documents, identify features, derive source code metrics for bugs prediction, assess code evolution, trace links between pairs of artifacts and detect code clones, among other things [42, 135].

5.2.3 Software Development Process Mining

Process modeling is a persistent topic in the research literature concerned with software development practices. The analysis of fingerprints in event logs [198], the discovery of deviating cases using trace clustering [91] and mining of sequences of developers interactions [53] are examples of topics covered by researchers to overcome or mitigate recurrent problems. However, often the suggested solutions are complex and difficult to automate in a coherent software development process mining pipeline. These constraints led researchers to highlight that software analytics does not need to be hard and, on the contrary, it can and should be simplified [4, 68, 69].

In Table 5.1, we present a comparison of typical text mining characteristics and purposes, along with how we view topic modeling applied in software development process mining.

Table 5.1: Traditional Text Mining (TM) vs. Software Development Process Mining (SDPM)

	TM	**SDPM**
Inputs		
Corpus	Documents/Articles	Development Work Sessions
Document	Mixture of Topics	Mixture of Behaviours
Topic	Frequent Words/Terms	Frequent Actions/Commands
Outputs		
Discovers	Distinct Subjects/Topics	Development Patterns
Usefulness	Identify Social Trends Frame Research Interests Sentiment Analysis	Optimize Resource Allocation Detect Practices Deviations Forensic Project Analysis

Software developers execute a stream of actions/commands when using their IDE. Those commands, seen as a work session, can also be represented textually as a narrative along the timeline. We expect that stream to contain the semantics required to identify different developer profiles. Therefore, logs containing a sequence of IDE commands/actions can be mined with topic modeling as any other document would be in searching for different topics. In this context, we are searching for different behaviors such as programming styles or patterns of IDE usage.

5.2.3.1 Preliminary definitions

To justify the usefulness of collecting IDE events, and provide context to our proposal, we introduce in this section some preliminary definitions required to understand concepts such as development actions, development sessions, development actions repository and development profiles.

Definition 1. Development action

- *A development action is an event defined as a tuple (a, c, t, (p_1,v_1),...,(p_n,v_n)) where a is the command action or process activity name, c is a development session or case id, t is the timestamp and the set (p_1,v_1),...,(p_n,v_n) (where $n \geqslant 0$) contains the event or case properties/attributes and corresponding values, such as developer location, operating system or IDE type.*
- *A development action is defined by a token t included in the development session vocabulary to be formed by a set containing all the possible IDE commands, denoted by V:*

  $V = (t_0,t_1,...t_n) : \forall t \in V, t = \langle ide\_command\_or\_activity \rangle$

Definition 2. Development session

- *A development session is a trace, defined by a non-empty sequence $\sigma=e_1,...,e_n$ of command actions such that $\forall i,j \in [1..n]$ $e_i.c=e_j.c$.*
- *A development session is defined by a sentence formed by a set of tokens from vocabulary V, denoted by:*

  $\omega = (t_0,t_1,...t_n) : \forall t \in \omega, t \in V$

Definition 3. Development actions repository

- *A repository of actions or event log is a set of development actions mapped to a variable number of development sessions, defined as* $L=\sigma_1,...,\sigma_n$.
- *We consider an event log a set of sequential tokens t from the vocabulary V, where t can be repeated.*

Definition 4. Development profile

- *An event log L can be partitioned into a finite set of groups called process variants or, in our case, profiles or fingerprints,* $\varsigma_1,\varsigma_2,...,\varsigma_n$, *where* $\exists p$ *such that* $\forall\ \varsigma_k$ *and* $\forall\ \sigma_i,\sigma_j \in \varsigma_k$, $\sigma_i.p=\sigma_j.p$.

The definition of process variant emphasizes that process executions in the same group must have the same value for a given attribute, and each process execution belongs uniquely to a process variant. In our approach, the same value for a given attribute will be dynamically computed and concatenated into the original dataset. The algorithms to model processes will then be based on this clustering action.

Definition 5. N-gram language models.

- *A language model is a statistical model that allows computing the probability of a sentence, or predict the next word in a sentence for a given language [31]. From a generative perspective, all sentences of a (natural) language can be described in terms of the product of a set of conditional probabilities [185]. Hence, the probability of a sentence* $\omega = (t_0,t_1,...t_n)$ *is given by :*

$$P(\omega) = P(t_0)P(t_1|t_0)P(t_2|t_0t_1)...P(t_n|t_0t_1...t_{n\text{-}1})$$

5.2.4 Related Work

5.2.4.1 Natural Language Models

Language Modeling. The use of natural language models was presented as an approach to recommend analogical libraries based on a knowledge base of analogical libraries mined from tags of millions of Stack Overflow questions [41]. This approach used a combination of a word embedding technique and domain-specific relational and categorical knowledge mined from Stack Overflow. Evidence showed that accurate recommendation of analogical libraries is not only possible but also a desirable solution.

In [185], a system was built to assists developers in Application Program Interface (API) usage with code completion recommendation, using a n-gram probabilistic language model, supported by API sentences extracted from source code corpora.

Topic Modeling. A survey on the use of topic models when mining software repositories is presented in [43]. The authors found that only a limited number of software engineering tasks were being targeted, and researchers use topic models as black boxes without fully evaluating their fundamental assumptions. Finally, they provide guidelines on how to apply topic models to specific software engineering tasks.

With the goal of predicting future developer behavior in the IDE and to make better recommendations to developers, [51] used topic models and specifically applied the Temporal Latent Dirichlet Allocation algorithm on two large interaction datasets for two different IDEs,

Microsoft Visual Studio and ABB Robot Studio. The authors concluded that the approach was promising for both predicting future IDE commands and producing empirically-interpretable observation.

An approach to detect duplicate bug reports, using information retrieval and topic modeling, namely LDA, was presented in [159]. The latter revealed an improvement of up to 20% in accuracy, when compared to other state-of-the-art approaches.

A study of software logging using topic models, with the aim of understanding the relationship between the topics of a code snippet and the likelihood of a code snippet being logged (i.e. to contain a logging statement) is described in [122]. The findings highlight the topics containing valuable information that can help to guide and driving developers' logging decisions. A similar approach is presented in [219], based on the structure and dynamics of knowledge network in domain-specific Q&A sites, particularly on Stack Overflow.

A large-scale study on security-related questions on Stack Overflow was presented in [218]. Two heuristics were used to extract the questions that are related to security from the dataset based on the posts' tags. Later, to cluster different security-related questions based on their texts, the authors used LDA tuned with a Genetic Algorithm (GA).

5.2.4.2 Mining Software Repositories

An application of mining three software repositories: team wiki (used during requirement engineering), version control system (development and maintenance), and issue tracking system (corrective and adaptive maintenance) in the context of an undergraduate Software Engineering course was presented in [142]. Visualizations and metrics provided insights into practices and procedures followed during various phases of a software development life-cycle, granting a multi-faceted view to the instructor and serving as a feedback tool on the development process and quality by students. Examples of insights produced by mining software repositories include understanding and assessing: (i) the degree of individual contributions in a team, (ii) the quality of commit messages, (iii) the intensity and consistency of commit activities, (iv) the trend and quality of the bug fixing process, (v) the component and developer entropy and, (vi) process compliance and verification. Experimentation revealed that not only product quality but also process quality varies significantly among student teams and mining process aspects can help the instructor in giving directed and specific feedback.

Mining Developers' Behavior. An investigation on how developers spend their time based on a fine-grained IDE interaction events dataset is presented in [139]. Its authors propose an inference model of development activities to precisely measure the time spent in editing, navigating and searching for artifacts, interacting with the user interface, and performing corollary activities, such as inspection and debugging.

In [184], the authors present an empirical study where app stores were mined to find out if developers update third-party libraries in mobile apps and also to identify update patterns. Evidence found unveiled that mobile developers rarely update their apps regarding used libraries and when they do, they mainly update Graphical User Interface (GUI)-related ones.

The measurement of developers' elapsed time in program comprehension activities beyond their IDE interactions is described in [216] in a field study with professionals. Findings showed

that, on average, developers spend 58% of their time on program comprehension activities, and they frequently use web browsers and document editors to perform program comprehension activities. Regarding the impact of programming languages, developers' experience, and project phase on the time spent on program comprehension, evidences shown that senior developers spend significantly fewer percentages of time on program comprehension than junior developers.

The assessment of development behaviors and testing practices in real-world projects is reported in [21]. The authors performed a study involving thousands of developers who were monitored closely on their development activities during the usage of four different IDEs. Results demonstrated that half of the developers' population does not test programs and they rarely run their tests in the IDE. Regarding the behaviors and beliefs towards TDD, findings show this activity as a non-frequent practice, and software developers only spend a quarter of their time engineering tests, whereas they think they test half of their time.

Mining End-Users' Behavior. Guidelines for the analysis of data collected during software in operation (i.e. when a software product is used by its end-users) are presented in [162]. The authors adopted techniques for extracting knowledge on software operation data, such as users' profiling, clickstream analysis, and classification analysis.

Wrap-up. The aforementioned approaches use a series of n-gram models, topic modeling, and process mining methods mainly to assist programmers in their most basic daily duties, and to discover how end-users operate software products.

Our work uses similar methods however, it focuses on finding developers' fingerprints with the aim of understanding and profiling programmers' behaviors. This approach may provide professors a way to assess students' performance within class tasks. Regarding software and project managers, at an enterprise level, they may use it to improve their task assignment strategies depending on project characteristics, devising adequate replacements in turnover situations, and balancing the constitution of software teams. As for process quality monitoring and enhancement, it can help in finding the good and bad processes followed by a development team or organization.

5.3 Study Setup

In a controlled experiment where the main objective is analysing programmers' behavior, there is an obvious main source of variability that should be blocked: the nature of the programming task itself. In other words, the optimal setting is to have several programmers[3] performing the same task. Other sources of variability are the programming language used, the IDE used, the working conditions and available schedule.

Being able "recruit"participants in industry for such an experiment, while blocking all the aforementioned factors is not feasible in a professional context. However, we were able to do that during an academic event dubbed Pythacon[4]. In this event, the same well-defined tasks on software development were performed individually by many participants. Pythacon's first

[3] as many as possible to achieve statistical significance

[4] A twisted contraction of Python + Hackaton : https://sites.google.com/iscte-iul.pt/pythacon

phase consisted of taking a Python in-class Massive Open Online Courses (MOOC) [75]. The second phase consisted in a programming contest with six problems with increasing difficulty. The Mooshak[5] automatic judge was used to assess participants' performance in their quest for producing solutions in Python for the aforementioned problems [117].

The subjects of this experiment were undergraduate students from three 1st cycle Bologna degrees[6] at Iscte, a public university in Lisbon, Portugal. LCD students did not attend the first phase because their syllabus already included two courses on Python. As such, they acted as the control group regarding the "treatment" of taking an in-class MOOC.

All subjects acted as Python developers while trying to build solutions to the proposed problems, upon the PyCharm IDE[7], in the same premises[8] and an equal sprint duration (4 hours). As such, the aforementioned confounding factors were blocked. We developed a PyCharm plugin that captures all relevant IDE events, such as, navigational, editing and debugging actions. Each Pythacon participant installed it in its IDE right after reading and signing an informed consent. When starting the IDE for the first time after plugin installation, they were requested to provide their student id number, that was added to the events log. As for the Mooshak automatic judge, that somehow mimics a continuous integration pipeline with an acceptance test battery, it has embedded login and logging mechanisms that allow identifying each participant and its events (problem submissions and corresponding outcomes).

5.3.1 Development Sessions Extraction and Storage

Interaction events collected with our PyCharm plugin, were stored in a `JSON` file on each subject's computer. A sample event instance is presented in Listing 5.1. The field tags are self-explanatory.

By the end of Pythacon's programming contest, all event files were uploaded to a central server. Data were then stored into a MySQL database table where the username and event timestamp were composed as an unique key for purging duplicated data. The BPMN model in Figure 5.2 presents the complete schema for the data collection workflow.

[5] Available from its home page at http://www.ncc.up.pt/mooshak

[6] LEI (Computer Engineering), ETI (Telecommunications and Computer Engineering), IGE (Computer Science and Business Management) and LCD (Data Science)

[7] https://www.jetbrains.com/pycharm/

[8] A large open-space where each participant had an individual table, a portable computer and good natural light

Listing 5.1: Sample PyCharm Event Instance

```
{
"session"  : "c51973e3-562a-4b65-b6df-49f4c37792e1",
"timestamp_begin" : "2020-09-18T09:00:06.054Z",
"username"  : "87788",
"graduation" : "IGE",
"projectname"  : "PythaconResolution",
"filename"  : "P4.py",
"extension"  : "py",
"categoryName": "NavBarToolbar",
"commandName": "Run",
"platform": "JetBrains s.r.o. / PyCharmCore",
"platform_branch": "PyCharm",
"platform_version": "2020.2.1",
"java": "11.0.8+10-b944.31",
"os": "Mac OS X 10.15.6",
"os_arch": "x86_64",
"country": "Portugal",
"city": "Lisbon",
....
"hash": "0000a3a2cf78485419f15d7913789b16" //To detect event tampering
}
```

5.3.2 Data Analysis

The complete workflow followed in data pre-processing, aggregation and analysis is presented in Figure 5.3.

5.3.2.1 N-gram language models evaluation

Documents containing natural language, software code or development sessions, are often repetitive and highly predictable. A good language model should capture the regularities in the corpus. If carefully produced from a representative corpus, it will predict, with high confidence, the contents of a new document drawn from the same population. In other words, the model will not find a new document particularly surprising. In NLP, this idea is captured by a measure called perplexity, or its log-transformed version, cross-entropy [90].

Given a document containing the textual representation of a development session within the IDE, $s = a_1, \ldots, a_n$, where terms represent development commands or activities, and a language model M, we assume that the probability of the document estimated by the model is $p_M(s)$. We can write down the cross-entropy measure as:

$$H_M(s) = -\frac{1}{n} \log p_M(a_1, \ldots, a_n) \tag{5.1}$$

and by the formulation presented in earlier:

$$H_M(s) = -\frac{1}{n} \sum_{1}^{n} \log p_M(a_i | a_1, \ldots, a_n) \tag{5.2}$$

This measures how "surprised" a model is by looking at an unseen document. A model with low entropy for target documents is expected to be a good model. Higher probabilities are given (closer to 1, and thus lower absolute log values) to more frequent words, and lower

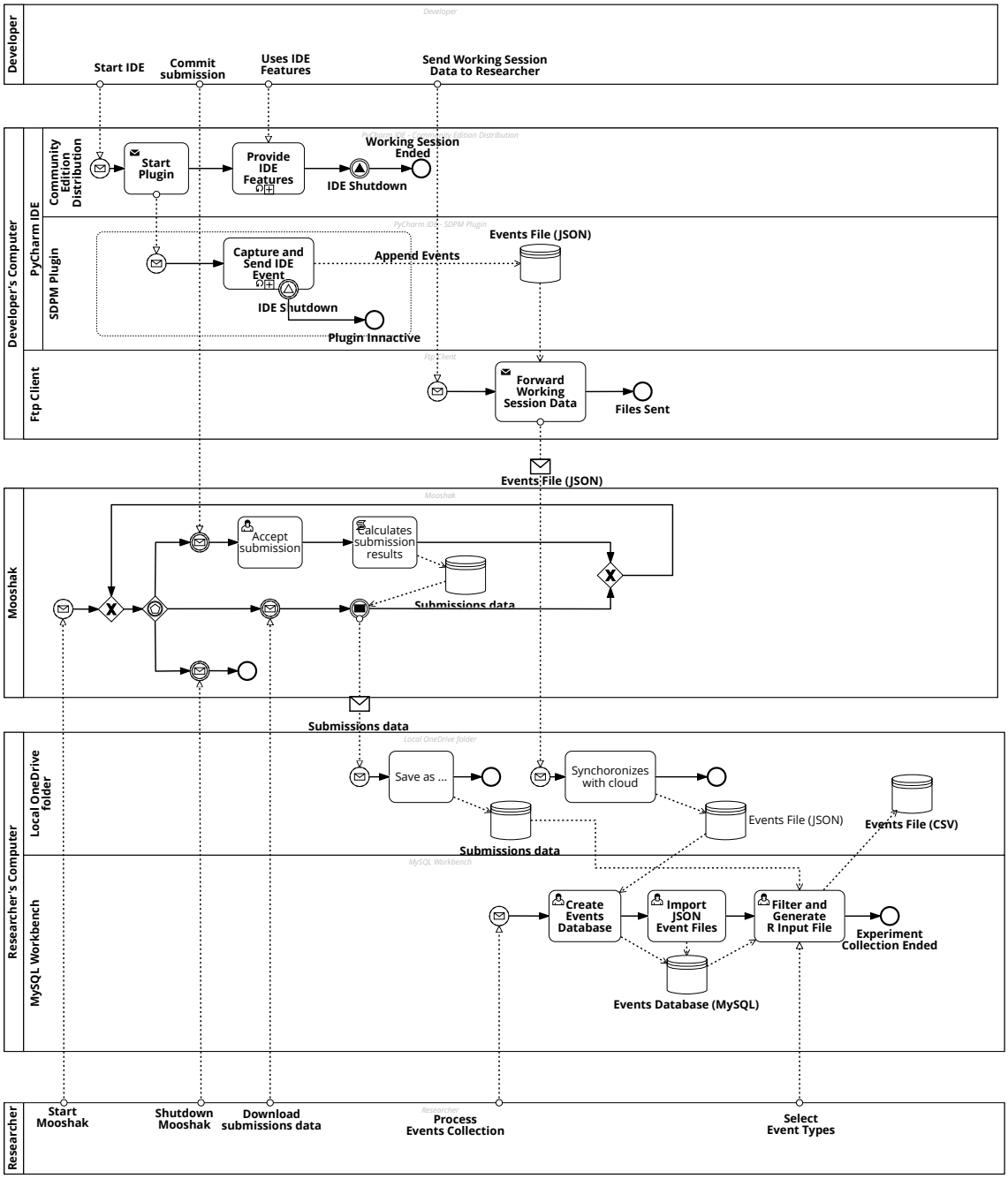

Figure 5.2: Data Collection Workflow

probabilities to rare ones. If a hypothetical optimal model is deployed to predict developers' actions, it is possible to guess what the next action would be and at the same time characterize developers' behaviors.

To shed light on how regular development sessions are, we performed a series of experiments with both natural language and development sessions corpora, first comparing the "naturalness" (using cross-entropy) of IDE actions on development sessions with English texts, and then comparing various session corpora to each other to further gain insight into the similarities and differences between sessions corpora.

Our natural language studies were based on a R package with widely used corpora from

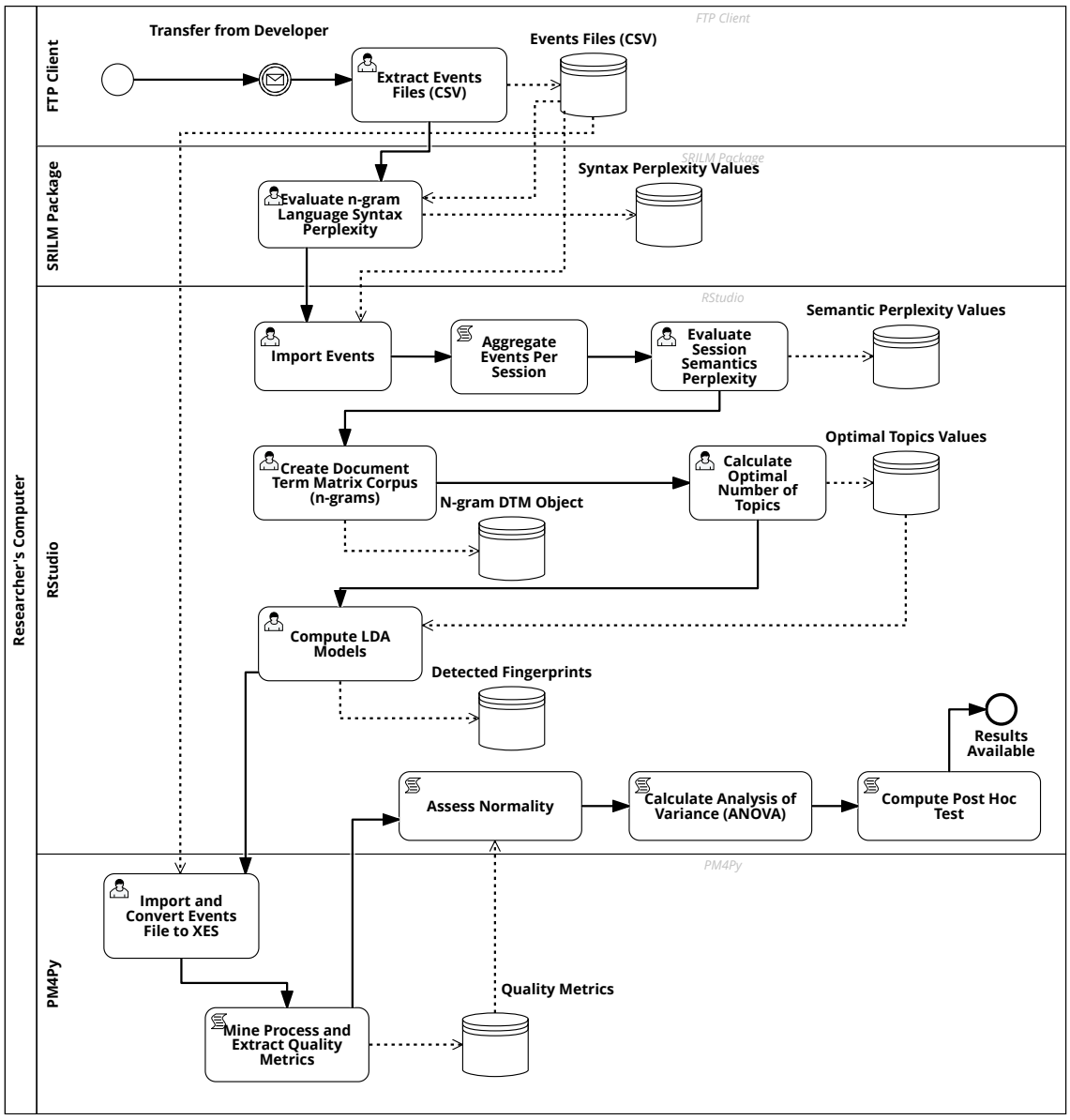

Figure 5.3: Study Computation and Analysis Process

Jane Austen's novels[9]. To compute the models perplexity and obtain the correspondent cross-entropy, we used the SRI Language Modeling (SRILM) package[10]. All the models were evaluated using a 5-fold cross validation strategy, meaning the corpus was randomly divided into 5 parts, where 80% was used as the training set and 20% as the test set, and this process was repeated 5 times.

5.3.2.2 Topic models evaluation

To determine the optimal number of topics to model developers' sessions, we used the R package *ldatuning*[11], that applies an empirical approach rather than intuition. Metrics such as

[9]https://cran.r-project.org/web/packages/janeaustenr/index.html
[10]SRILM Toolkit - http://www.speech.sri.com/projects/srilm
[11]https://cran.r-project.org/web/packages/ldatuning/ldatuning.pdf

CaoJuan2009 [38] and Arun2010 [13] are to be minimized (tend to 0), whilst metrics like Deveaud2014 [59] and Griffiths2004 [77] are expected to be maximized (tend to 1). The lower the distances to the objective values, the better the model and, consequently, the optimal number of topics are found in that particular point.

5.3.2.3 Process models evaluation

Process Mining is now a mature discipline with validated techniques producing accurate outcomes on several business domains [165]. Discovery is the ability to construct a process model, by capturing the behavior of a process based on an event log [206].

Following model discovery, conformance checking stands for the confrontation of a process model with the "reality" represented by the logged events during the actual execution of the corresponding deployed process. Conformance checking can be used to detect deviations from prescribed processes, determine differences and/or similarities between process variants or verify the accuracy of documented processes [206]. It can also be used to calculate the efficiency or to measure the quality of a process model. Quality is normally assessed considering four metrics:

- **Fitness.** Represents how much behavior in a log is correctly captured (or can be reproduced) by a discovered model [23].
- **Precision.** Refers to how much more behavior is captured in the model than what was observed in the log. It deals with avoiding overly under fitted models [148].
- **Generalization.** Focuses on avoiding overly precise models based on the assumption that logs are by their nature incomplete, meaning that, to a certain extent, a model should be able to reproduce not yet seen behaviour in the log [33].
- **Simplicity.** Alludes to the rule that the simplest model that can describe the behavior found in a log is indeed the best model. Model complexity, the opposite of simplicity, is dependent on the number of nodes and arcs in the underlying graph [177].

To calculate the previous metrics we used the Process Mining library for Python (PM4Py)[12].

5.3.3 Research Questions

The research questions for this work are the following:

- **RQ8:** Do n-gram language models capture local regularities in software development sessions?
  Methods used. Computation of n-gram language models perplexity/cross-entropy using SRILM and LDA with n-gram windows.
- **RQ9:** Can we coherently characterize development sessions in terms of fingerprints?
  Methods used. Topic Modeling using the LDA algorithm with n-gram window tuning.

[12] https://pm4py.fit.fraunhofer.de/documentation#discovery

- **RQ10:** Are there any significant variation in sessions simplicity and interactions magnitude between distinct participants?
  Methods used. Process models discovery using the Directly Follows Graph (DFG) mining algorithm and hypothesis testing.

5.4 Study Results

In this section, we present the results of our experiment, regarding its research questions.

Table 5.2: Participants Statistics

	LEI	ETI	IGE	LCD	Total
Participants	12	9	7	9	37
Attended MOOC	Yes	Yes	Yes	No	28

RQ8. Do n-gram language models capture local regularities in software development sessions ?

Local syntactic structure. To answer this question we estimated n-gram models of plain English corpus and the development session IDE commands and their categories.

From Figure 5.4, we observe that, although English has a higher level of cross-entropy across all n-gram models, it declines rapidly, saturating around tri- or four-grams. The same happens with our development sessions models, which have generally lower cross-complexity for unigram models, and also saturate around tri-grams models. This indicates, as expected, that development sessions repetitive context can also be captured by language models.

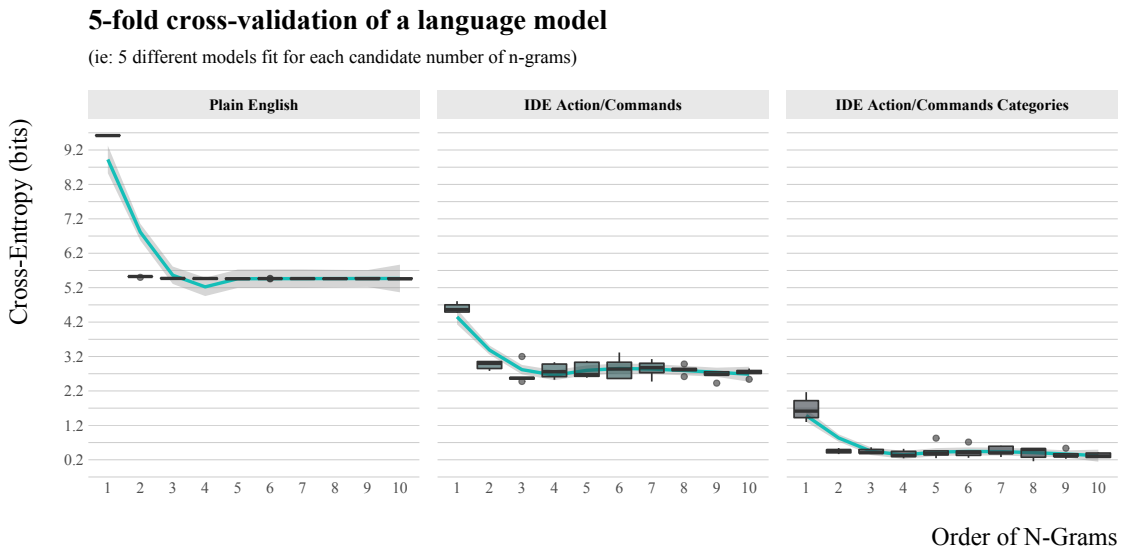

Figure 5.4: Plain English vs. Python Development Sessions Cross-Entropies using n-gram models

We find that a typical development session is far more regular than English, with entropies starting from 4.2 bits and declining to 2.7 bits by IDE command and starting at 2.2 bits and saturating around 0.7 bits for command categories.

Our findings may have implications in the way we manage developers' activities. They provide more and detailed evidence to confirm what was already mentioned by [51], the possibility to design and build even more optimized recommendation systems to help and guide developers on the activities they are executing or should be doing next. Moreover, they shed light on the optimal number of n-grams to use, thus avoiding the waste of computing resources and at the same time provide further evidence for the usefulness of using text mining techniques to detect and monitor developers' behaviors.

Semantics. In the context of IDE usage, each development session may have its own semantics. Whilst to capture the local syntactic structure of a language we used n-gram language models, to assess the semantics of the development sessions we used LDA. Figure 5.5 shows the cross-entropy regarding the semantics analysis for the development sessions. It consists in finding the entropy for n-gram models, each having k topics, where k varies from 2 to 10, and where each combination of n-grams and k topics was calculated with a 5-fold cross-validation strategy.

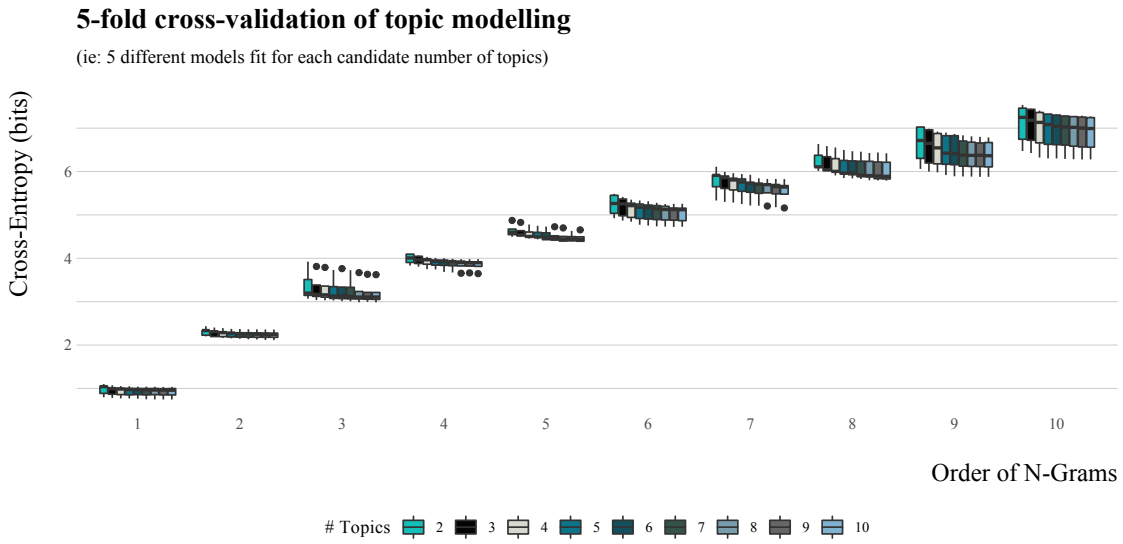

Figure 5.5: Development sessions modeling using LDA with k topics and n-grams

As one can easily observe, when using LDA to assess the number of topics, entropy grows with the order of the n-grams. The higher the n-gram window the less the model is able to predict future cases because the perplexity is higher. Regarding the number of topics on each n-gram model, we can confirm the expected behavior, when the number of topics increase, independently of the n-gram model, the entropy tends to decline.

Concerning the interpretation of the n-gram results perspective, they show that, given the randomness of IDE actions performed by developers, to increase the n-gram value in characterizing a session, decreases the ability for LDA to find similar ones. As for the number of topics, the bigger the number of topics the better the model can detect similar sessions.

Based on Figure 5.5, we argue that, when using LDA to detect similarities within development sessions, we should evaluate carefully the use of more than tri- or four-gram models. In one hand we know that the higher the n-gram model, the higher the computational resources needed. On the other hand we have evidences that five-gram models have an entropy of around 5 bits, which is by itself a high value.

RQ9. Can we coherently characterize development sessions in terms of fingerprints ?

Figure 5.1 in section 5.2.1 provided a small sample of the activities aggregating the commands issued by developers. Those were defined according to the method used by [139], and by adding extra activities reflecting the results of the submission events. Regarding IDE interactions, the commands were recoded into activities like: *Editing*, *Navigating*, *Debugging*, *Refactoring*, *Executing and Spurious*. As for the submission actions, we used their native identifiers : *Accepted_Answer*, *Wrong_Answer*, *Compile_Time_Error*, *Invalid_Submission*, *Runtime_Error and Time_Limit_Exceeded*. From these, we computed the optimal number of patterns(topics) by assessing the probabilistic coherence of multiple topics using the metrics described in 5.3.2.2 and uni-gram, bi-grams, tri-grams and four-grams models only.

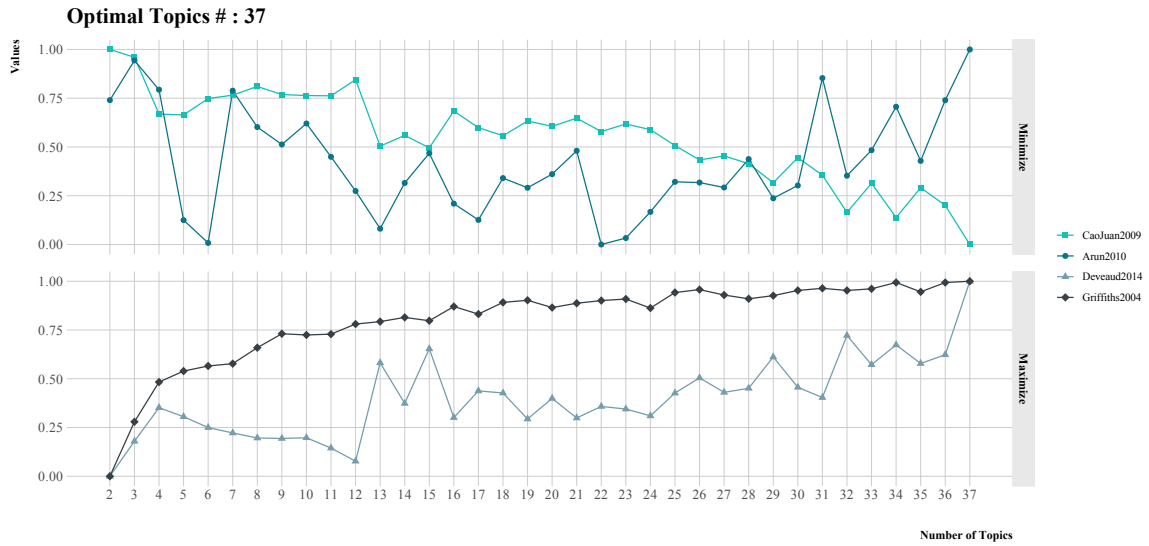

Figure 5.6: Assessing the optimal number of distinct patterns to search

Optimal number of patterns. To decide the optimal number of patterns, we took into account the highest value for the number of topics where any of the metric is close to the objective value, either when minimizing or maximizing its value and therefore, we picked 37. This number represents the size of the population, which in reality confirms that there is a great deal of variance between sessions. When applying the LDA algorithm with k=37, due to the average of the probabilities of an activity to belong to a session and the average of probabilities of a participant to belong to a specific session pattern, LDA has placed the participants in only 19 different patterns. Figure 5.6 shows the optimal number of topics evaluation.

High performers. Figure 5.7 shows the topics, or in our case, the referred fingerprints, identified to characterize what we call the high performers[13], the same is to say, the ones with good process smells. In this group we have those who ranked above quartile 3, meaning that they have answered correctly to at least four exercises. From the six fingerprints found for those participants, we can observe the following:

- **Fingerprint 19. Cautious coder. Aggressive executor.** Contains a profile of development centered in frequent editing actions, mixed with permanent execution of the code to validate the result before submitting to Mooshak. This pattern was the fingerprint of participant D.
- **Fingerprint 20. Cautious coder.** Reveals a similar pattern, however, with less prevalence of program execution and therefore less testing actions. Exercise submission actions are very rare. With this fingerprint we find 2 participants, G and H.
- **Fingerprint 24. Cautious coder. Test skipper.** With no surprise, in this fingerprint, we find editing also as the most frequent action. However, the next common action is not program execution, but submissions for validation. This pattern was the fingerprint of participant C.
- **Fingerprint 26. Insecure. Testers.** Participants characterized in this group followed explicitly a permanent program execution, followed by editing activities. They have submitted their answers infrequently, meaning that they have probably tested well their work before any submission. This pattern was the fingerprint of participant F.
- **Fingerprint 30. Insecure. Debuggers.** This pattern reveals participants more focused on debugging activities, followed by a mix of editing and navigational actions. In a certain sense, it looks as if they have replaced their program execution tests with fine grain debugging practices. This pattern was the fingerprint of participant E.
- **Fingerprint 35. Balanced coders. Confident.** Provides evidence for a pattern of high frequency in editing, followed by a balanced persistence of program execution practices and navigational activities. There were however no frequent activities related with the submission of code to answer the exercises. It suggests these participants only submitted their answers after careful review of their code and without the need for deeper debugging tasks. With this fingerprint we find the top 2 participants, A and B.

Low performers. Figure 5.8 represents the characteristics of those who had more difficulties in executing the tasks and which we may consider as having bad process smells. It plots the unique fingerprints of the last eight participants, the ones with zero or just one correct answer.

- **Fingerprint 7. General coding limitations.** Reveals a practice focused almost exclusively on editing actions, and a very small prevalence (near zero) of navigational, program executions or exercise submission activities. This pattern was the fingerprint of participant V.

[13] Eight participants were in this condition.

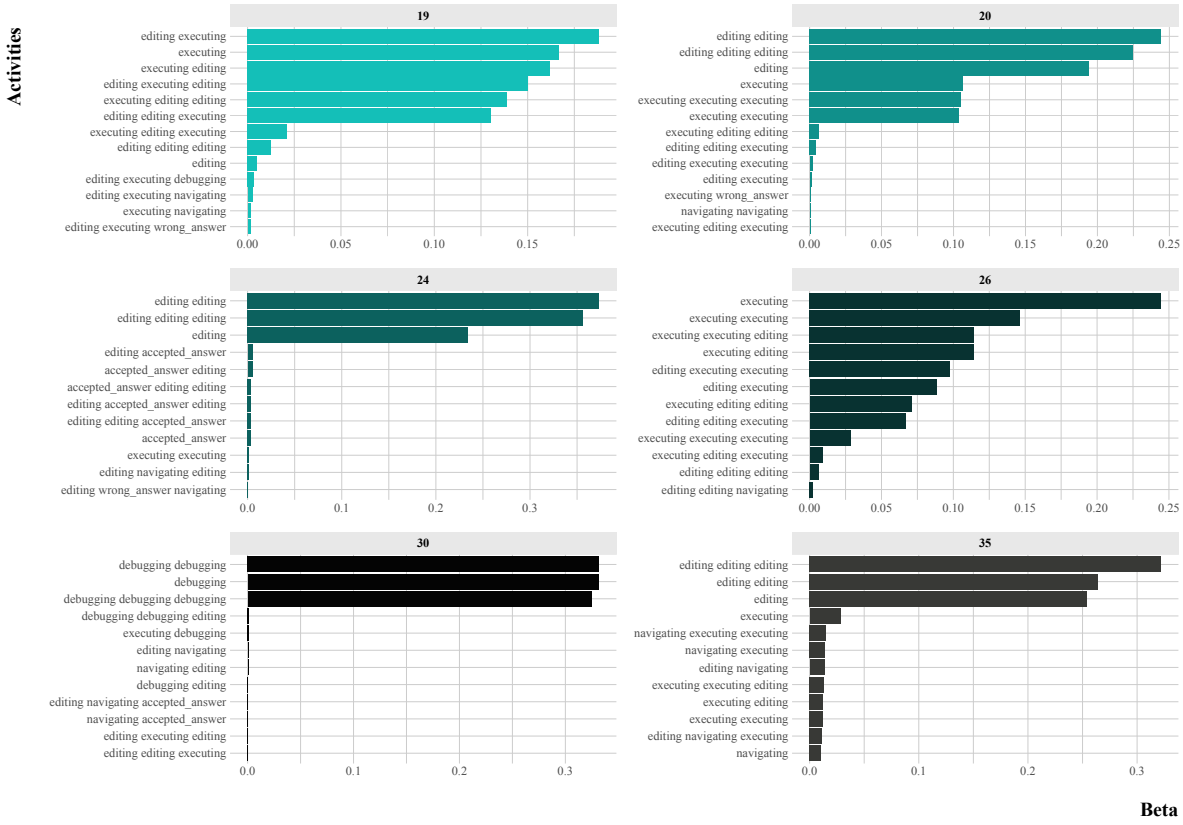

Figure 5.7: Development fingerprints for the top(8) performers

- **Fingerprint 18. General coding limitations.** Shows a similar profile as fingerprint 7 regarding editing practices. However, program executions and answer submissions appear more often in the complete work sessions, yet with a low frequency (near zero) when compared with editing. This pattern was the fingerprint of participant U.

- **Fingerprint 25. Limited python/algorithmic skills.** It characterizes a practice where editing is also the prevalent action and combines this frequently with debugging and program execution activities. Answer submission is however infrequent, as none shows in the most common actions. This pattern was the fingerprint of 5 participants, S, T, W, X and Y. None of them has attended the MOOC training sessions.

- **Fingerprint 33. Limited python/algorithmic skills.** This practice is characterized as usual by frequent editing actions, and then followed by a decreasing and balanced editing, navigational and refactoring decisions. However, submission actions are absent. This pattern was the fingerprint of participant Z.

It is striking to observe that the fingerprints characterizing the high performers are significantly different, either in the top activities and also in probabilities, from the ones describing the low performers. Figure 5.9 presents the distinct fingerprints detected to characterize all participants. Based on these findings, one may argue that the variation in the participants scores was only due to the quality of the code they have produced, moreover, that the variation in the fingerprints was due to their own programming skills. Additionally one may suggest as

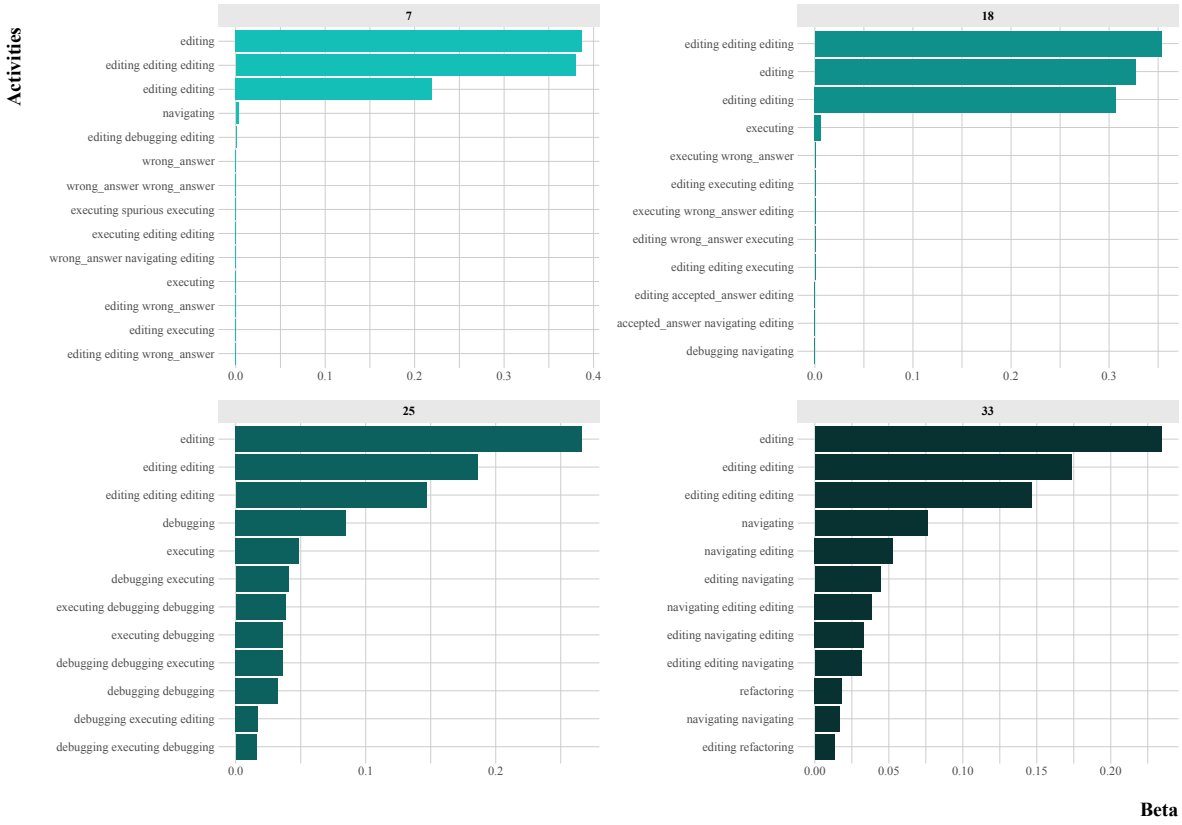

Figure 5.8: Development fingerprints for the bottom(8) performers

an explanation for the top performers, the knowledge acquired in the course they belong to, due to the MOOC training attendance or the dimension of their coding interactions during the contest and not based on their coding behaviors. That is a possibility we cannot reject immediately. However, we may assess this hypothesis if we mine, from a different perspective, the overall process for each participant, course as a group of participants with same backgrounds, according to their MOOC training participation and finally according to their performance. If the reason for the higher performance is related with the magnitude of their interactions or the quality of their code, we expect to see no significant variance in the process simplicity amongst different participants. On the contrary, if variation in the process exists between groups, that may be an indication that the quality of the outcome is indeed related to the development workflow. Following the above rationale, we mined the correspondent processes using a mining algorithm appropriate for processes with thousands of events and where a fuzzy or spaghetti-like process behavior is expected to exist. We used a DFG algorithm from the Process Mining library for Python mentioned earlier, and assessed the models produced using the quality metrics described in section 5.3.2.3. Table 5.3 summarizes the fingerprint results for the referred participants, along with the metrics used to evaluate the quality for the process models discovered for each of them. The hypothesis we later tested was as follows: Are there significant differences in the processes complexity or development interactions between the different graduation courses or between the top, bottom and the rest of the participants ?.

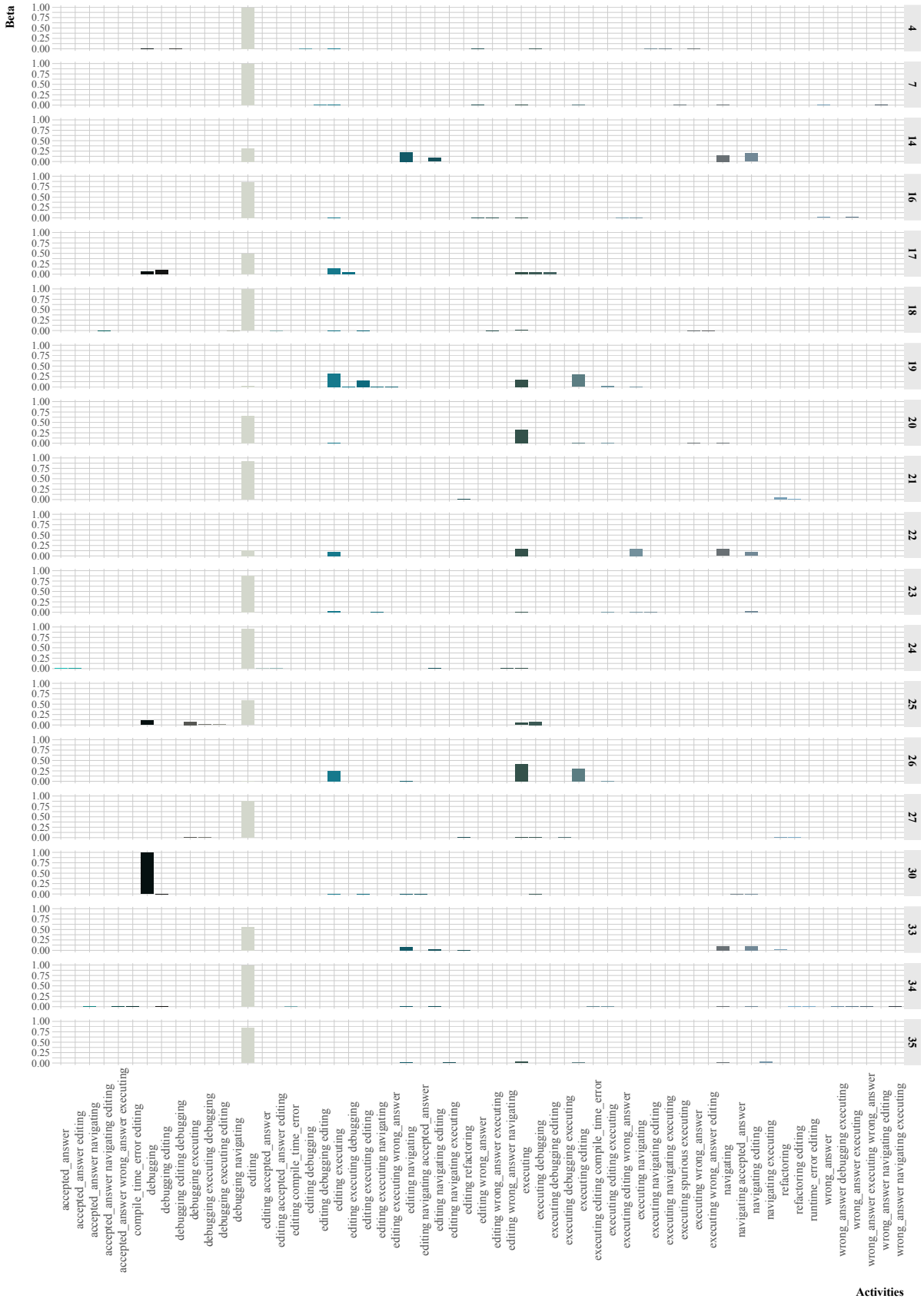

Figure 5.9: Development fingerprints characterizing all participants

Table 5.3: Process Models Evaluation

	Course	Fingerprint	Interactions	Fitness	Precision	Generalization	Simplicity	Average	Duration
By Course									
LEI	–	–	33011	0.017	1	0.142	0.432	0.398	00:07:09
ETI	–	–	26557	0.818	1	0.153	0.439	0.602	00:05:01
IGE	–	–	16635	0.199	1	0.143	0.438	0.445	00:02:19
LCD	–	–	30057	0.198	1	0.126	0.426	0.438	00:06:47
MOOC Training									
MOOC	–	–	76203	0.039	1	0.124	0.411	0.394	00:27:27
NO_MOOC	–	–	30057	0.198	1	0.126	0.426	0.438	00:06:18
Performance Type									
High (Top 8)	–	–	23351	0.009	1	0.140	0.437	0.396	00:04:07
Low (Bottom 8)	–	–	25869	0.957	1	0.153	0.447	0.639	00:04:10
Middle	–	–	57040	0.037	1	0.121	0.410	0.392	00:19:58
High Performers									
A	ETI	35	4447	0.089	1	0.158	0.489	0.434	00:00:32
B	IGE	35	3554	0.123	1	0.181	0.538	0.460	00:00:21
C	LEI	24	1515	0.908	1	0.192	0.507	0.652	00:00:05
D	LEI	19	2194	0.118	1	0.202	0.524	0.461	00:00:09
E	IGE	30	4809	0.353	1	0.162	0.472	0.497	00:00:35
F	IGE	26	2495	0.228	1	0.173	0.543	0.486	00:00:12
G	LEI	20	2481	0.978	1	0.181	0.529	0.672	00:00:10
H	LEI	20	1856	0.170	1	0.174	0.506	0.462	00:00:07
Low Performers									
S*	LCD	25	3520	0.042	1	0.179	0.514	0.434	00:00:22
T*	LCD	25	4615	0.204	1	0.186	0.524	0.478	00:00:35
U	LEI	18	2200	0.021	1	0.173	0.521	0.429	00:00:10
V	IGE	7	1064	0.987	1	0.189	0.617	0.698	00:00:03
W*	LCD	25	2599	0.128	1	0.188	0.544	0.465	00:00:12
X*	LCD	25	4393	0.336	1	0.239	0.527	0.526	00:00:32
Y*	LCD	25	1964	0.145	1	0.203	0.585	0.483	00:00:07
Z	ETI	33	2781	0.831	1	0.251	0.565	0.662	00:00:13

Interactions - Represent actions within the IDE and Mooshak, **Duration** - Means the time to build/compute the process model
* - Participant did not attend the MOOC

RQ10. Are there any significant variation in sessions simplicity and interactions magnitude between distinct participants ?

Simplicity is one of the dimensions to analyze a process model, and to calculate it, PM4Py takes into account only a Petri net model. The criteria adopted for calculating simplicity is the inverse arc degree as described in [148]. Since we mined individual processes, they would represent the behavior simplicity of each participant in the programming exercise.

Interactions magnitude refers to the sum of the number of command actions executed in the IDE plus the submission of answers in the Mooshak platform. In other words, interactions are represented by the events generated during the programming exercise by each individual.

The objective of this test is to assess if there is a relation between the performance along with the sessions simplicity or magnitude of interactions on different sets of participants. For this purpose we tried the Analysis of Variance (ANOVA).

ANOVA Test. Tests if there are significant different statistics between groups of participants, or the same is to say, helps to figure out if one needs to accept or reject the null hypothesis. A one way ANOVA is used to compare two means from two independent (unrelated) groups using the F-distribution. The null hypothesis for the test is that the two means are equal. Therefore, a significant result means that the two means are unequal. It has the ability to tell if at least two groups were different from each other, however, it won't tell which groups were different and by which magnitude. If a test returns a significant f-statistic, then one may need to run an ad hoc test (eg: Tukey HSD) to learn exactly which groups had a difference in means.

Tukey HSD ("honestly significant difference"or "honest significant difference"). Is a statistical tool used to determine if the relationship between two sets of data is statistically significant – that is, whether there's a strong chance that an observed numerical change in one value is causally related to an observed change in another value. In other words, the Tukey test is a way to test an experimental hypothesis.

Variables Assumptions. The use of ANOVA has several assumptions, such as: i) the dependent variable should be measured at the continuous level or absolute scale; ii) the independent variables should define at least two categorical treatments, that corresponds to the groups to which the participants belong; iii) there should be no significant outliers in the groups since they can have a negative effect on ANOVA; iv) the distribution of the dependent variables should be as normally distributed as possible. Having all other conditions satisfied, we assessed the normality.

Normality Tests. To test normality, we may use two well-known tests of normality, namely the Kolmogorov-Smirnov and the Shapiro-Wilk tests. We only considered the Shapiro-Wilk test to assess normality since the latter is more appropriate for small sample sizes (< 50 samples). The results are presented in Table 5.4, and from them, we cannot reject the null hypothesis, therefore, we accept that both Simplicity and Interactions are normally distributed justifying the use of ANOVA.

Table 5.4: Normality Tests

	Shapiro-Wilk	
Factor	***Statistics(W)***	***Sig.****
Symplicity	0.94802	0.08334
Interactions	0.95760	0.16940

**Statistically significant if Sig. < 0.05*

Findings. From Table 5.5, we can confirm the significant variance between the ones with less quality in their code(Bottom5) and the rest of the participants (Top5 and Others), and this difference is larger between the top five and bottom five performers. Figures 5.10 and 5.11 show the process models discovered for both, respectively. These results provide evidences to state that, the differences in the proficiency between certain participants are not only related with the coding skills each of them may have.

The process complexity followed by each developer may also influence the outcome, or at least, may be used as a valid indicator for assessing quality between developers. However, when these groups are configured to contain the top and bottom eight participants in terms of score, that significance is no longer visible within the same levels of confidence ($\alpha < 0.05$). This reinforces the fact that if significant differences exist, they are most likely and only in the individual behaviors. As for analyzing the variance between different courses, we found no significant differences, either for the development sessions simplicity as well as for the magnitude of interactions.

As mentioned earlier, we have assembled a method to capture local regularities and overall structure of development processes, the so called fingerprints. Based on the data we obtained, namely the probabilities of activities in the development sessions, and the metrics from the discovered processes, we may classify those fingerprints as good or bad process smells and start to create a catalog of software development process smells. Later on, models can be built to evaluate automatically if a coding session is following a good or a bad practice and suggest guidance actions to developers.

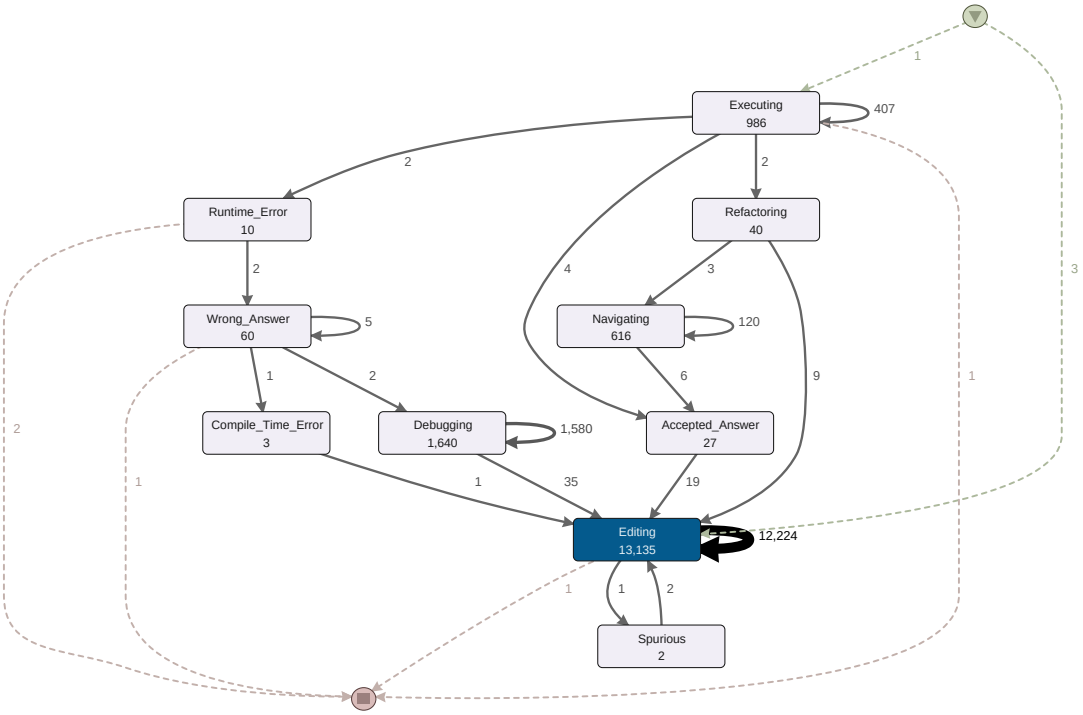

Figure 5.10: Process Model characterizing Top5 Participants

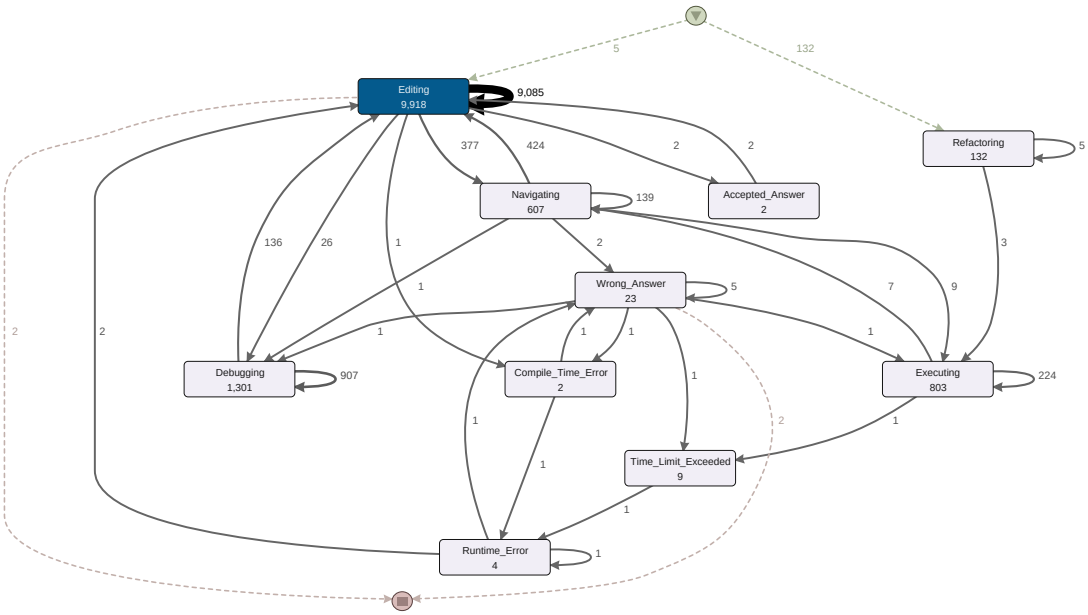

Figure 5.11: Process Model characterizing Bottom5 Participants

Table 5.5: One-Way ANOVA Results

	Factor	Df.	Sum Sq.	Mean Sq.	F-value	*p-value*
Analysis of Variance Test - Top5/Bottom 5/Others						
Simplicity	**Performance**	2	0.01150	0.005750	5.984	**0.00594***
	Residuals	34	0.03267	0.000961		
Post Hoc Test						
	Treatments	**Diff**	**Lower**	**Upper**	***p-adj***	
	Others-Top5	0.01418519	-0.02279834	0.05116871	0.6192293	
	Bottom5-Top5	0.06160000	0.01355700	0.10964300	**0.0094695***	
	Bottom5-Others	0.04741481	0.01043129	0.08439834	**0.0094774***	
Interactions	**Performance**	2	1431864	715932	0.555	0.579
	Residuals	34	43821404	1288865		
Analysis of Variance Test - > Q3(Top 8)/Q1(Bottom 15)/Others						
Simplicity	**Performance**	2	0.00335	0.001676	1.396	0.262
	Residuals	34	0.04082	0.001201		
Interactions	**Course**	2	165352	82676	0.062	0.94
	Residuals	34	45087916	1326115		
Analysis of Variance Test - Top8/Bottom 8/Others						
Simplicity	**Performance**	2	0.00656	0.003279	2.963	0.0651
	Residuals	34	0.03762	0.001106		
Interactions	**Performance**	2	34612	17306	0.013	0.987
	Residuals	34	45218656	1329960		
Analysis of Variance Test - LEI/ETI/LCD/IGE						
Simplicity	**Course**	3	0.00371	0.001237	1.009	0.401
	Residuals	33	0.04046	0.001226		
Interactions	**Course**	3	3919333	1306444	1.043	0.386
	Residuals	33	41333934	1252543		

**Statistically significant if p-value < 0.05*

Df. - Degrees of freedom, **Sum Sq.** - Sum of Square, **Mean Sq.** - Mean of Square

5.5 Threats to Validity

The following types of validity issues were considered when interpreting the results of this article.

Construct Validity. Construct validity refers to the degree to which inferences can legitimately be made from the operationalizations in a study to the theoretical constructs on which those operationalizations were based.

For operationalizing language models assessment, we used metrics such as perplexity and cross-entropy, and CaoJuan2009, Arun2010, Deveaud2014, and Griffiths2004 to evaluate, from an empirical perspective, the optimal number of topics, and validated their values from multiple perspectives. Other metrics could have been used for the same purpose, such as topic coherence, which may lead to recommending a different optimal number of topics.

Since our sample was not very large, we had to use it for training and test purposes. To strengthen significance, models were trained using 5-fold cross-validation. We are aware that process model metrics such as Precision and Generalization are far from being usable in a more generic process mining context. However, in this study, we were only focused on process simplicity, and regarding that purpose, the mining and tests are valid since we used the same algorithm for all participants.

Each participant used their student number to activate the PyCharm events collector plugin. This approach served as an identification method. Events collected and stored in JSON files on developers' devices could have been manually changed. We tried to mitigate this threat of having data tampering by using a hash function on each event at the moment of its creation. As such, each event contains not only information about the IDE activities, but also a hash code introduced as a new property in the event for later comparison with original event data.

Internal Validity. Internal validity refers to the degree of confidence that the causal relationship being tested is trustworthy and not influenced by other factors or variables.

One typical threat to internal validity related to how subjects are selected. In our case, the population from where our sample was taken, corresponds to all undergraduate students in computer science areas in our university that had attended at least two programming courses. That population was invited to participate by email. The sampling process was the result of a random process of free will where those students that spontaneously decided to participate performed their inscription online. As such, we do not consider this to be a significant validity threat.

Another recurrent internal validity threat is the existence of spurious factors affecting the outcome of the experiment. In mitigation, the programming contest in our study allowed us to block possible confounding factors since they were constant for all subjects: the programming language (Python), IDE (PyCharm), problem complexity (same requirements spec), sprint schedule (4 hours), environment conditions (large shared open space with private tables), and external interference (no contacts were allowed). Once again, we believe that this threat is also not significant.

External Validity. External validity refers to the extent to which results from a study can be applied (generalized) to other situations, groups or events.

To fully claim that undergraduate students are surrogates of professional programmers, a

representative sample of both groups should be assigned the same requirements specification for a Python program, to measure the difference on their outcome. We are not aware of such a study having been published. Nevertheless, there is a likelihood that our students are at least good surrogates for novice professional software developers in Python, because:

(i) Python has a low learning curve, based on our experience, corroborated by [152], so that the level of proficiency of a professional Python programmer seems to be achievable quickly;

(ii) our students had attended successfully, on average, two Python courses;

(iii) a questionnaire filled during inscription showed that participating students, albeit having gone through similar academic paths, had different maturity and skills, as we would expect in professional programmers; that difference most probably will not fade out within the one or two years that will take for the vast majority of these students to become professionals.

Conclusion Validity. The conclusion validity describes our ability to draw statistically correct conclusions based on the measurements. A common threat here is the sample size, but in our case we were able to get a sufficiently large number of subjects to grant statistical significance.

We carefully evaluated the models perplexity computed to answer RQ9, and the assumption tests to justify the applications of the statistical tests in answering RQ10, however, we have also to accept that our sample is not of large proportions. We performed an experiment using data from 37 software developers executing well defined and identical programming tasks. Since this is a moderate population size for this type of analysis, we agree this may be a threat to generalize conclusions or make bold assertions. Nevertheless, to our best knowledge, this is the first study involving development sessions and the usage of language models, text and process mining to detect developer's fingerprints during programming tasks. As such, researchers can start from our initial findings and try to falsify our current results and correspondent conclusions.

5.6 Summary

Main conclusions. Profiling developers is challenging since many factors, such as their skills, experience, development environment and behaviors, may influence a detailed analysis and the delivery of coherent interpretations.

We mined the PyCharm and Mooshak events from a group of developers during a Python programming contest aiming to solve six different exercises. We used n-gram language models and text mining to characterize developers' profiles, and process mining algorithms to discover their overall workflows and extract the correspondent metrics for further evaluation.

Our research regarding development interactions shows that they can be mined as a natural language and using text mining methods with tri- or four-grams being the optimal value for such task. Findings show that we can clearly characterize with a coherent rationale most developers, and distinguish the top performers from the ones with more challenging behaviors.

We also confirm a significant difference in the process simplicity between the top performers and the ones with unsatisfactory outcomes on the programming exercises.

In summary, results provide evidences to sustain that, to achieve software with good quality, it is not only needed to have developers with the right skills and consistent knowledge about the languages and tools used on their daily tasks. It is also desirable to have developers to follow consistent practices during the development sessions, otherwise, their behaviors may impact the final outcome, thus, properly profiling developers, provides a software project manager a clue for the selection of appropriate tasks he/she/they should be assigned. Moreover, this approach may lead ultimately to the creation of a catalog of software development process smells.

Our approach can be particularly relevant in cases where educators want to assess development profiles within a group of students, before and after classes are given. It can be also a valid approach to measure and monitor productivity within and between software teams. As we showed, by analyzing the development sessions fingerprints and complexity, non efficient developers can easily be detected.

Last generation IDEs provide a plethora of functionalities, such as code completion, automated packaging and optimized continuous integration features to assist programmers on their daily activities. However, these IDEs do not guide developers on their coding practices. The fingerprints we detected, either classified as good or bad practices[14], may be used as a trigger for IDE vendors to evaluate the possibility to include additional intelligence in their tools, such as task/workflow monitoring and suggesting program runs, identify testing slots and appropriately recommend debugging and refactoring actions along a development session.

Future work. We are still scratching the surface in mining developers' activities. Existing mining tools are not ready yet to automate the complete flow of: collect and pre-process data, discover processes behaviors, compute metrics, and export results. Further work is required to set up a pipeline capable of providing just-in-time feedback, both to software developers, to provide self-awareness on performance/behavior, as to software project managers, since the profile of team members allows a more informed resource allocation.

Novel software development paradigms, such as low and no code, shift the focus from the textual programming and put it into the visual artifacts and components from which modern applications are built upon. Our work fits well in cases where textual programming is banned, giving rise to the so-called *citizen developers*, and therefore, most likely to distinct development processes and coding behaviors when compared with conventional programming practices. We plan to perform additional experiments using low or no code platforms and assess developers' process fingerprints and overall behaviors.

[14]We already called them - software development process smells

[This page has been intentionally left blank]

Part IV.

Conclusion

This part concludes this dissertation.

Chapter 6

Conclusions and Future Work

Contents

This chapter summarizes the main contributions of this work, draws opportunities for further research and concludes this dissertation.

Companion Soundtrack: God Moving Over The Face of the Waters - Moby (Heat Soundtrack)

"If two people agree on everything, one of them is unnecessary."
—Winston Churchill(1874-1965)[1]

6.1 Introduction

Modern software projects require the proper allocation of human, technical and financial resources. Very often, software project managers make decisions supported only by their personal experience, intuition, or simply by mirroring activities performed by others in similar contexts. To mitigate the risks associated with such practices, software repositories are used as data sources for many different forms of analytics related to the software development processes. However, in such a context, these repositories have many flaws, thus impacting the quality of the data used and the interpretation of findings in a broad set of analyses. As a consequence, wrong conclusions and decisions may emerge frequently causing financial losses and other undesired outcomes, such as low quality or overdue deliveries.

Mining software development sessions, process fragments, or complete end-to-end processes may contribute to reducing the uncertainty related to the real duration of development tasks and the workflow adopted by developers. We mined the use of the IDE by software developers in three different scenarios by collecting events during their development sessions. Furthermore, we used process mining to discover their real workflows, unveil insights from their coding practices in multiple refactoring tasks and characterize their behaviors in a programming contest.

We expect that our research may open new research threads and on deriving mitigation solutions to problems faced by both developers and their organizations regarding the software development process. In this chapter we provide a synthesis on how far we have progressed in that direction, recapping both the contributions and their limitations and provide a roadmap for future work along several lines.

6.2 Synthesis

In chapter 1, we presented the scope, fundamental topics, the research problems, and the correspondent questions to sustain the purpose of this dissertation: i) to address the incompleteness of the repositories used to assess software processes, ii) the inability to accurately express developers' workflows and their impact on software products and iii) the challenges in profiling developers' due to the great variability of their behaviors during development sessions.

In chapter 2, a systematic literature review characterizes the current state of the art concerned with software development analytics and its application in the field paving the way to include our proposed methods within the overall software development domain. We concluded that few works were performed in mining software development processes beyond what traditional software repositories allow. Moreover, we found that process mining methods were rarely used, and when they were, it was to mine code executions or end-users behaviors.

[1] Politician, Writer and Nobel Prize in Literature(1953)

To assess the possibility of using process mining algorithms in mining development practices, in chapter 3, we evaluated the efficiency of development teams and confirmed that a great deal of variability exists among individuals and among teams.

In chapter 4, we mined during distinct refactoring activities, performed by more than one hundred developers, both product and process metrics, aiming to evaluate the existence of relationships between them. We found non-neglectable correlations between product and process complexities and several other process-driven metrics. Based only on these metrics, we built prediction models to detect the type of refactoring practices applied by those developers. Additionally, we were able to produce an accurate prediction of the potential reduction in software complexity, achievable by a refactoring task.

Chapter 5 presents a study where we test some of our assumptions and validate our approach and proposals in mining software development sessions. We were able to detect, not only coherent fingerprints left by developers, but also confirm that differences in proficiency between high and low performers were, at least partially, related to the followed process (i.e. their coding practices).

In chapter 6, we summarize the main contributions and identify potential use cases for our proposed methods, targeting both industry and academia. Additionally, we describe a few issues still present when applying process mining methods and tools in software development analytics. Finally, to conclude this dissertation, we detail some of the main opportunities in research related to software development process mining and where it can fit in contemporary software development projects.

For a quick reference, we provide the research problems and main findings in Table 6.1.

6.2.1 Limitations

We are just scratching the surface in mining developers' activities using process mining tools. Although our method collects events during real development sessions, we cannot underestimate the fact that we used only IDE events, and in one case, check-in events in the context of a programming contest. In a more complete scenario, events from other tools should also be considered, such as those from documentation, website browsing, bug tracking tools, and project management platforms. In other words, adding events from tools containing information about documentation, project management decisions, communication between developers and managers, Q&A services, test suites, and bug tracking systems, would allow building more robust models to comprehend development practices and the relations among software product features and their underlying development processes.

Notwithstanding, our approach provides new insights on the software development process, such as identification of correlations between product and process dimensions, prediction capabilities around coding practices, and characterization of developers' behaviors.

Table 6.1: Summary of findings

	Research Questions	Results / Findings
Chapter III RP1	**RQ1**	From an event log containing developers IDE commands/actions, we were able to model teams' behavior with moderate-to-strong Fitness and Precision values and yet achieve readable models. The need to answer this question is vital to discover processes followed by different people, that may be using different tools, in different locations but contributing to the same final outcome or product. The importance of understanding and measure teams' dynamics has grown with the current business trends that lead to GSE and GSD, as in these kinds of projects, the usual monitoring techniques are obsolete [105]. We confirmed that, even for a well-defined software development task, there may be a great deal of process variability due to the human factor. We were able to identify when developers were more or less focused in the essential tasks they were required to perform. Less focused teams had the more complex process models, due to the spurious / non-essential actions that were carried out. In other words, they were less efficient. Experts' opinion confirmed that those teams also were less effective in their expected delivery. We therefore concluded that a self-awareness of the performed process rendered by our approach, may be used to identify corrective actions that will improve process efficiency (less wasted effort) and may yield to better deliverables, i.e. improved process effectiveness.
	RQ2	We were able to discover and reconstruct process models representing the efficiency of software development teams, where, in some cases, members were working individually, each with their own IDE setup configurations. We confirmed that process mining may play a fundamental role in assessing the efficiency of software development teams and in potentially contributing to keep them focused on their tasks by checking and enforcing compliance to the prescribed processes. Every project manager wants to have in the projects he/she manages the most efficient and/or adequate resources. As this is expected to increase productivity in the development, measuring which teams or individuals are more efficient is a step further for better planning future software development projects.

Continued on next page

Table 6.1 – continued from previous page

	Research Questions	Results / Findings
	RQ3	By assessing the way a task is executed and the proficiency achieved, we were looking if there was any relation between those on the software development realm. In general, teams with less complexity in their models were among the most proficient in the task. This means that, they not only understood what was requested, but also had the maturity to deliver what was expected by following a simple process. They were not only effective, they were also efficient by being focused in the task. On the contrary, teams with insufficient proficiency produced long and complex models or, in very short time, they created very fuzzy models with too many generic events. These teams were the ones where more risk aroused from a development project perspective due to their erratic behavior and uncertainty around the expected deliveries. Some of those teams did not perform very well and quality was impacted, and some others did not even deliver what was expected. In a real-world scenario, these teams would have been identified as the most expensive teams because their productivity was indeed very low. This gives us some evidence that teams' proficiency can be inferred by analyzing mined process models representing their behavior.
Chapter IV RP2	**RQ4**	We confirm, as expected that teams using a plugin for refactoring ***(JDeodorant)*** reduced software complexity more effectively and with simpler processes than the ones that performed refactoring using only ***Eclipse*** native features.
	RQ5	We found moderate correlations **(≈43%)** between software cyclomatic complexity and process cyclomatic complexity in manual refactoring tasks.
	RQ6,RQ7	Using only process driven metrics, we computed ≈30,000 models aiming to predict the type of refactoring method (automatic or manual) teams had used and the expected level of software cyclomatic complexity reduction after their work sessions. The best models found for the refactoring method and cyclomatic complexity level predictions, had an accuracy of **92.95%** and **94.36%**, respectively.

Continued on next page

Table 6.1 – continued from previous page

	Research Questions	Results / Findings
Chapter V RP3	**RQ8**	From a syntactic structure perspective on the development sessions, we observed that, although English has a higher level of cross-entropy across all n-gram models, it declines rapidly, saturating around tri- or 4-grams. The same happens with development sessions models, which have generally lower cross-complexity for unigram models, and also saturate around tri-grams models. This indicates, as expected, that development sessions repetitive context can also be captured by language models. Regarding the semantics, cross-entropy grows with the order of the n-grams. The higher the n-gram window the less the model is able to predict future cases because the perplexity is higher. Regarding the number of topics on each n-gram model, we can confirm the expected behavior, when the number of topics increase, independently of the n-gram model,the entropy tends to decline.
	RQ9	Findings show that we can clearly characterize with a coherent rationale most developers, and distinguish the top performers from the ones with more challenging behaviors. This approach may lead ultimately to the creation of a catalog of software development process smells. It was striking to observe that the fingerprints characterizing the high performers are significantly different, either in the top activities and also in probabilities, from the ones describing the low performers.
	RQ10	We confirmed the significant variance between the developers with less quality in their code(Bottom5) and the rest of the participants (Top5 and Others), and this difference is larger between the top five and bottom five performers. Process models discovery and quality metrics provide evidences to state that, the differences in the proficiency between certain participants was not only related with the coding skills each of them may have. The process complexity followed by each developer may also influence the outcome, or at least, may be used as a valid indicator for assessing quality between developers.

6.3 Overall Benefits

We demonstrated the feasibility of building cross-cutting analytical models in software projects, such as the one used for detecting manual or automatic refactoring practices. Events from the development tools and support activities can be collected, transformed, aggregated, and analyzed with fewer privacy concerns or technical constraints than source code-driven metrics. This makes our approach agnostic to programming languages, geographic location, or development practices, making it suitable for challenging contexts such as in current global software development projects. Initial findings are encouraging and lead us to suggest practitioners may use our approach in other development tasks, such as defect analysis and unit or integration tests. Among others, we foresee that our contributions may provide benefits for the following stakeholders: IDE providers, consultants, practitioners, and researchers.

For IDE Providers. The last generation of traditional IDEs[2], and those hosting their services in the cloud[3], provide a plethora of functionalities, such as code completion, automated packaging, and optimized continuous integration features to assist programmers in their daily activities. These platforms ground their functionality on integration with source code repositories[4] and provide consistent connections with issue/bug tracking systems[5] to manage ongoing projects.

Many of these IDEs offer the possibility to collect event data and usage statistics from the development sessions. While the goal of this feature is to provide data for other vendors to improve their own products, we believe there might be another purpose for this data - the opportunity to build native repositories, similar to the ones that exist for the source code but, instead, for storing and dealing with development events. A niche market, therefore, seems to exist for analytics services associated with the software development process, as an add-on for software project management platforms. Cloud-based IDEs are particularly suitable for this purpose since their underlying platforms[6] allow the combination of development data, event collection, event management, and storage in one single place.

Our approach allows software development managers to understand on-the-fly the development status without using project management tools, which are often not updated, or without code repositories statistic tools. In near real-time, they can check the processes being followed by individual developers or the teams they manage. Continuous conformance checking is another feature allowed by this approach as it allows us to compare modeled processes against their real executions.

This approach might be the birth of a set of methods for forensic analysis on software development projects since we can combine source code evolution with process evolution in one single source-of-truth.

For Consultants and Practitioners. Process mining may be applied in finding bottlenecks, deviations, or performance issues within software development processes, the kind of things that

[2] e.g. Eclipse, Visual Studio Code, IntelliJ IDEA, PyCharm, NetBeans, Sublime Text or Brackets
[3] e.g. Cloud9, Codio, Glitch, Gitpod or Azure Visual Studio Code Online
[4] e.g. Github, BitBucket, SourceForge, ProjectLocker or GitLab
[5] e.g. Bugzilla, Jira, Airbrake, Backlog, Mantis or Redmine
[6] e.g. Azure, AWS or Google Cloud

usually trigger software process improvement programs within many software development organizations or departments. The knowledge related to developers' behavior while using an IDE can be the basis to design innovative consultancy services. For instance, the availability of APIs, cloud components, and serverless technologies, also known as Function as a Service (FaaS), may be used to analyze software development processes on-demand. Clients therefore would not need to maintain the infrastructure to support their decision-making processes within the software development domain.

Our proposed approach can be particularly relevant in cases where product metrics are not available or are difficult to obtain. It can be also a valid approach to measure and monitor productivity within and between software teams. As we showed by analyzing sessions' complexity and software cyclomatic complexity variance, non-efficient teams can be easily detected. Our approach easily supports real-time data collection from individuals located in different geographic zones and with a multitude of development environments. Because data collection is not dependent on code repositories and is decoupled from check-ins and/or commits, process and code analysis can be performed before repositories are updated. Development organizations can leverage this approach to apply conformance checking methods to verify adherence of developers' practices with internally prescribed processes. This facilitates mainly the detection of low-performance practices and may trigger quick correction actions by project managers.

For Researchers. Some limitations are related to data collected, as IDE events alone are not enough to provide a holistic perspective on developers' daily activities. The combination with other events flowing from the operating system can provide an improved understanding of the software development process, using the same or alternative process mining methods and tools.

Our approach can be useful in cases where educators want to assess development profiles within a group of students, before and after classes are taught. It can be used to measure and monitor productivity within and between software teams. As we showed, by analyzing product and process complexities, non-efficient developers can easily be tracked.

Current work can be expanded in breadth and in-depth. We mostly explored the control-flow perspective, but others are worth exploring, such as the organizational and performance perspectives. Devising team dynamics based upon the identification of artifacts impacted/-touched by developers can also be another interesting research path.

6.4 Research Opportunities

It is important to understand what went wrong or unplanned, based upon past process instances, to enable just-in-time corrective actions while the process is being executed. The IDE-based process mining architecture presented in this dissertation is the base of our Software Process On-the-run Tracking System (SPOTS) platform, that will provide near real-time software development process insights, at the individual or team level, such as in the PSP [93], or TSP approaches [94], but in an automated fashion. According to [206], this kind of operational support is the most advanced form of process mining action. Initial research artifacts are presented in Appendix A in sections A.2, A.3 and A.4.

There is also margin to research how the development process smells [211] may be used to assess software process drift management. Machine learning techniques are plausible candidates to automatically classify mined models as good or bad process smells.

6.4.1 Software Development Process Mining Microservices

The constant change in organizations' business requirements often induces the modification of software application design. Such demand requires software development teams to also find innovative software architecture styles to accommodate application scalability, development flexibility, and adaptability to evolving requirements. However, many organizations have single-tiered software applications in which all functionality is bundled into a single program for a single platform. Those so-called "monolithic"applications, where components are usually highly interwoven, rather than being architecturally separate, jeopardize the accommodation of the aforementioned change. To mitigate problems arisen by traditional monolithic software systems, organizations are migrating to MicroServices Architectures (MSA), where applications' functionality is broken into a small set of services, each running independently, often deployed in a virtualized environment [116]. The MSA, a variant of the Service-Oriented Architecture (SOA) structural style, arranges an application as a collection of loosely coupled services and brings forth numerous benefits, whilst from the user's perspective, the interaction with the system remains the same. MSA has been also introduced in the context of software development processes connected to the DevOps realm, where development and continuous deployment are closely linked [194].

Many of the aggregation, transformation, and analysis methods we used in this work can be ported to a public, private, or hybrid cloud platform with a MSA approach. Services can be organized around software development process mining capabilities and be implemented using different programming languages, databases, hardware, and software environment, depending on what fits best. They should remain small in size, messaging-enabled, bounded by the software development process context, autonomously developed, and independently deployable. The blockchain protocol can help in decentralizing, building, and releasing such microservices with automated and yet secure processes [197]. It can also foster innovation if included from the ground in platforms aiming to reward people according to the experiment data they have shared with the community.

6.4.2 Software Development Process Mining Pipeline

As mentioned earlier, traditional software repositories have limitations and imprecisions. To expand the analytics coverage on mining software development processes, we should explore non trivially used repositories, such as the IDE. This is particularly interesting to drive studies aiming to combine development perspectives: i) product quality and ii) the underlying development process.

However, to streamline such research, many manual tasks need to be properly automated, considering that most process mining tools are not ready for non-human intervention. Due to this reality, multiple metrics in this work had to be extracted semi-automatically, using a tool but not dispensing user interaction. This is a strong limitation in advancing the research based

on event data and current process mining methods. A MSA approach seems to be a good fit for building a coherent, automated, and orchestrated pipeline for software development process mining using the earlier identified microservices.

Additionally, research combining software product and process data is scarce and experiments in this area are difficult to design and execute. To mitigate this problem, we expect an increment in shared datasets containing this hybrid data, providing that privacy and/or anonymity on sensitive information is guaranteed. This may be facilitated if the suggested pipeline building blocks are available.

6.4.3 Low and No Code Paradigms

Modern software development projects incorporate low and no code practices to enable faster development cycles requiring little to no coding in order to build and deliver applications and processes. Low-code development platforms are seen as advanced IDEs which employ drag-and-drop software components and visual interfaces to replace extensive coding. They provide higher levels of abstraction that allow a major reduction in hand-coding to develop an application by means of high-level visual modeling languages [86]. This paradigm shift in software development may also require a change in the way we assess critical properties of a software project, such as quality, maintainability, and evolvability.

The approach proposed in this dissertation is applicable in low and no-code platforms, where textual programming is brought to a minimum or even banned, giving rise to the so-called *citizen developers*. It seems more adequate to assess product complexity and project effort estimation based on the interaction events with the platform's IDE, instead of evaluating the automatically generated code, since those events allow identifying the composed components for building an app, as well as citizen developers' behavior.

6.4.4 Forensic Readiness in Software Development Processes

Digital forensics is the process of investigating a computer system to determine the cause of an incident [189]. Although forensic readiness is a well-known concept, addressing it in the context of the software development process is a recent endeavor [56]. Software development process forensics is therefore the process of investigating the sequence of activities, events, and decisions to determine the cause of error-prone components or failed projects.

The approach proposed in this dissertation opens the opportunity for new research related to forensic analysis on software development processes, exploring a combined perspective of produced artifacts along with the underlying processes. As mentioned in section 1.1.1, it is somehow the equivalent of combining the information collected by the FDR and the CVR. This analysis is typically in the interest of figuring out what happened, when it happened, how it happened, and who was involved in software project failures from a process perspective.

Closure. On revising and summarizing the evidence provided by this dissertation, we have to encourage others to pursue this investigation. As Alvin Toffler suggested us in his legacy book *"Future Shock"* fifty years ago, even if it seems easier to focus on software artifacts - the *"things"*, we should pay attention to software development interactions - the *"processes"*.

We have strong evidence to think that Process Mining is vital in software process comprehension, especially when novel data sources are added to the equation. Prediction based on event data captured from the IDE is possible and overall event-based software project predictions (including financial aspects and human resources involved) is a long term and challenging journey, but yet, an achievable goal. ■■

[This page has been intentionally left blank]

Bibliography

[1] M. Abdellatif, F. Capretz, and D. Ho. "Software Analytics to Software Practice: A Systematic Literature Review." In: *1st International Workshop on Big Data Software Engineering*. IEEE/ACM, 2015, pp. 30–36.

[2] F Brito e Abreu, R Esteves, and M Goulão. "The Design of Eiffel Programs: Qualitative Evaluation using the MOOD Metrics." In: *Proc. of 20th International Conference on Technology of Object Oriented Languages and Systems (TOOLS'96 USA)*. Ed. by R. Ege. Santa Barbara, USA: Zenodo, 1996, pp. –.

[3] F. Brito e Abreu. *Using OCL to formalize object oriented metrics definitions*. Tech. rep. ES007/2001. INESC, May 2001.

[4] A. Agrawal, W. Fu, D. Chen, X. Shen, and T. Menzies. "How to "DODGE"Complex Software Analytics?" In: *Transactions in Software Engineering* (Feb. 2019).

[5] A. Agrawal, W. Fu, and T. Menzies. "What is Wrong with Topic Modeling? (and How to Fix it Using Search-based Software Engineering)." In: *Information and Software Technology Journal* (Aug. 2018).

[6] F. Akiyama. "An Example of Software System Debugging." In: *IFIP Congress (1)*. Ed. by C. V. Freiman, J. E. Griffith, and J. L. Rosenfeld. North-Holland, 1971, pp. 353–359.

[7] A. Aldahmash, A. M. Gravell, and Y. Howard. "A Review on the Critical Success Factors of Agile Software Development." In: *European Conference on Software Process Improvement*. Springer, Cham, 2017, pp. 504–512.

[8] M. Aniche, E. Maziero, R. Durelli, and V. H. S. Durelli. *The Effectiveness of Supervised Machine Learning Algorithms in Predicting Software Refactoring*. Tech. rep. Delft University of Technology, 2020.

[9] H. Anwar and D. Pfahl. "Towards greener software engineering using software analytics: A systematic mapping." In: *Proceedings - 43rd Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2017*. Institute of Electrical and Electronics Engineers Inc., Sept. 2017, pp. 157–166.

[10] P. Ardimento, M. L. Bernardi, M. Cimitile, and G. De Ruvo. "Mining Developer's Behavior from Web-Based IDE Logs." In: *2019 IEEE 28th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE)*. IEEE, June 2019, pp. 277–282.

[11] P. Ardimento, M. L. Bernardi, M. Cimitile, and F. M. Maggi. "Evaluating Coding Behavior in Software Development Processes: A Process Mining Approach." In: *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. IEEE, May 2019, pp. 84–93.

[12] P. Ardimento, M. L. Bernardi, M. Cimitile, and G. D. Ruvo. "Learning analytics to improve coding abilities: A fuzzy-based process mining approach." In: *IEEE International Conference on Fuzzy Systems*. Vol. 2019-June. Institute of Electrical and Electronics Engineers Inc., June 2019, pp. 1–7.

[13] R. Arun, V. Suresh, C. E. Madhavan, and M. N. Murty. "On finding the natural number of topics with Latent Dirichlet Allocation: Some observations." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 6118 LNAI. 2010, pp. 391–402.

[14] Association for Computing Machinery Special Interest Group on Software Engineering. "Software Developers Perceptions of Productivity." In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*.

[15] S. Astromskis, A. Janes, and M. Mairegger. "A process mining approach to measure how users interact with software: an industrial case study." In: *Proceedings of the 2015 International Conference on Software and System Process - ICSSP 2015*. 2015.

[16] *Aviation Accident Statistics, http://www.planecrashinfo.com/cause.htm*. 2020.

[17] L. Bao, Z. Xing, X. Xia, D. Lo, and A. E. Hassan. "Inference of development activities from interaction with uninstrumented applications." In: *Empirical Software Engineering* 23.3 (June 2018), pp. 1313–1351.

[18] L. Bao, Z. Xing, X. Xia, D. Lo, and A. E. Hassan. "Inference of development activities from interaction with uninstrumented applications." In: *Empirical Software Engineering* 23.3 (June 2018), pp. 1313–1351.

[19] F. A. Batarseh and A. J. Gonzalez. "Predicting failures in agile software development through data analytics." In: *Software Quality Journal* 26.1 (Mar. 2018), pp. 49–66.

[20] K. Beck, M. Fowler, J. Brant, W. Opdyke, and D. Roberts. "Bad Smells in Code." In: *Improving the design of existing code*. O'Reilly, 1999. Chap. 3, pp. –.

[21] M. Beller, G. Gousios, A. Panichella, S. Proksch, S. Amann, and A. Zaidman. "Developer Testing in the IDE: Patterns, Beliefs, and Behavior." In: *IEEE Transactions on Software Engineering* 45.3 (Mar. 2019), pp. 261–284.

[22] P. Berander. "Using students as subjects in requirements prioritization." In: *Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE'04.* IEEE. 2004, pp. 167–176.

[23] A. Berti and W. van der Aalst. *A Novel Token-Based Replay Technique to Speed Up Conformance Checking and Process Enhancement*. Tech. rep. Aachen: Process and Data Science group - Lehrstuhl fur Informatik 9 52074, RWTH Aachen University, July 2020.

[24] D. M. Blei, A. Y. Ng, and M. I. Jordan. "Latent Dirichlet Allocation." In: *Journal of Machine Learning Research* 3 (2003), pp. 993–1022.

[25] B. W. Boehm. *Software Engineering Economics*. 1st. Upper Saddle River, NJ, United States: Prentice Hall PTR, 1981, p. 768.

[26] A. Bolt, W. M. van der Aalst, and M. de Leoni. "Finding process variants in event logs: (Short Paper)." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 10573 LNCS. Springer Verlag, 2017, pp. 45–52.

[27] A. Bolt, M. de Leoni, and W. M. van der Aalst. "Process variant comparison: Using event logs to detect differences in behavior and business rules." In: *Information Systems* 74 (May 2018), pp. 53–66.

[28] A. Bolt, M. de Leoni, and W. M. van der Aalst. "Process variant comparison: Using event logs to detect differences in behavior and business rules." In: *Information Systems* 74.1 (May 2018), pp. 53–66.

[29] M. Borges Ribeiro, V. Diniz Duarte, E. Gomes Salgado, and C. Vieira Castro. "Prioritization of Critical Success Factors In The Process of Software Development." In: *IEEE Latin America Transactions* 15.1 (Jan. 2017), pp. 137–144.

[30] G. Botterweck and C. Werner, eds. *Mastering Scale and Complexity in Software Reuse*. Vol. 10221 LNCS. Springer Verlag, 2017, pp. –.

[31] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. "Class-Based N-Gram Models of Natural Language." In: *Computational Linguistics* 18.4 (Dec. 1992), pp. 467–479.

[32] M. Bruch, E. Bodden, M. Monperrus, M. Mezini, and M. M. Ide. "IDE 2.0: Collective Intelligence in Software Development." In: (2010).

[33] J. C. Buijs, B. F. Van Dongen, and W. M. Van Der Aalst. "Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity." In: *International Journal of Cooperative Information Systems* 23.1 (2014).

[34] R. P. L. Buse and T. Zimmermann. *Analytics for Software Development*. Tech. rep. Microsoft Research, 2010.

[35] R. P. Buse and T. Zimmermann. "Information needs for software development analytics." In: *Proceedings - International Conference on Software Engineering*. 2012, pp. 987–996.

[36] J. Caldeira and F. Brito e Abreu. "Software development process mining: Discovery, conformance checking and enhancement." In: *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*. IEEE, Sept. 2016, pp. 254–259.

[37] J. Caldeira, F. Brito e Abreu, J. Reis, and J. Cardoso. "Assessing Software Development Teams' Efficiency using Process Mining." In: *2019 International Conference on Process Mining (ICPM)*. Institute of Electrical and Electronics Engineers (IEEE), Aug. 2019, pp. 65–72.

[38] J. Cao, T. Xia, J. Li, Y. Zhang, and S. Tang. "A density-based method for adaptive LDA model selection." In: *Neurocomputing* 72.7-9 (Mar. 2009), pp. 1775–1781.

[39] J. Cardoso, J. Mendling, G. Neumann, and H. A. Reijers. "A discourse on complexity of process models." In: *International Conference on Business Process Management*. Springer. 2006, pp. 117–128.

[40] G. Casale, C. Chesta, P. Deussen, E. Di Nitto, P. Gouvas, S. Koussouris, V. Stankovski, A. Symeonidis, V. Vlassiou, A. Zafeiropoulos, and Z. Zhao. "Current and Future Challenges of Software Engineering for Services and Applications." In: *Procedia Computer Science*. Vol. 97. Elsevier B.V., 2016, pp. 34–42.

[41] C. Chen, Z. Xing, and Y. Liu. "What's Spain's Paris? Mining analogical libraries from Q&A discussions." In: *Empirical Software Engineering* 24.3 (June 2019), pp. 1155–1194.

[42] L. Chen and M. A. Babar. "A systematic review of evaluation of variability management approaches in software product lines." In: *Information and Software Technology* 53.4 (2011), pp. 344–362.

[43] T. H. Chen, S. W. Thomas, and A. E. Hassan. "A survey on the use of topic models when mining software repositories." In: *Empirical Software Engineering* 21.5 (Oct. 2016), pp. 1843–1919.

[44] M. Chinosi and A. Trombetta. "BPMN: An introduction to the standard." In: *Computer Standards & Interfaces* 34.1 (2012), pp. 124–134.

[45] T. Chow and D.-B. Cao. "A survey study of critical success factors in agile software projects." In: *Journal of Systems and Software* 81.6 (June 2008), pp. 961–971.

[46] J. E. Cook and A. L. Wolf. "Discovering Models of Software Processes from Event-Based Data." In: *ACM Transactions on Software Engineering and Methodology*. Vol. 7. ACM, July 1996, pp. 215–249.

[47] V. Cosentino, J. L. Izquierdo, and J. Cabot. "A Systematic Mapping Study of Software Development with GitHub." In: *IEEE Access* 5 (2017), pp. 7173–7192.

[48] P. B. Crosby. *Quality is free : the art of making quality certain*. McGraw-Hill, 1979, p. 309.

[49] L. Cruz, R. Abreu, and D. Lo. "To the attention of mobile software developers: guess what, test your app!" In: *Empirical Software Engineering* 24 (2019), 2438–2468.

[50] B. Curtis, S. B. Sheppard, and P. Milliman. "Third Time Charm: Stronger Prediction of Programmer Performance by Software Complexity Metrics." In: *Proceedings of the 4th International Conference on Software Engineering*. ICSE '79. IEEE Press, 1979, pp. 356–360.

[51] K. Damevski, H. Chen, D. C. Shepherd, N. A. Kraft, and L. Pollock. "Predicting future developer behavior in the IDE using topic models." In: *IEEE Transactions on Software Engineering* 44.11 (Nov. 2018), pp. 1100–1111.

[52] K. Damevski, D. C. Shepherd, J. Schneider, and L. Pollock. "Mining Sequences of Developer Interactions in Visual Studio for Usage Smells." In: *IEEE Transactions on Software Engineering* 43.4 (Apr. 2017), pp. 359–371.

[53] K. Damevski, D. C. Shepherd, J. Schneider, and L. Pollock. "Mining Sequences of Developer Interactions in Visual Studio for Usage Smells." In: *IEEE Transactions on Software Engineering* 43.4 (Apr. 2017), pp. 359–371.

[54] J. H. M. Daniel Jurafsky. *Speech and Language Processing*. third. Pearson Prentice Hall, 2020.

[55] S. Dasanayake, J. Markkula, and M. Oivo. "Concerns in software development: A systematic mapping study." In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*. Association for Computing Machinery, 2014, pp. 1–4.

[56] L. Daubner, M. MacAk, B. Buhnova, and T. Pitner. "Verification of forensic readiness in software development: A roadmap." In: *Proceedings of the ACM Symposium on Applied Computing*. Association for Computing Machinery, Mar. 2020, pp. 1658–1661.

[57] T. H. Davenport, J. G. Harris, and R. Morison. *Analytics at work : smarter decisions, better results*. Harvard Business Press, 2010.

[58] W. E. Deming. *Out of the Crisis: Quality, Productivity and Competitive Position*. Massachusetts Institute of Technology, Center for advanced engineering study, 1986.

[59] R. Deveaud, E. Sanjuan, P. Bellot, and E. SanJuan. "Accurate and Effective Latent Concept Modeling for Ad Hoc Information Retrieval." In: *Document Numérique, Lavoisier*, (2014), pp. 61–84.

[60] A. P. Dillon and S. Shingo. *Zero Quality Control : Source Inspection and the Poka-yoke System i Translated by With a preface by Zero Quality Control :* 1986, pp. 1–200.

[61] T. Dybå and T. Dingsøyr. "Strength of Evidence in Systematic Reviews in Software Engineering." In: *ESEM'08: Proceedings of the 2008 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*. 2008, pp. 178–187.

[62] K. E. Emam and A. G. Koru. "A Replicated Survey of IT Software Project Failures." In: *IEEE Software* 25.5 (2008), pp. 84–90.

[63] B. M. Enis, K. K. Cox, and J. E. Stafford. "Students as subjects in consumer behavior experiments." In: *Journal of Marketing Research* 9.1 (1972), pp. 72–74.

[64] S. A. Fahrenkrog-Petersen and W. M. van der Aa Hanand. "PRIPEL: Privacy-Preserving Event Log Publishing Including Contextual Information." In: *Business Process Management*. Cham: Springer International Publishing, 2020, pp. 111–128.

[65] Y. Fan, X. Xia, D. Lo, and S. Li. "Early prediction of merged code changes to prioritize reviewing tasks." In: *Empirical Software Engineering* 23.6 (Dec. 2018), pp. 3346–3393.

[66] A. V. Feigenbaum. *Total Quality Control, 4th Ed.: Achieving Productivity, Market Penetration, and Advantage in the Global Economy*. McGraw-Hill Higher Education, 2005, p. 896.

[67] J. Finlay, R. Pears, and A. M. A. Connor. "Data stream mining for predicting software build outcomes using source code metrics." In: *Information and Software Technology* 56.2 (2014), pp. 183–198.

[68] W. Fu and T. Menzies. "Easy over hard: A case study on deep learning." In: *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. Vol. Part F130154. Association for Computing Machinery, Aug. 2017, pp. 49–60.

[69] W. Fu, T. Menzies, and X. Shen. "Tuning for software analytics: Is it really necessary?" In: *Information and Software Technology* 76 (Aug. 2016), pp. 135–146.

[70] D. Fucci and B. Turhan. "On the role of tests in test-driven development: A differentiated and partial replication." In: *Empirical Software Engineering* 19.2 (2014), pp. 277–302.

[71] A. Fuggetta and E. Di Nitto. "Software process." In: *Proceedings of the on Future of Software Engineering - FOSE 2014*. New York, New York, USA: ACM Press, 2014, pp. 1–12.

[72] A. Fuggetta, E. D. Nitto, and P. Milano. "Software Process." In: *Proceedings of the on Future of Software Engineering* (2014), pp. 1–12.

[73] C. d. S. Garcia, A. Meincheim, E. R. Faria Junior, M. R. Dallagassa, D. M. V. Sato, D. R. Carvalho, E. A. P. Santos, and E. E. Scalabrin. "Process mining techniques and applications – A systematic mapping study." In: *Expert Systems with Applications* 133 (Nov. 2019), pp. 260–295.

[74] D. A. Garvin. *Managing quality : the strategic and competitive edge*. Free Press, 1988, p. 319.

[75] J. P. G. Gomes. "Learning to code in class with MOOCs: process, factors and outcomes." Supervision: Cláudia Werner (COPPE/UFRJ) and Fernando Brito e Abreu (ISTAR/Iscte. Master in Computer Science and Business Management. Lisbon, Portugal: Instituto Universitário de Lisboa (Iscte), 2020.

[76] T. L. Gomes, T. C. Oliveira, D. Cowan, and P. Alencar. "Mining reuse processes." In: *CIBSE 2014: Proceedings of the 17th Ibero-American Conference Software Engineering*. Pucon, Chile: Curran Associates, 2014, pp. 179–191.

[77] T. L. Griffiths and M. Steyvers. "Finding scientific topics." In: *Proceedings of the National Academy of Sciences of the United States of America* 101.SUPPL. 1 (Apr. 2004), pp. 5228–5235.

[78] D. Güemes-Peña, C. López-Nozal, R. Marticorena-Sánchez, and J. Maudes-Raedo. "Emerging topics in mining software repositories: Machine learning in software repositories and datasets." In: *Progress in Artificial Intelligence* 7.3 (Sept. 2018), pp. 237–247.

[79] L. Guerrouj, Z. Kermansaravi, V. Arnaoudova, B. C. Fung, F. Khomh, G. Antoniol, and Y. G. Guéhéneuc. "Investigating the relation between lexical smells and change- and fault-proneness: an empirical study." In: *Software Quality Journal* 25.3 (Sept. 2017), pp. 641–670.

[80] C. Günther and E Verbeek. *XES standard definition*. Tech. rep. BPMcenter. org, 2014.

[81] G. M. Hampton. "Students as subjects in international behavioral studies." In: *Journal of International Business Studies* (1979), pp. 94–96.

[82] T. Hariprasad, G. Vidhyagaran, K. Seenu, and C. Thirumalai. "Software complexity analysis using halstead metrics." In: *Proceedings - International Conference on Trends in Electronics and Informatics, ICEI 2017*. Vol. 2018-January. Institute of Electrical and Electronics Engineers Inc., Feb. 2018, pp. 1109–1113.

[83] A. E. Hassan, A. Hindle, M. Shepperd, P. Devanbu, S. Kim, and H. Kong. *What's Next in Software Analytics*. Tech. rep.

[84] S. Hassan, W. Shang, and A. E. Hassan. "An empirical study of emergency updates for top android mobile apps." In: *Empirical Software Engineering* 22.1 (Feb. 2017), pp. 505–546.

[85] S. Hassan, C. Tantithamthavorn, C. P. Bezemer, and A. E. Hassan. "Studying the dialogue between users and developers of free apps in the Google Play Store." In: *Empirical Software Engineering* 23.3 (June 2018), pp. 1275–1312.

[86] H. Henriques, H. Lourenço, V. Amaral, and M. Goulão. "Improving the developer experience with a low-code process modelling language." In: *Proceedings - 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018*. Association for Computing Machinery, Inc, Oct. 2018, pp. 200–210.

[87] S Henry and D Kafura. "Software Structure Metrics Based on Information Flow." In: *IEEE Trans. Softw. Eng.* 7.5 (Sept. 1981), pp. 510–518.

[88] J. Herbsleb. "Building a Socio-Technical Theory of Coordination: Why and How (Outstanding Research Award)." In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, pp. 2–10.

[89] A. Herczyński, C. Cernuschi, and L. Mahadevan. "Painting with drops, jets, and sheets." In: *Physics Today* 64.6 (June 2011), pp. 31–36.

[90] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu. "On the Naturalness of Software." In: *Proceedings of the 34th International Conference on Software Engineering*. Zurich, Switzerland: IEEE Press, 2012, 837–847.

[91] B. Hompes, W. Van Der Aalst, P. Dixit, B. F. A. Hompes, J. C. A. M. Buijs, W. M. P. Van Der Aalst, P. M. Dixit, and J Buurman. *Discovering Deviating Cases and Process Variants Using Trace Clustering*. Tech. rep. Department of Mathematics and Computer Science Eindhoven University of Technology, Eindhoven, The Netherlands, 2015.

[92] M. Höst, B. Regnell, and C. Wohlin. "Using students as subjects—a comparative study of students and professionals in lead-time impact assessment." In: *Empirical Software Engineering* 5.3 (2000), pp. 201–214.

[93] W. S. Humphrey. "Personal Software Process (PSP)." In: *Encyclopedia of Software Engineering*. Ed. by J. J. Marciniak. 2nd. New York, NY, USA: John Wiley & Sons, 2002, p. 1584.

[94] W. S. Humphrey. "Team Software Process (TSP)." In: *Encyclopedia of Software Engineering* (2002).

[95] IEEE Computer Society. *SWEBOK V3.0*. 1 V3.0. IEEE Computer Society, 2014, p. 346.

[96] C Ioannou, A Burattin, and B Weber. "Mining Developers' Workflows from IDE Usage." In: *Lecture Notes in Business Information Processing*. Vol. 316. Springer, 2018, pp. 167–179.

[97] C Ioannou, A Burattin, and B Weber. "Mining Developers' Workflows from IDE Usage." In: *Lecture Notes in Business Information Processing*. Vol. 316. Springer, 2018, pp. 167–179.

[98] K. Ishikawa. *What is total quality control? The Japanese way*. Prentice-Hall, 1985, p. 215.

[99] D. Izquierdo-Cortazar, N. Sekitoleko, J. M. Gonzalez-Barahona, and L. Kurth. "Using Metrics to track code review performance." In: *ACM International Conference Proceeding Series*. Vol. Part F128635. Association for Computing Machinery, June 2017, pp. 214–223.

[100] *Jackson Pollock*. The American Museum of Beat Art.

[101] *Jackson Pollock's Unique Style*.

[102] S. B. Jagtap and B. G. Kodge. "Census Data Mining and Data Analysis using WEKA." In: *International Conference in Emerging Trends in Science, Technology and Management-2013*. Singapore, 2013, pp. –.

[103] A. K. Jha, S. Lee, and W. J. Lee. "An empirical study of configuration changes and adoption in Android apps." In: *Journal of Systems and Software* 156 (Oct. 2019), pp. 164–180.

[104] J. Jiang, D. Lo, J. He, X. Xia, P. S. Kochhar, and L. Zhang. "Why and how developers fork what from whom in GitHub." In: *Empirical Software Engineering* 22.1 (Feb. 2017), pp. 547–578.

[105] F. Jurado and P. Rodriguez. "Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues." In: *Journal of Systems and Software* 104 (2015), pp. 82–89.

[106] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. "An in-depth study of the promises and perils of mining GitHub." In: *Empirical Software Engineering* 21.5 (Oct. 2016), pp. 2035–2071.

[107] N. Kaoungku, K. Suksut, R. Chanklan, K. Kerdprasop, and N. Kerdprasop. "The silhouette width criterion for clustering and association mining to select image features." In: *International Journal of Machine Learning and Computing* 8.1 (Feb. 2018), pp. 69–73.

[108] H. Karna, L. Vicković, and S. Gotovac. "Application of data mining methods for effort estimation of software projects." In: *Software - Practice and Experience*. Vol. 49. John Wiley and Sons Ltd, Feb. 2019, pp. 171–191.

[109] M. Kersten and G. C. Murphy. "Using Task Context to Improve Programmer Productivity." In: *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. Portland, Oregon, USA: ACM, 2006, pp. 1–11.

[110] M. Kim, D. Cai, and S. Kim. "An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution." In: *2011 33rd International Conference on Software Engineering (ICSE)*. Honolulu, HI, USA, 2011, pp. 151–160.

[111] M. Kim, T. Zimmermann, R. DeLine, and A. Begel. "The emerging role of data scientists on software development teams." In: *Proceedings - International Conference on Software Engineering*. Vol. 14-22-May-2016. IEEE Computer Society, May 2016, pp. 96–107.

[112] M. Kim, T. Zimmermann, and N. Nagappan. "An Empirical Study of Refactoring Challenges and Benefits at Microsoft." In: *Transactions on Software Engineering* (2014), 633–649.

[113] B. Kitchenham and P. Brereton. "A systematic review of systematic review process research in software engineering." In: *Information and Software Technology* 55.12 (2013), pp. 2049–2075.

[114] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. "Systematic literature reviews in software engineering - A systematic literature review." In: *Information and Software Technology* 5 (2009), pp. 7–15.

[115] S. C. Kotakonda and R. Engu. *Are Students Good Proxies for Studying Professional: A Systematic Literature Review*. 2012.

[116] A. Kwan, H.-A. Jacobsen, A. Chan, and S. Samoojh. "Microservices in the Modern Software World." In: *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*. CASCON '16. USA: IBM Corp., 2016, pp. 297–299.

[117] J. P. Leal and F. Silva. "Mooshak: A Web-based multi-site programming contest system." In: *Software - Practice and Experience* 33.6 (May 2003), pp. 567–581.

[118] M Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand. "Recursion aware modeling and discovery for hierarchical software event log analysis." In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 2018, pp. 185–196.

[119] M. Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand. "The Statechart Workbench: Enabling scalable software event log analysis using process mining." In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, Mar. 2018, pp. 502–506.

[120] A. M. Lemos, C. C. Sabino, R. M. F. Lima, and a. L. Oliveira. "Conformance Checking of Software Development Processes Through Process Mining." In: *Seke* (2011).

[121] P. Lerthathairat and N. Prompoon. "An Approach for Source Code Classification Using Software Metrics and Fuzzy Logic to Improve Code Quality with Refactoring Techniques." In: *International Conference on Software Engineering and Computer Systems (ICSECS 2011)*. 2011, p. 830.

[122] H. Li, T. H. P. Chen, W. Shang, and A. E. Hassan. "Studying software logging using topic models." In: *Empirical Software Engineering* 23.5 (Oct. 2018), pp. 2655–2694.

[123] H. Li, W. Shang, Y. Zou, and A. E. Hassan. "Towards just-in-time suggestions for log changes." In: *Empirical Software Engineering* 22.4 (Aug. 2017), pp. 1831–1865.

[124] Y. Liu, J. Wang, L. Wei, C. Xu, S. C. Cheung, T. Wu, J. Yan, and J. Zhang. "DroidLeaks: a comprehensive database of resource leaks in Android apps." In: *Empirical Software Engineering* 24.6 (2019), pp. 3435–3483.

[125] L. Madeyski, M. Jureczko, and M Jureczko. "Which process metrics can significantly improve defect prediction models? An empirical study." In: *Software Quality Journal* 23 (2015), pp. 393–422.

[126] Maikel L. van Eck(B), Xixi Lu, Sander J.J. Leemans, and Wil M.P. van der Aalst. "A Process Mining Project Methodology." In: *International Conference on Advanced Information Systems Engineering: CAiSE 2015: Advanced Information Systems Engineering* 1626 (1999), pp. 41–56.

[127] A. R. C. Maita, L. C. Martins, C. R. López Paz, L. Rafferty, P. C. K. Hung, S. M. Peres, and M. Fantinato. "A systematic mapping study of process mining." In: *Enterprise Information Systems* 12.5 (May 2018), pp. 505–549.

[128] R. Malhotra and A. Sharma. "Analyzing machine learning techniques for fault prediction using web applications." In: *Journal of Information Processing Systems* 14.3 (2018), pp. 751–770.

[129] P. Mayer, M. Kirsch, and M. A. Le. "On multi-language software development, cross-language links and accompanying tools: a survey of professional software developers." In: *Journal of Software Engineering Research and Development* 5 (2017), p. 1.

[130] T. J. Mccabe. "A Complexity Measure." In: *IEEE Transactions on Software Engineering* SE-2.4 (Dec. 1976), pp. 308–320.

[131] S. McIlroy, N. Ali, and A. E. Hassan. "Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store." In: *Empirical Software Engineering* 21.3 (June 2016), pp. 1346–1370.

[132] P. A. McQuaid. "Software disasters—understanding the past, to improve the future." In: *Journal of Software: Evolution and Process* 24.5 (2012), pp. 459–470.

[133] A. Meidan, J. A. García-García, I. Ramos, and M. J. Escalona. "Measuring Software Process." In: *ACM Computing Surveys* 51.3 (June 2018), pp. 1–32.

[134] T. Menzies, C. Bird, T. Zimmermann, W. Schulte, and E. Kocaganeli. "The Inductive Software Engineering Manifesto: Principles for Industrial Data Mining." In: *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*. Association for Computing Machinery, 2011, 19–26.

[135] T. Menzies, L. Minku, and F. Peters. "The Art and Science of Analyzing Software Data; Quantitative Methods." In: *Proceedings - International Conference on Software Engineering*. Vol. 2. IEEE Computer Society, Aug. 2015, pp. 959–960.

[136] T. Menzies and T. Zimmermann. "Software analytics: So what?" In: *IEEE Software* 30.4 (2013), pp. 31–37.

[137] T. Menzies and T. Zimmermann. "Software Analytics: What's Next?" In: *IEEE Software Engineering* (2018), pp. 64–70.

[138] A. N. Meyer, L. E. Barton, G. C. Murphy, T. Zimmermann, and T. Fritz. "The Work Life of Developers: Activities, Switches and Perceived Productivity." In: *IEEE Transactions on Software Engineering* 43.12 (Jan. 2017), pp. 1178–1193.

[139] R. Minelli, A. Mocci, and M. Lanza. "I Know What You Did Last Summer An Investigation of How Developers Spend Their Time." In: *23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 25–35.

[140] L. L. Minku and S. Hou. "Clustering dycom an online cross-company software effort estimation study." In: *ACM International Conference Proceeding Series*. Association for Computing Machinery, Nov. 2017, pp. 12–21.

[141] M. Mittal and A. Sureka. "MIMANSA : Process Mining Software Repositories from Student Projects in an Undergraduate Software Engineering Course Categories and Subject Descriptors." In: *Software Engineering Education and Training | ICSE 2014* (2014), pp. 344–353.

[142] M. Mittal and A. Sureka. "Process mining software repositories from student projects in an undergraduate software engineering course." In: *36th International Conference on Software Engineering, ICSE Companion 2014 - Proceedings*. Association for Computing Machinery, 2014, pp. 344–353.

[143] N Moha, Y. G. Guéhéneuc, and P Leduc. "Automatic generation of detection algorithms for design defects." English. In: *21st IEEE/ACM International Conference on Automated Software Engineering, ASE 2006*. 2006, pp. 297–300.

[144] P. Mohagheghi and R. Conradi. "Quality, productivity and economic benefits of software reuse: A review of industrial studies." In: *Empirical Software Engineering* 12.5 (Oct. 2007), pp. 471–516.

[145] P. Mohagheghi and M. Jorgensen. "What Contributes to the Success of IT Projects? Success Factors, Challenges and Lessons Learned from an Empirical Study of Software Projects in the Norwegian Public Sector." In: *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, May 2017, pp. 371–373.

[146] I. Morales-Ramirez, F. M. Kifetew, and A. Perini. "Speech-acts based analysis for requirements discovery from online discussions." In: *Information Systems* 86 (2018), pp. 94–112.

[147] N. Munaiah and A. Meneely. "Vulnerability severity scoring and bounties: Why the disconnect." In: *SWAN 2016 - Proceedings of the 2nd International Workshop on Software Analytics, co-located with FSE 2016*. Association for Computing Machinery, Inc, Nov. 2016, pp. 8–14.

[148] J. Munoz-Gama and J. Carmona. "A Fresh Look at Precision in Process Conformance." In: *Business Process Management - 8th International Conference, BPM2010, Hoboken, NJ, USA, September 13-16, 2010*. Ed. by R. Hull, J. Mendling, and S. Tai. Vol. 6336. Lecture Notes in Computer Science. Springer, 2010, pp. 211–226.

[149] G. C. Murphy, M. Kersten, and L. Findlater. "How are java software developers using the eclipse IDE?" In: *IEEE Software* (2006).

[150] G. C. Murphy, P. Viriyakattiyaporn, and D. Shepherd. "Using activity traces to characterize programming behaviour beyond the lab." In: *IEEE International Conference on Program Comprehension* (2009), pp. 90–94.

[151] E. Murphy-Hill, C. Parnin, and A. P. Black. "How We Refactor, and How We Know It." In: *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009, 287–297.

[152] A. Nagpal and G. Gabrani. "Python for data analytics, scientific and technical applications." In: *2019 Amity international conference on artificial intelligence (AICAI)*. IEEE. 2019, pp. 140–145.

[153] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Tech. rep. www.bitcoin.org, 2009.

[154] M. Nayebi, G. Ruhe, R. C. Mota, and M. Mufti. "Analytics for software project management - Where are we and where do we go?" In: *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering Workshops, ASEW 2015*. Institute of Electrical and Electronics Engineers Inc., Mar. 2016, pp. 18–21.

[155] S. Negara, N. Chen, M. Vakilian, R. E. Johnson, and D. Dig. "A Comparative Study of Manual and Automated Refactorings." In: *Proceedings of the 27th European conference on Object-Oriented Programming*. Springer, Berlin, Heidelberg, 2013, pp. 552–576.

[156] S. Negara, N. Chen, M. Vakilian, R. E. Johnson, and D. Dig. "A Comparative Study of Manual and Automated Refactorings." In: *Proceedings of the 27th European conference on Object-Oriented Programming*. Springer, Berlin, Heidelberg, 2013, pp. 552–576.

[157] S. Negara, M. Vakilian, N. Chen, R. E. Johnson, and D. Dig. "Is It Dangerous to Use Version Control Histories to Study Source Code Evolution?" In: *European Conference on Object-Oriented Programming, ECOOP 2012: ECOOP 2012 – Object-Oriented Programming*. Springer, Berlin, Heidelberg, 2012, pp. 79–103.

[158] S. Negara, M. Vakilian, N. Chen, R. E. Johnson, and D. Dig. "Is It Dangerous to Use Version Control Histories to Study Source Code Evolution?" In: *European Conference on Object-Oriented Programming, ECOOP 2012: ECOOP 2012 – Object-Oriented Programming*. Springer, Berlin, Heidelberg, 2012, pp. 79–103.

[159] A. T. Nguyen, T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. "Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling." In: *2012 27th IEEE/ACM International Conference on Automated Software Engineering, ASE 2012 - Proceedings*. New York, New York, USA: ACM Press, 2012, pp. 70–79.

[160] M. Niazi, S. Mahmood, M. Alshayeb, M. R. Riaz, K. Faisal, N. Cerpa, S. U. Khan, and I. Richardson. "Challenges of project management in global software development: A client-vendor analysis." In: *Information and Software Technology* 80.C (Dec. 2016), pp. 1–19.

[161] M Oltrogge, E Derr, C Stransky, Y Acar, S Fahl, C Rossow, G Pellegrino, S Bugiel, and M Backes. "The Rise of the Citizen Developer: Assessing the Security Impact of Online App Generators." In: *2018 IEEE Symposium on Security and Privacy (SP)*. 2018, pp. 634–647.

[162] S. Pachidi, M. Spruit, and I. Van De Weerd. "Understanding users' behavior with software operation data mining." In: *Computers in Human Behavior* 30 (2014), pp. 583–594.

[163] N. W. Paper. "SOFTWARE Key Enabler for Innovation." In: July (2014).

[164] F. Peters. "On Privacy and Utility while Improving Software Quality." In: *International Conference on Current Trends in Theory and Practice of Computer Science*. Vol. 75. SOFSEM SRF, 2017, pp. –.

[165] W. Poncin, A. Serebrenik, and M. V. D. Brand. "Process Mining Software Repositories." In: *2011 15th European Conference on Software Maintenance and Reengineering* (2011), pp. 5–14.

[166] W. Poncin, A. Serebrenik, and M. Van Den Brand. "Process mining software repositories." In: *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR*. 2011, pp. 5–13.

[167] A. Porter and L. Votta. "Comparing detection methods for software requirements inspections: A replication using professional subjects." In: *Empirical software engineering* 3.4 (1998), pp. 355–379.

[168] A. A. Porter, L. G. Votta, and V. R. Basili. "Comparing detection methods for software requirements inspections: A replicated experiment." In: *IEEE Transactions on software Engineering* 21.6 (1995), pp. 563–575.

[169] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo. "Categorizing the Content of GitHub README Files." In: *Empirical Software Engineering* 24.3 (June 2019), pp. 1296–1327.

[170] B. Purnima and K. Arvind. "EBK-Means: A Clustering Technique based on Elbow Method and K-Means in WSN." In: *International Journal of Computer Applications* 105.9 (2014), pp. 17–24.

[171] F. Rahman and P. Devanbu. "How, and Why, Process Metrics Are Better." In: *Proceedings of the 2013 International Conference on Software Engineering*. San Francisco, CA, USA: IEEE Press, 2013, 432–441.

[172] M. S. Rakha, C. P. Bezemer, and A. E. Hassan. "Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval." In: *Empirical Software Engineering* 23.5 (Oct. 2018), pp. 2597–2621.

[173] M. S. Rakha, W. Shang, and A. E. Hassan. "Studying the needed effort for identifying duplicate issues." In: *Empirical Software Engineering* 21.5 (Oct. 2016), pp. 1960–1989.

[174] J. Ratzinger, T. Sigmund, P. Vorburger, and H. Gall. "Mining software evolution to predict refactoring." In: *Proceedings - 1st International Symposium on Empirical Software Engineering and Measurement, ESEM 2007*. IEEE, Sept. 2007, pp. 354–363.

[175] W. Remus. "Using students as subjects in experiments on decision support systems." In: *Proceedings of the Twenty-Second Annual Hawaii International Conference on System Sciences. Volume III: Decision Support and Knowledge Based Systems Track*. Vol. 3. IEEE Computer Society. 1989, pp. 176–177.

[176] D. Rodriguez, I. Herraiz, and R. Harrison. "On software engineering repositories and their open problems." In: *2012 1st International Workshop on Realizing AI Synergies in Software Engineering, RAISE 2012 - Proceedings*. 2012, pp. 52–56.

[177] F. Rojas Blum. *Metrics in process discovery*. Tech. rep. Chile: Computer Science Department, Universidad de Chile, 2015.

[178] A. Rozinat, A. de Medeiros, C. Günther, A. Weijters, and W. van der Aalst. "Towards an evaluation framework for process mining algorithms." In: *Beta, Research School for Operations Management and Logistics*. (2007), pp. 1–20.

[179] A. Rozinat, M Veloso, and W. M. P. van der Aalst. "Evaluating the Quality of Discovered Process Models." In: *Information Systems Journal* 16.Section 2 (2008), pp. 1–8.

[180] V. A. Rubin, I. Lomazova, and W. M. P. v. d. Aalst. "Agile development with software process mining." In: *Proceedings of the 2014 International Conference on Software and System Process - ICSSP 2014* (2014), pp. 70–74.

[181] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. van der Aalst. "Process mining can be applied to software too!" In: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14* (2014), pp. 1–8.

[182] P. Runeson. "Using students as experiment subjects–an analysis on graduate and freshmen student data." In: *Proceedings of the 7th International Conference on Empirical Assessment in Software Engineering*. Citeseer. 2003, pp. 95–102.

[183] R. Saborido, R. Morales, F. Khomh, Y. G. Guéhéneuc, and G. Antoniol. "Getting the most from map data structures in Android." In: *Empirical Software Engineering* 23.5 (Oct. 2018), pp. 2829–2864.

[184] P. Salza, F. Palomba, D. D. Nucci, C. D'uva, A. De Lucia, and F. Ferrucci. "Do Developers Update Third-Party Libraries in Mobile Apps." In: *Proceedings of the 26th Conference on Program Comprehension*. Vol. 12. Association for Computing Machinery, 2018, 255–265.

[185] A. L. Santos, G. Prendi, H. Sousa, and R. Ribeiro. "Stepwise API usage assistance using n-gram language models." In: *Journal of Systems and Software* 131 (Sept. 2017), pp. 461–474.

[186] A. A. Sawant, R. Robbes, and A. Bacchelli. "To react, or not to react: Patterns of reaction to API deprecation." In: *Empirical Software Engineering* 24.6 (2019), pp. 3824–3870.

[187] S. Shappell, C. Detwiler, K. Holcomb, C. Hackworth, A. Boquet, and D. A. Wiegmann. "Human error and commercial aviation accidents: An analysis using the human factors analysis and classification system." In: *Human Factors* 49.2 (Apr. 2007), pp. 227–242.

[188] F. K. Shuptrine. "On the validity of using students as subjects in consumer behavior investigations." In: *The Journal of Business* 48.3 (1975), pp. 383–390.

[189] A. Sivaprasad and S. Jangale. "A complete study on tools & techniques for digital forensic analysis." In: *2012 International Conference on Computing, Electronics and Electrical Technologies, ICCEET 2012*. 2012, pp. 881–886.

[190] K. Z. Sultana, B. J. Williams, and T. Bhowmik. "A study examining relationships between micro patterns and security vulnerabilities." In: *Software Quality Journal* 27.1 (Mar. 2019), pp. 5–41.

[191] M. Svahnberg, A. Aurum, and C. Wohlin. "Using students as subjects-an empirical evaluation." In: *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. 2008, pp. 288–290.

[192] S. E. S. Taba, I. Keivanloo, Y. Zou, and S. Wang. "An exploratory study on the usage of common interface elements in android applications." In: *Journal of Systems and Software* 131 (Sept. 2017), pp. 491–504.

[193] G. Taguchi and Asian Productivity Organization. *Introduction to quality engineering : designing quality into products and processes*. The Organization, 1986, p. 191.

[194] D. Taibi, V. Lenarduzzi, C. Pahl, and A. Janes. "Microservices in Agile Software Development: A Workshop-Based Study into Issues, Advantages, and Disadvantages." In: *Proceedings of the XP2017 Scientific Workshops*. XP '17. New York, NY, USA: Association for Computing Machinery, 2017.

[195] M. Tanner and U. Von Willingh. "Factors leading to the success and failure of agile projects implemented in traditionally waterfall environments." In: *Management, Knowledge and Learning*. University of Cape Town, South Africa. Cape Town, 2014, pp. 693–700.

[196] C. Tantithamthavorn and A. E. Hassan. "An experience report on defect modelling in practice: Pitfalls and challenges." In: *Proceedings - International Conference on Software Engineering*. IEEE Computer Society, May 2018, pp. 286–295.

[197] D. Tapscott and A. Tapscott. *Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World*. Portfolio, 2016.

[198] F. Taymouri, M. La Rosa, and J. Carmona. "Business Process Variant Analysis Based on Mutual Fingerprints of Event Logs." In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 12127 LNCS. Springer, 2020, pp. 299–318.

[199] P. Thongtanunam, W. Shang, and A. E. Hassan. "Will this clone be short-lived? Towards a better understanding of the characteristics of short-lived clones." In: *Empirical Software Engineering* 24.2 (Apr. 2019), pp. 937–972.

[200] C. Thornton, H. H. Hoos, and K. Leyton-Brown. "Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA." In: *Journal of Machine Learning Research* 1 (2017), p. 429.

[201] Y. Tian, M. Nagappan, D. Lo, and A. E. Hassan. "What are the characteristics of high-rated apps? A case study on free Android Applications." In: *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings*. Institute of Electrical and Electronics Engineers Inc., Nov. 2015, pp. 301–310.

[202] L. W. Tim Menzies and T. Zimmermann, eds. *Perspectives on Data Science for Software Engineering*. Elsevier, 2016.

[203] F. Tu, J. Zhu, Q. Zheng, and M. Zhou. "Be careful of when: An empirical study on time-related misuse of issue tracking data." In: *ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Association for Computing Machinery, Inc, Oct. 2018, pp. 307–318.

[204] M. Vakilian, N. Chen, S. Negara, B. Ambresh, R. Roshanak, Z. Moghaddam, and R. E. Johnson. "The Need for Richer Refactoring Usage Data." In: *Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools*. Association for Computing Machinery, 2011, pp. 31–38.

[205] M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, B. P. Bailey, and R. E. Johnson. "Use, disuse, and misuse of automated refactorings." In: *Proceedings - International Conference on Software Engineering*. 2012, pp. 233–243.

[206] W. Van Der Aalst. *Process Mining: Data Science in Action*. 2nd ed. Springer-Verlag Berlin Heidelberg, 2016.

[207] W. Van Der Aalst, T. Weijters, and L. Maruster. "Workflow mining: Discovering process models from event logs." In: *IEEE Transactions on Knowledge and Data Engineering* 16.9 (Sept. 2004), pp. 1128–1142.

[208] W. Van Der Aalst et al. "Process mining manifesto." In: *Lecture Notes in Business Information Processing* 99 LNBIP (2012), pp. 169–194.

[209] I. Vanderfeesten, J. Cardoso, J. Mendling, H. A. Reijers, and W. Van Der Aalst. *Quality Metrics for Business Process Models*. Tech. rep. Technische Universiteit Eindhoven, 2007.

[210] A. H. Watson, T. J. Mccabe, and D. R. Wallace. *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Tech. rep. NIST, 1996.

[211] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers. "Refactoring large process model repositories." In: *Computers in Industry* 62.5 (2011), pp. 467–486.

[212] C. Wohlin. "Guidelines for snowballing in systematic literature studies and a replication in software engineering." In: *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering - EASE '14*. 2014, pp. 1–10.

[213] R. Wu, M. Wen, S. C. Cheung, and H. Zhang. "ChangeLocator: locate crash-inducing changes based on crash reports." In: *Empirical Software Engineering* 23.5 (Oct. 2018), pp. 2866–2900.

[214] W. Wu, F. Khomh, B. Adams, Y. G. Guéhéneuc, and G. Antoniol. "An exploratory study of api changes and usages based on apache and eclipse ecosystems." In: *Empirical Software Engineering* 21.6 (Dec. 2016), pp. 2366–2412.

[215] T. Xia, R. Krishna, J. Chen, G. Mathew, X. Shen, and T. Menzies. *Hyperparameter Optimization for Effort Estimation*. Tech. rep. North Carolina State University, Apr. 2018.

[216] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li. "Measuring Program Comprehension: A Large-Scale Field Study with Professionals." In: *IEEE Transactions on Software Engineering* 44.10 (Oct. 2018), pp. 951–976.

[217] M. Yan, X. Xia, D. Lo, A. E. Hassan, and S. Li. "Characterizing and identifying reverted commits." In: *Empirical Software Engineering* 24.4 (2019), 2171–2208.

[218] X.-L. Yang, D. Lo, X. Xia, Z.-Y. Wan, and J.-L. Sun. "What Security Questions Do Developers Ask? A Large-Scale Study of Stack Overflow Posts." In: *Journal of Computer Science and Technology* 31.5 (2016), 910–924.

[219] D. Ye, Z. Xing, and N. Kapre. "The structure and dynamics of knowledge network in domain-specific Q&A sites: a case study of stack overflow." In: *Empirical Software Engineering* 22.1 (Feb. 2017), pp. 375–406.

[220] Y. S. Yoon and B. A. Myers. "An exploratory study of backtracking strategies used by developers." In: *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE)*. 2012, pp. 138–144.

[221] Y. S. Yoon and B. A. Myers. "A longitudinal study of programmers' backtracking." In: *2014 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 2014, pp. 101–108.

[222] Y. Yoon and B. A. Myers. "Capturing and Analyzing Low-Level Events from the Code Editor." In: *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools*. PLATEAU '11. New York, NY, USA: Association for Computing Machinery, 2011, pp. 25–30.

[223] C. Zannier, G. Melnik, and F. Maurer. "On the success of empirical studies in the international conference on software engineering." In: *Proceedings - International Conference on Software Engineering*. 2006, pp. 341–350.

[224] D. Zhang, S. Han, Y. Dang, J.-G. Lou, H. Zhang, M. Research Asia, and T. Xie. "Software Analytics in Practice." In: *IEEE Software* (2013), pp. 30–37.

[225] L. Zhang, J. H. Tian, J. Jiang, Y. J. Liu, M. Y. Pu, and T. Yue. "Empirical Research in Software Engineering — A Literature Survey." In: *Journal of Computer Science and Technology* 33.5 (Sept. 2018), pp. 876–899.

[226] T. Zimmermann. "Software productivity decoded: How data science helps to achieve more (keynote)." In: *ACM International Conference Proceeding Series*. Vol. Part F128767. Association for Computing Machinery, July 2017, pp. 1–2.

[This page has been intentionally left blank]

Appendix A.

Dissertation Workbench

Contents

This Appendix presents the tools used to perform current research and in writing this dissertation.

A.1 Companion Tools

Table A.1: Software Tools used during this Dissertation

Software	Usage/Purpose	URL
Software Development		
Eclipse	SDPM Plugin for Eclipse	https://www.eclipse.org
IntelliJ IDEA	SDPM Plugin for PyCharm	https://www.jetbrains.com/idea
Sublime Text	Engine for Event Formatting & Aggregation	https://www.sublimetext.com
Event Temporary Persistence Layer		
CSV Files	Store Events Locally at Runtime	
Apache Kafka	Store Events Locally or Remotely at Runtime	https://kafka.apache.org
Azure Event Hub	Store Events Remotely at Runtime	https://azure.microsoft.com
Trello Rest API	Load Software Project Tasks at Runtime	https://www.trello.com
Event Repository, Transformation and Exporting		
MySQL	Store, Transform and Query Events	https://www.mysql.com
Process Data Mining		
ProM	Process Discovery & Conformance Checking	http://www.promtools.org
StateChart Workbench	Process Complexity Metrics Extraction	
PM4Py	Process Mining for Python	https://pm4py.fit.fraunhofer.de
Disco	Process Discovery and Analysis	https://www.fluxicon.com
Product Data Mining		
Metrics	Software Metrics Extraction	http://metrics.sourceforge.net
Data Science Workbench		
R	Data Analysis and Charts Generation/Export	https://www.rproject.org
R Studio	Graphical User Interface for R	https://www.rstudio.com
Weka	Model Training and Selection	https://www.cs.waikato.ac.nz
Auto-Weka	Model Hyperparameter Optimization	
Software Development Process Mining Personal Dashboard		
Docker	Containers Engine	https://www.docker.com
Docker Desktop	Graphical User Interface for Docker	
Dissertation Writing		
Draw.io	Diagram Creation and Management	https://www.draw.io
Mendeley	Reference Management	https://www.mendeley.com
Overleaf	Online Document Writing	https://www.overleaf.com

A.2 Software Development Process Mining Plugin for Eclipse

A.2.1 Installation Manual

- Use Eclipse 4.6.x or later (only tested in these platforms)
- Use Java 1.8.x

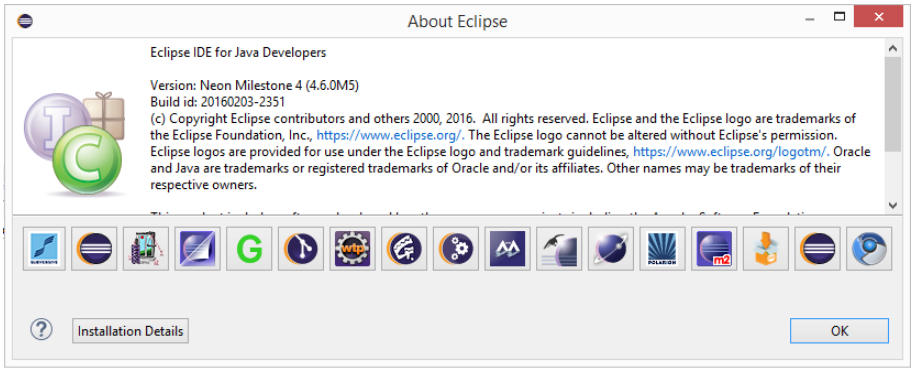

Figure A.1: Validating Eclipse Requirements

- Install the plugin from a Remote Repository
  - Select Help in the Main Menu -> Install New Software
  - Add a name and URL for the new Repository –> Press button Add

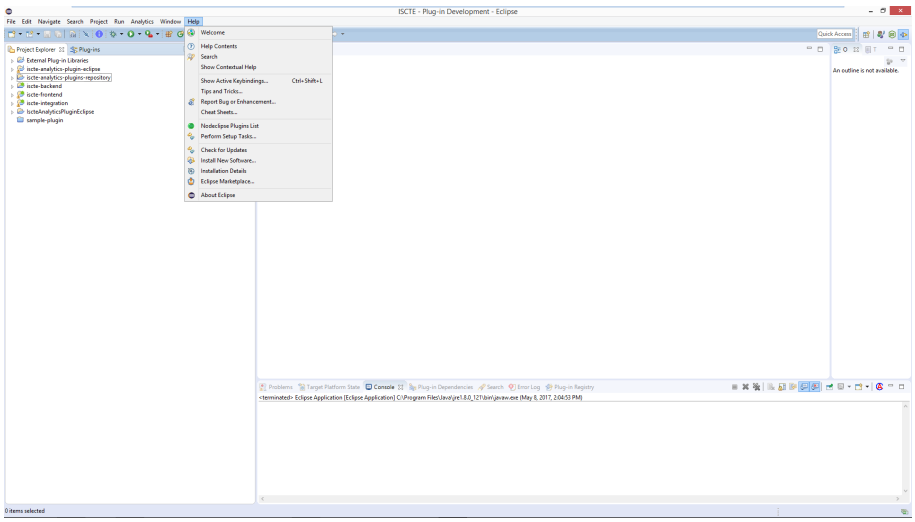

Figure A.2: Install New Software

- Add a Name and URL to the Repository:
  - Type a Name for the Repository (its up to you this name).
  - https://github.com/jcaldeir/iscte-analytics-plugins-repository/raw/master
  - Press Ok

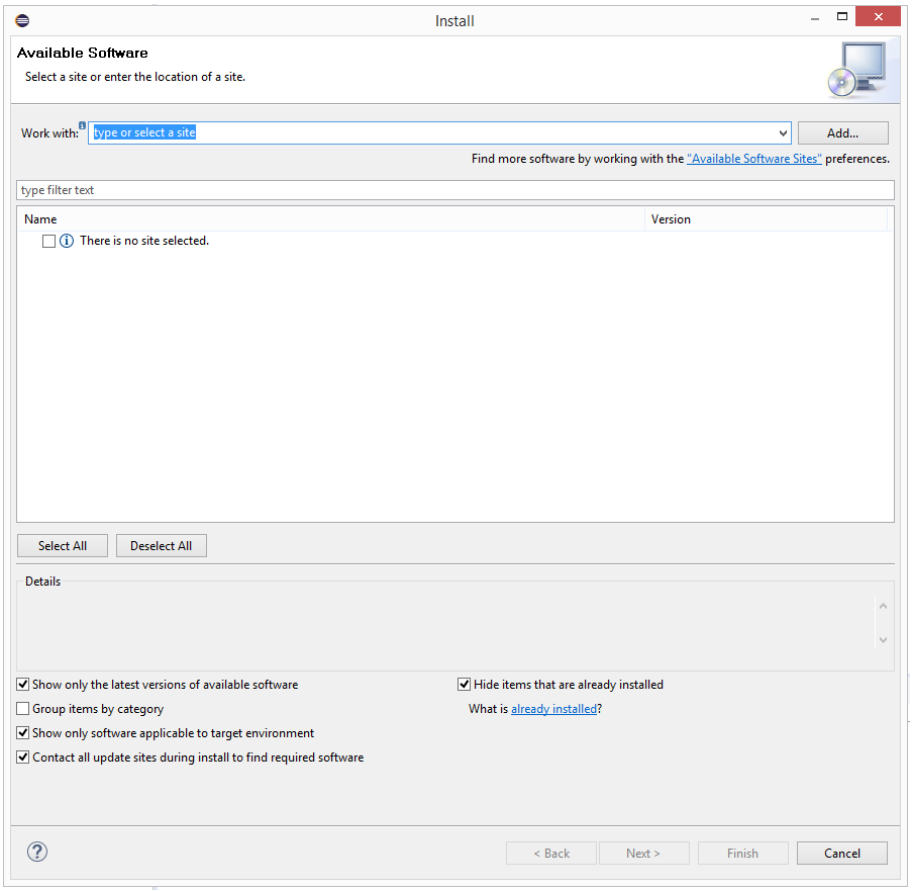

Figure A.3: Add New Repository

- You should see something similar to this screenshot A.4

  - Select Software Development Process Mining Plugin for Eclipse entry

  - If you don't see the plugin listed, unselect the option : Group items by category

  - Press Next

- Select the latest version for the Software Development Process Mining Plugin for Eclipse

  - Accept the License Agreement and press Finish

  - The plugin should start to install. On the security Warning, press OK

  - Restart Eclipse

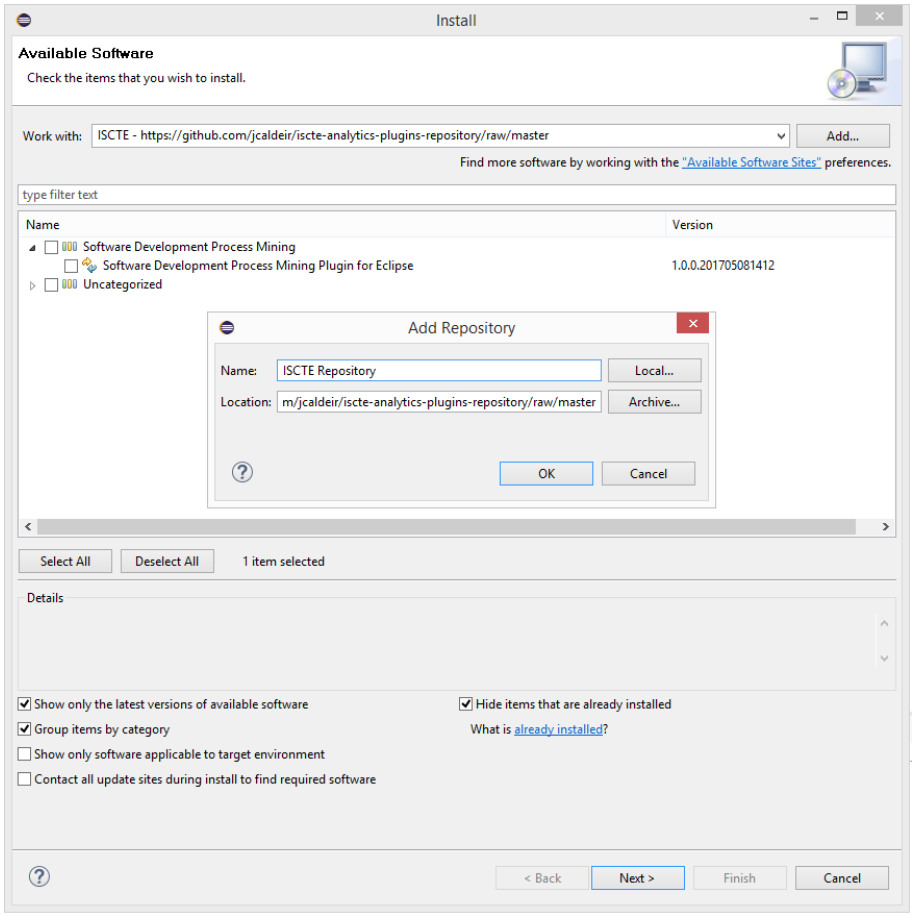

Figure A.4: Add Plugin remote repository

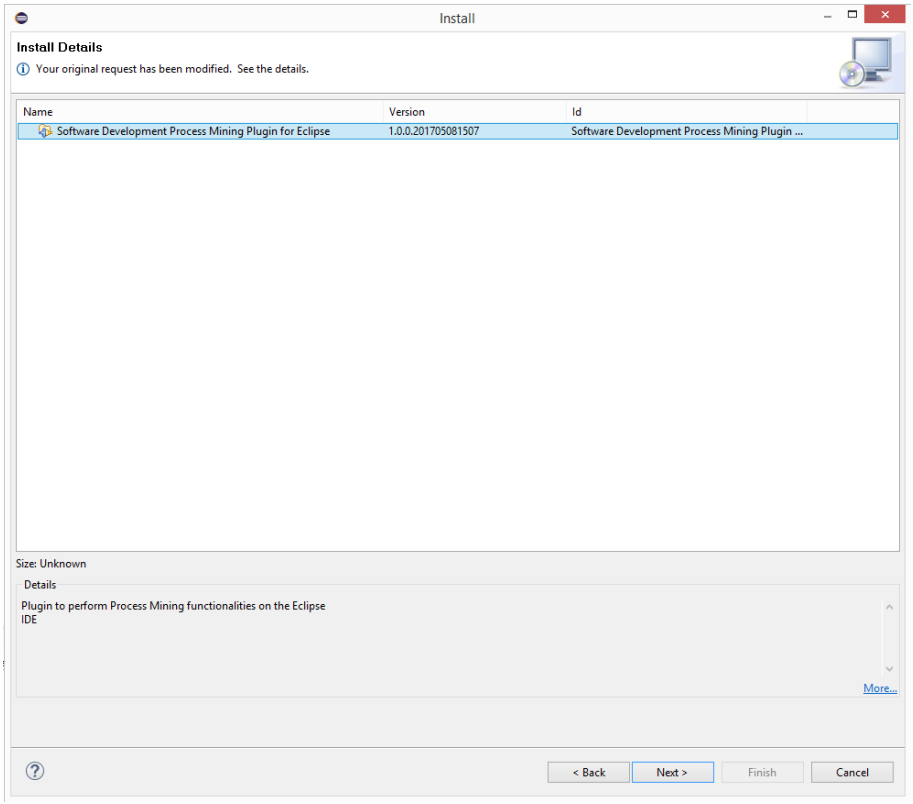

Figure A.6: Confirm version to Install

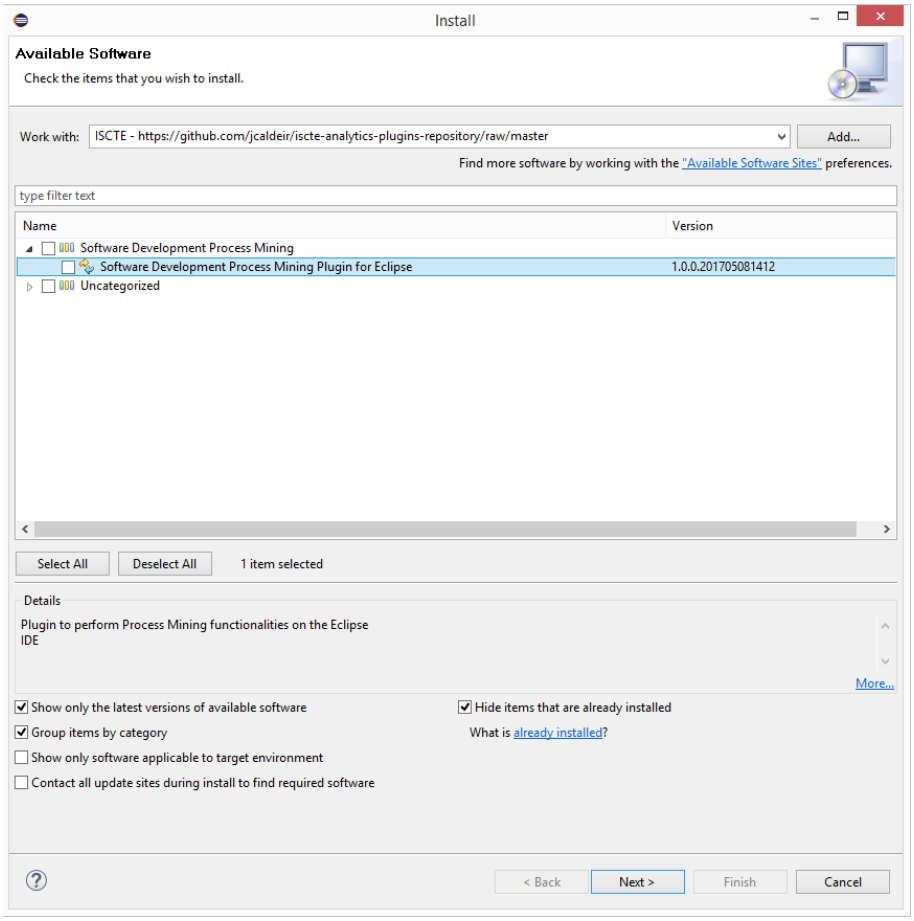

Figure A.5: Select plugin version to Install

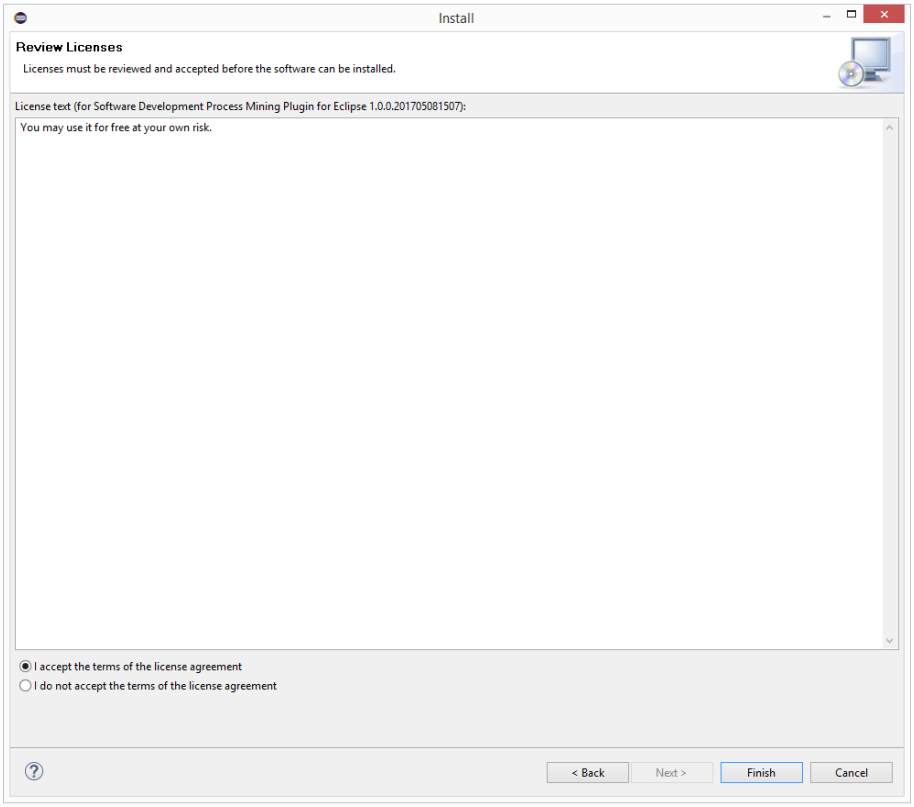

Figure A.7: Accept the License Agreement

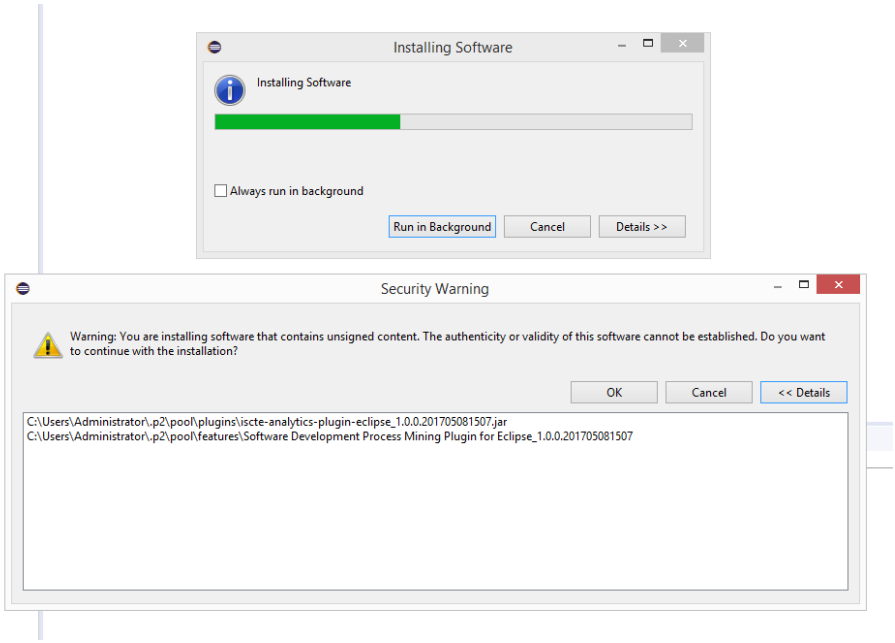

Figure A.8: Execute the Installation

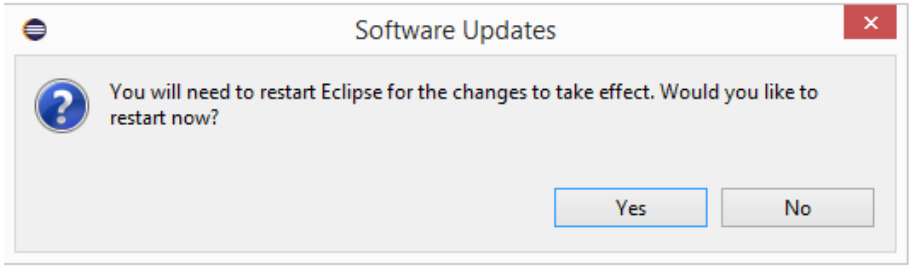

Figure A.9: Restart Eclipse

A.2.2 User Guide

- Enter Credentials

  - (Do it only once per workspace. Multiple workspaces require to enter the credentials again). If another user uses your Eclipse, it must change credentials every time he/she is working with it

  - From the Process Mining Menu –> My Credentials

  - Enter your account Username provided after the registration in the SDPM web site[1]

  - Enter the Key provided by the SDPM team after registration

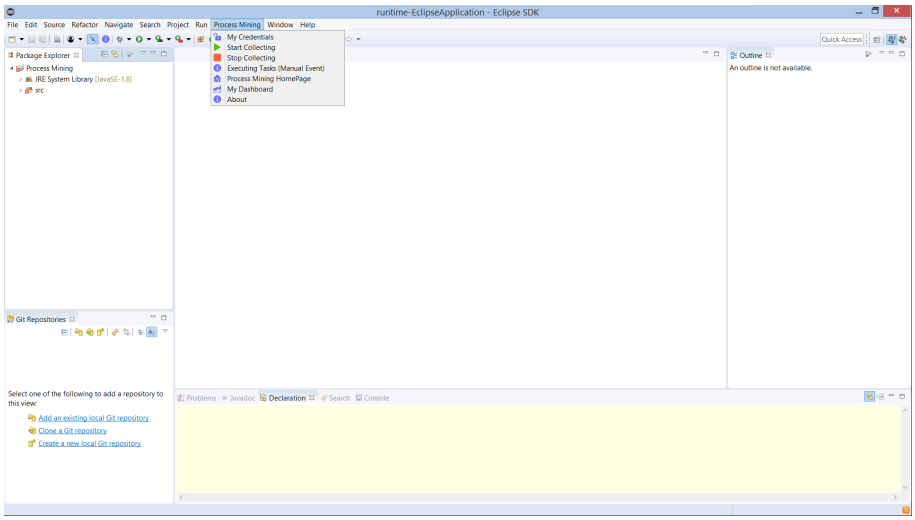

Figure A.10: Process Mining Menu

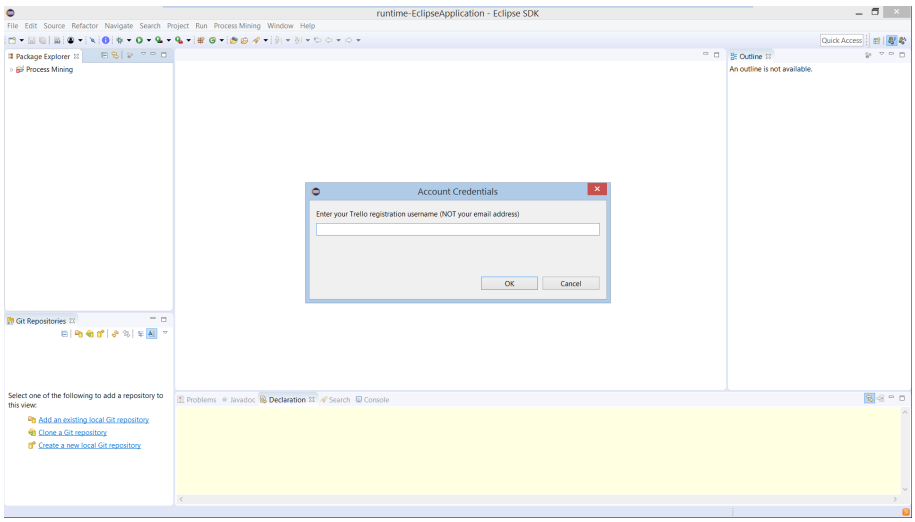

Figure A.11: Enter Username

[1] http://www.software-development-process-mining.com/

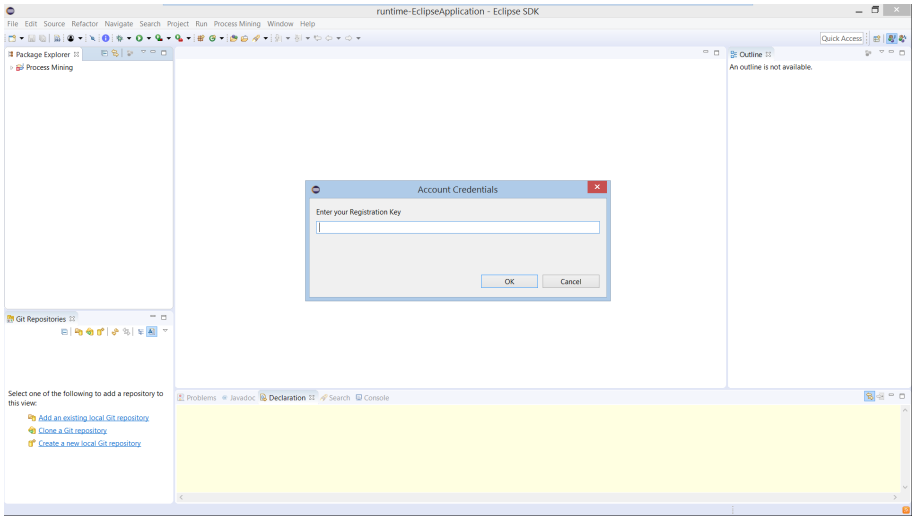

Figure A.12: Enter Key

A.2.2.1 Menu Actions

- Start Collection

  - It starts the data/event collection. Collection is started by default once you start Eclipse, but if you ever hit Stop Collection during a work session, you have to hit Start Collection again to resume data/event collection

  - If you close Eclipse and open it again, data collection is started automatically

  - You may want to check if its collecting by trying to start the collection or to check the plugin About menu

  - NOTE: Never forget to check if the collection is started. If it's not started, please start it

- Stop Collection

  - It stops the data/event collection. Collection is started by default once you start Eclipse, but if you ever hit Stop Collection during a work session, you have to hit Start Collection again to resume data/event collection. Until then the plugin is not collecting any data

  - You may want to check if its collecting by trying to stop the collection or to check the plugin About menu.

  - You may want to check if its collecting by trying to start the collection or to check the plugin About menu.

  - NOTE: Never forget to check if the collection is started. If it's not started, please start it.

- Executing Tasks

  - This option will fetch your Trello Activities and lists them so that you can choose to declare/annotate what task you are doing

- Every time you start to do any task in your project, you should choose it from the list before doing the work you are going to do (coding) and Press OK.

- In addition to your Trello activities, you can find pre-defined tasks/events

- You also find a task called “Other”, which lets you input manually as in free text what task you are doing

- Select one task at a time from the tasks list. If the list is not listing your own group Trello tasks, please contact the teachers.

- You may repeat tasks.

- Process Mining Homepage

  - It points you to the www.software-development-process-mining.com page

- My Dashboard

  - It links to your personal dashboard to view your own data collected by the plugin. It will be active soon

- About

  - Some info about the plugin, if its collecting or not, and under which user context the plugin is collecting the data/events You should check this option to confirm if the plugin is collecting or not and to check the user credentials.

  - Example : In this example the plugin is collecting data (true) and activated to collect data for user pedropascoal5

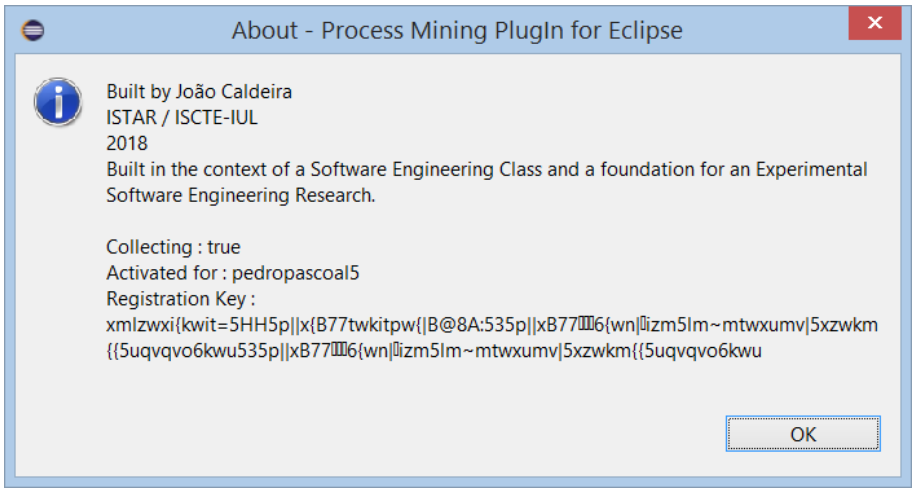

Figure A.13: About the Plugin

The below screenshot is an example from an activity list from a randomly chosen group

A.2.3 Integrations

- The plugin stores the preferences and the collection files (csv and json files) in the users' workspaces folder

  - Ex: `C:\Iscte\.metadata\.plugins\iscte.analytics.plugin.eclipse`

- Listing A.1 shows the credentials file

- Listing A.2 shows the integrations configuration file

  - Three integrations are possible: Trello, Azure Event Hub and Kafka

Listing A.1: Plugin Credentials Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<section name="Credentials">

    <item key="key" value="538e9f328c15f11e81072f6f53344aba0217d650efad4"/>
    <item key="username" value="joaocaldeira"/>

</section>
```

Listing A.2: Plugin Integrations Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<section name="Integrations">

    <item key="trelloActive" value="true"/>
    <item key="trelloKey" value="11549b1bb90dc2287b80e6c527adc133"/>
    <item key="trelloToken" value="d43e4022b8ef156d767cc931d493d3873d"/>
    <item key="trelloPrefixCode" value="SID"/>

    <item key="azureActive" value="true"/>
    <item key="namespaceName" value="iscte-iul-eclipse-plugin"/>
    <item key="eventHubName" value="process-mining"/>
    <item key="sasKeyName" value="eclipse-plugin-access-key"/>
    <item key="sasKey" value="NJ99K/0pQctu2b5ZDAdVKRcc=RzHpzn2CzkTioNt6hTm"/>

    <item key="kafkaActive" value="true"/>
    <item key="kafkaServers" value="localhost:9092"/>
    <item key="kafkaTopic" value="sdpm"/>

</section>
```

A.2.3.1 Trello Integration

- Using the Trello integration, a user can list and choose the activities he/she is executing in the project under development
- This integration requires the configuration in file A.2 of some Trello properties
  - If the integration should be active or not: *trelloActive*
  - The Trello API Key: *trelloKey*
  - The Trello API Token: *trelloToken*
  - Bring only project activities filtered by Projects prefix code: *trelloPrefixCode*

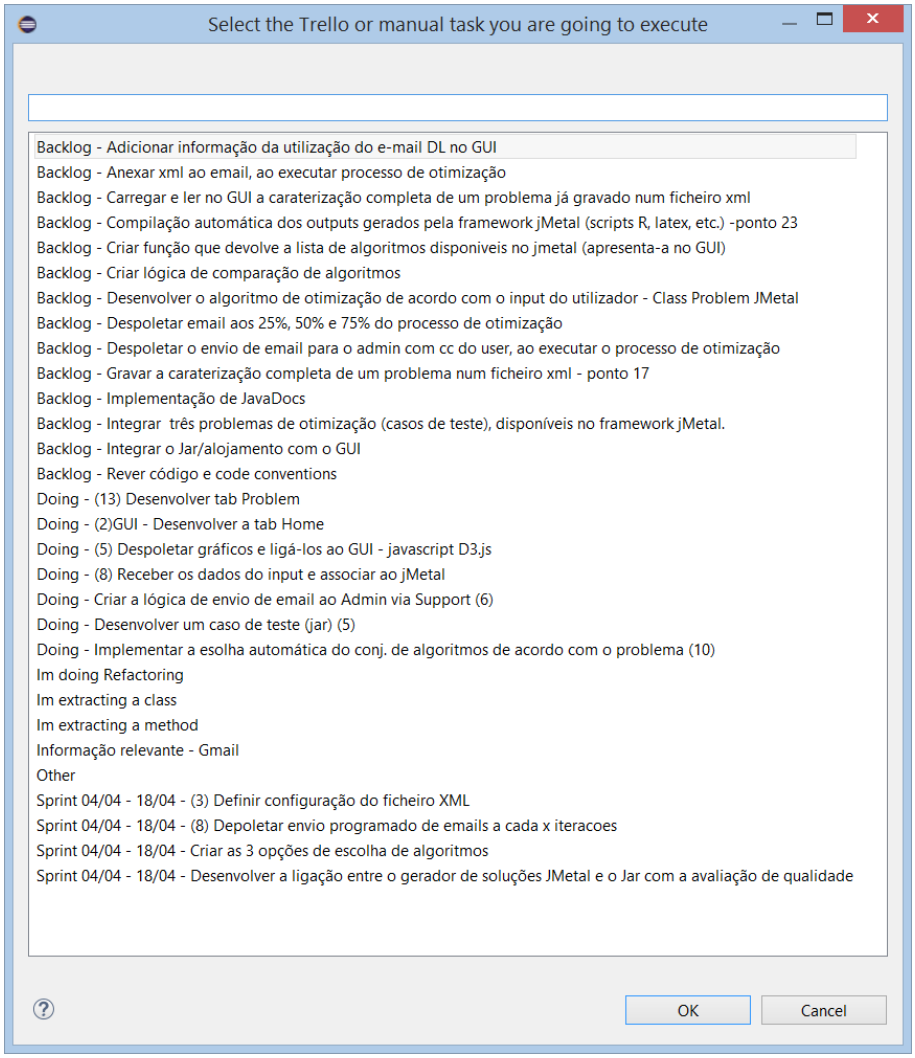

Figure A.14: Enter Key

A.2.3.2 Azure Integration

- Using the Azure integration, a user can store the events generated by the SDPM plugin running within Eclipse, in a Azure repository.

- This integration requires the configuration in file A.2 of a few Azure properties

  - If the integration should be active or not: *azureActive*

  - The Azure namespace name: *namespaceName*

  - The Event Hub Name: *eventHubName*

  - The Azure API Access Key Name: *sasKeyName*

  - The Azure API Access Key: *sasKey*

A.2.3.3 Azure Event Hub Setup

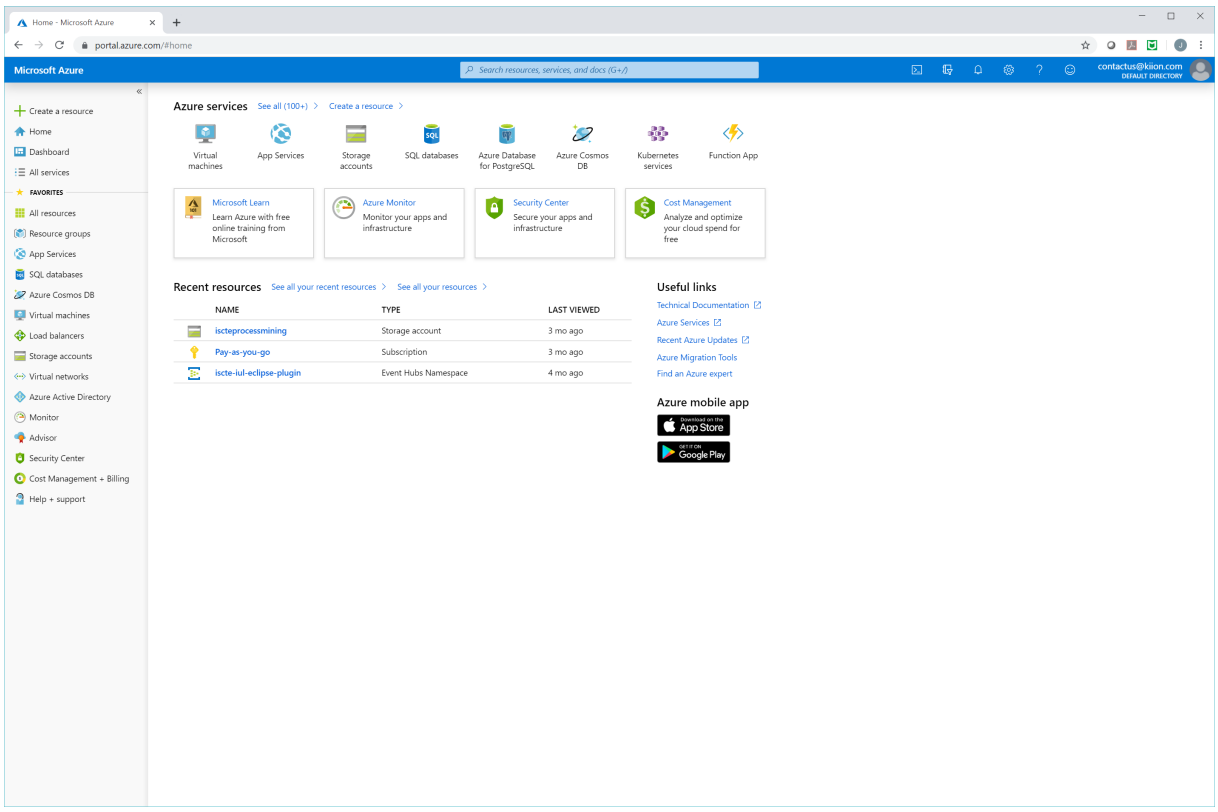

Figure A.15: Azure Services Setup

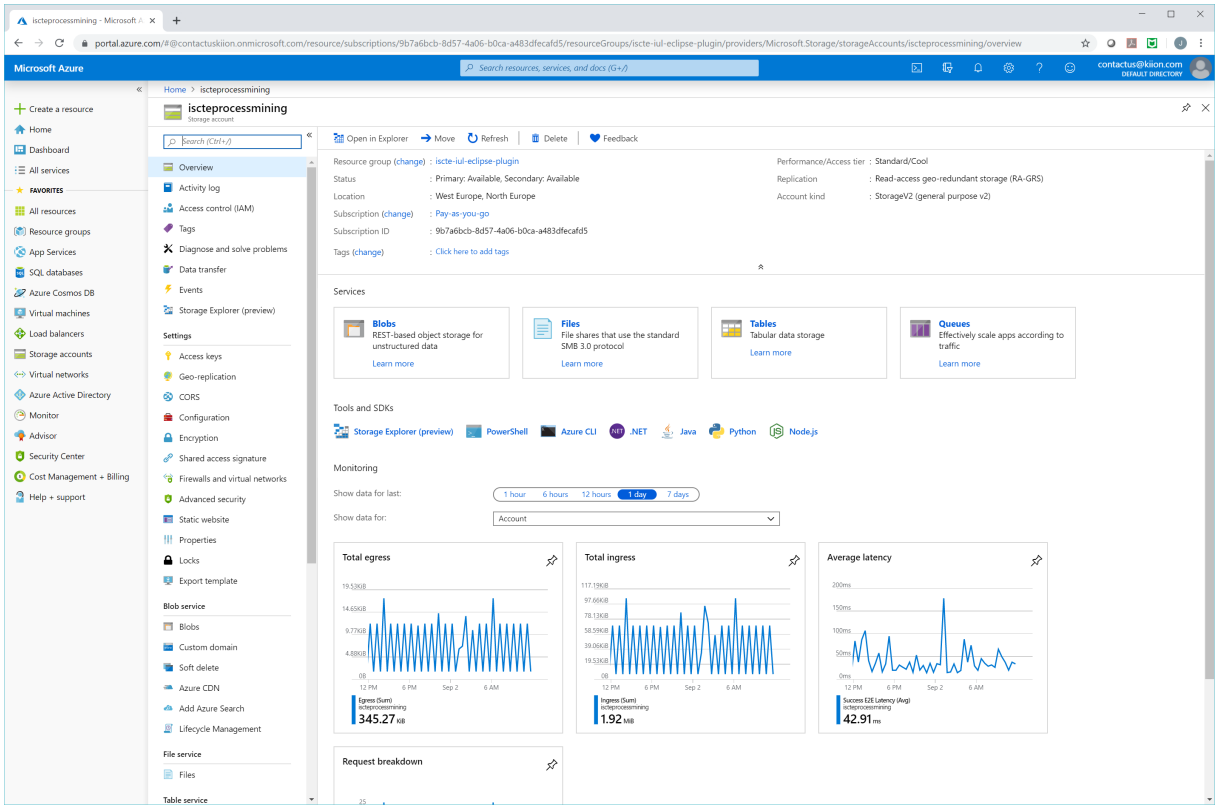

Figure A.16: Configuring a Storage Account - Create Account

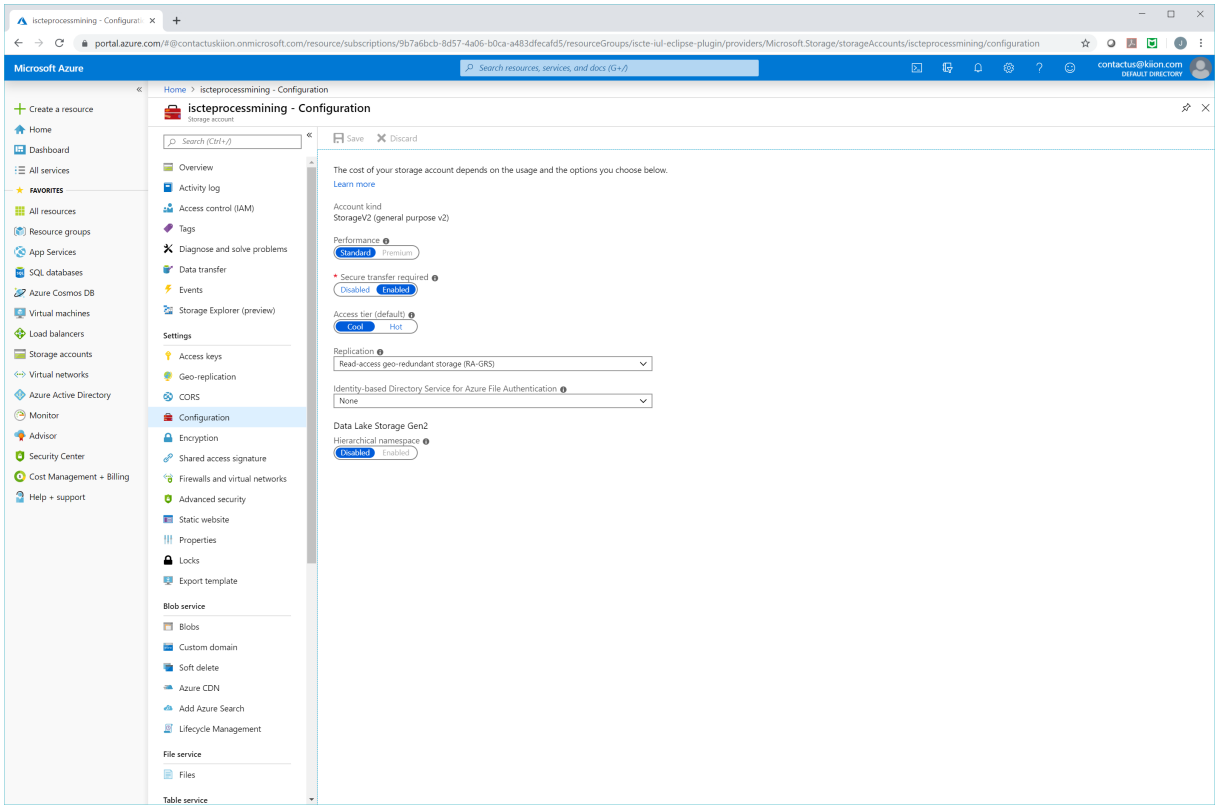

Figure A.17: Configuring a Storage Account

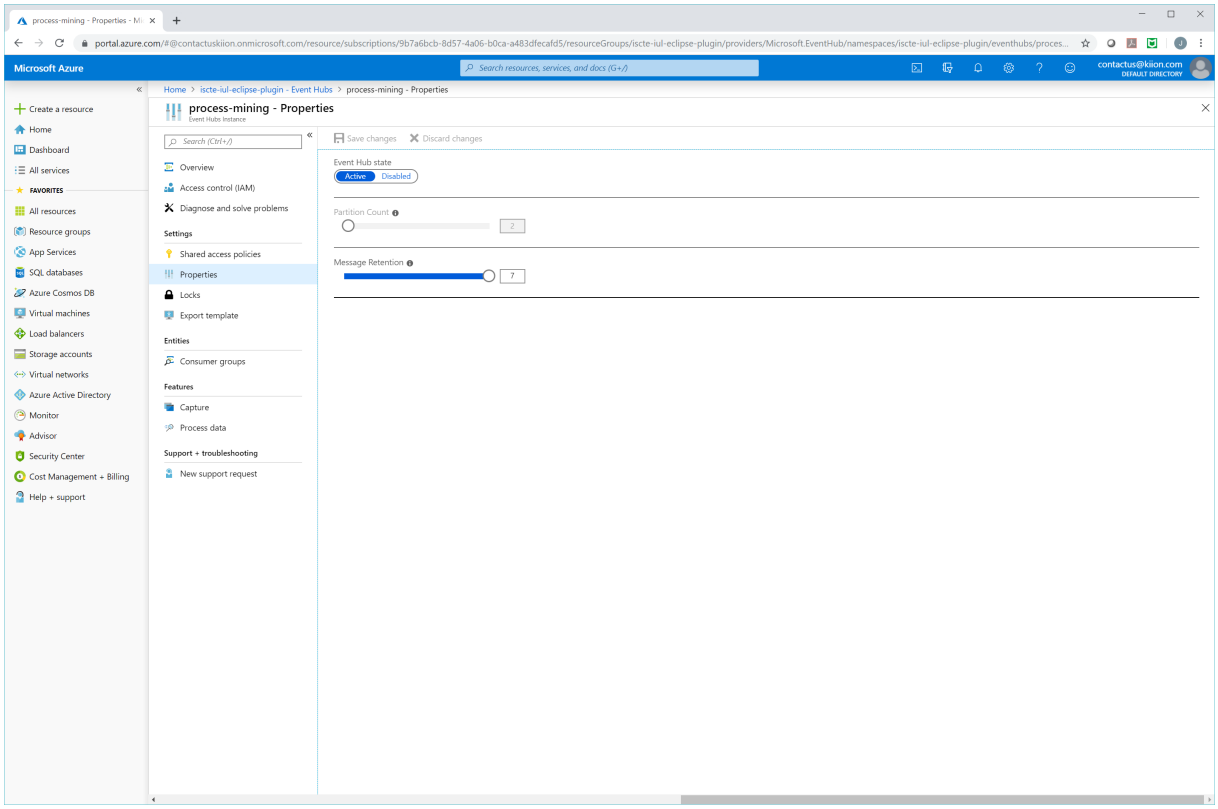

Figure A.18: Configure an Event Hub Instance - Instance State and Event Retention Period

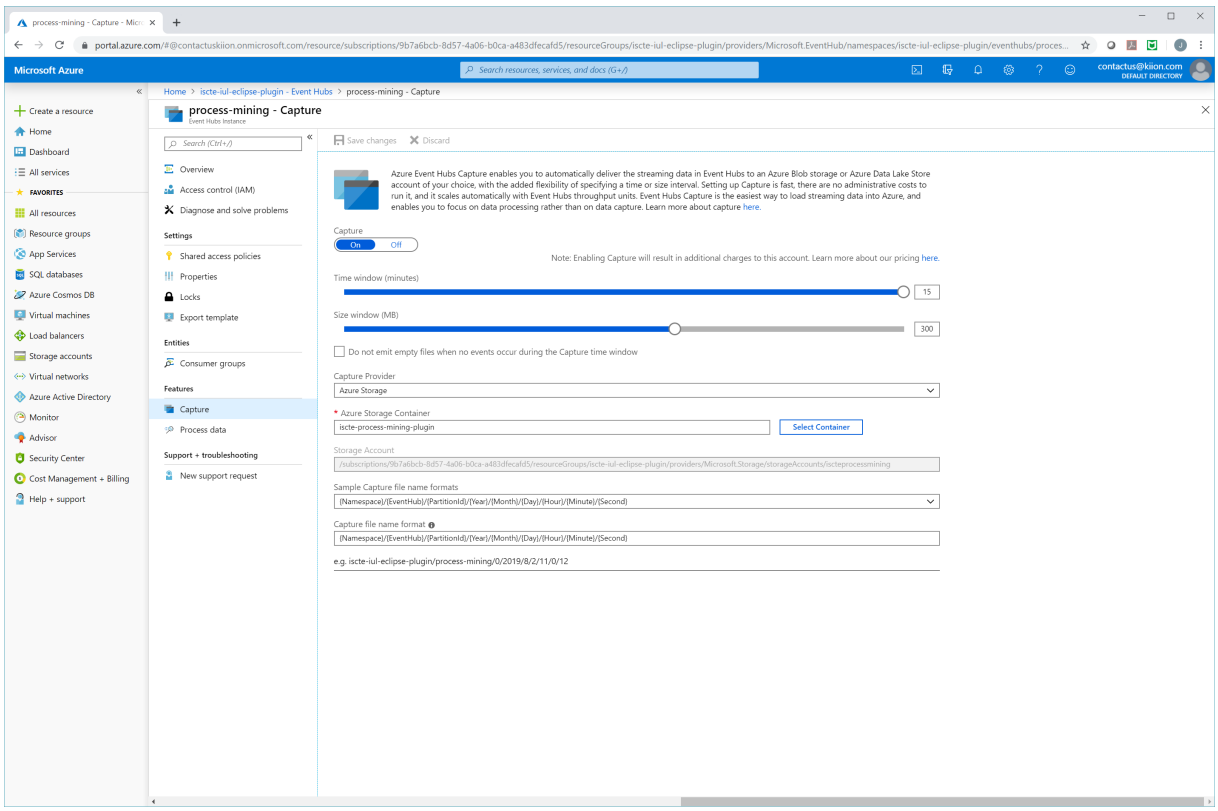

Figure A.19: Configure an Event Hub Instance - Capture Data Policy

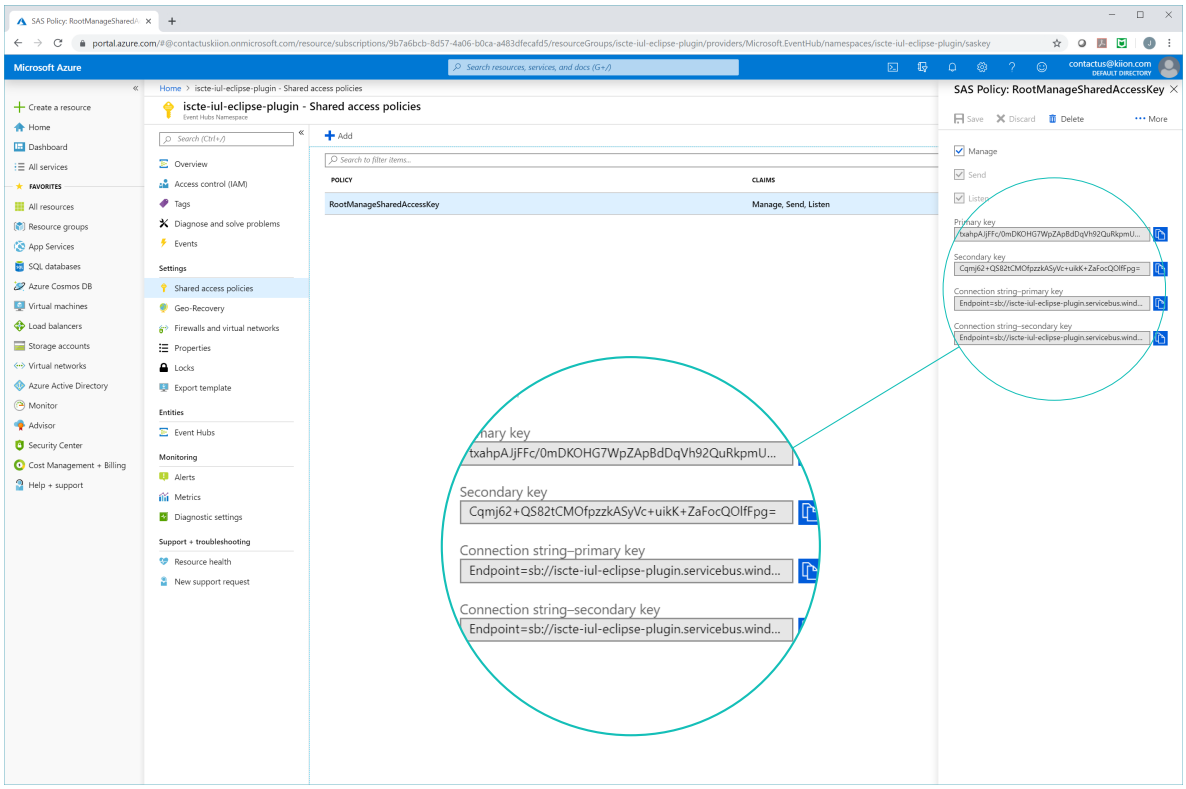

Figure A.20: Azure API Namespace Integration Key

A.2.3.4 Kafka Integration

- Using the Kafka integration, a user can store, locally or remotely, the events generated by the SDPM plugin running within Eclipse, in a Kafka repository.
- This integration requires the configuration in file A.2 of a few Kafka properties
  - If the integration should be active or not: *kafkaActive*
  - The Kafka cluster servers list: *kafkaServers*
  - The Kafka topic on where to store the events: *kafkaTopic*

A.3 Software Development Process Mining Plugin for PyCharm

A.3.1 Installation Manual for PyCharm

- Use PyCharm 2020.2 or later (only tested in these platforms)
- Use Java 1.8.x

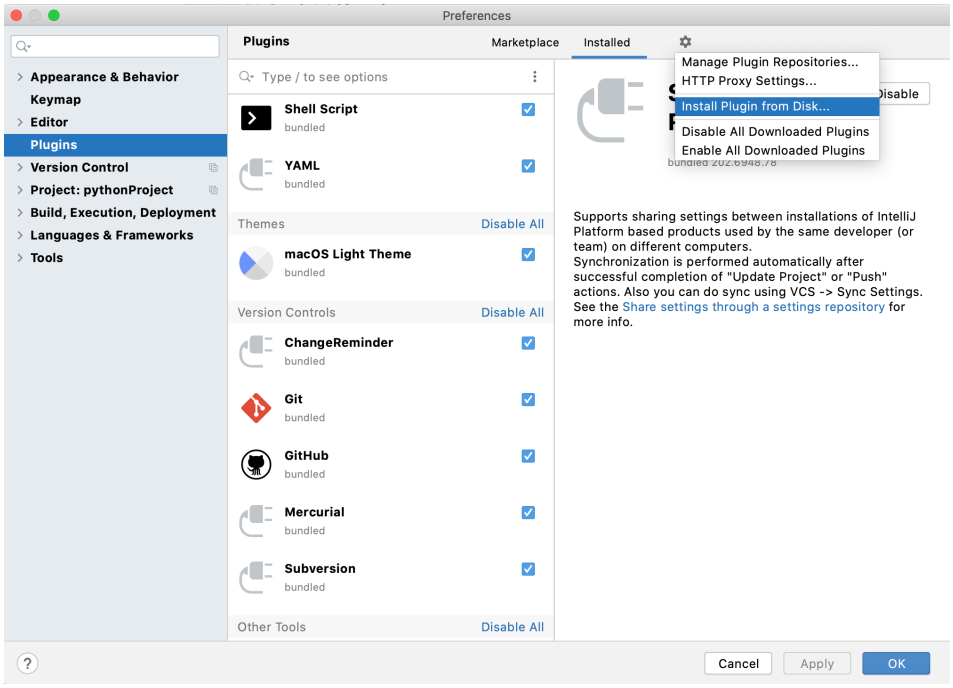

Figure A.21: Installing Software Development Process Mining Plugin for PyCharm from Disk

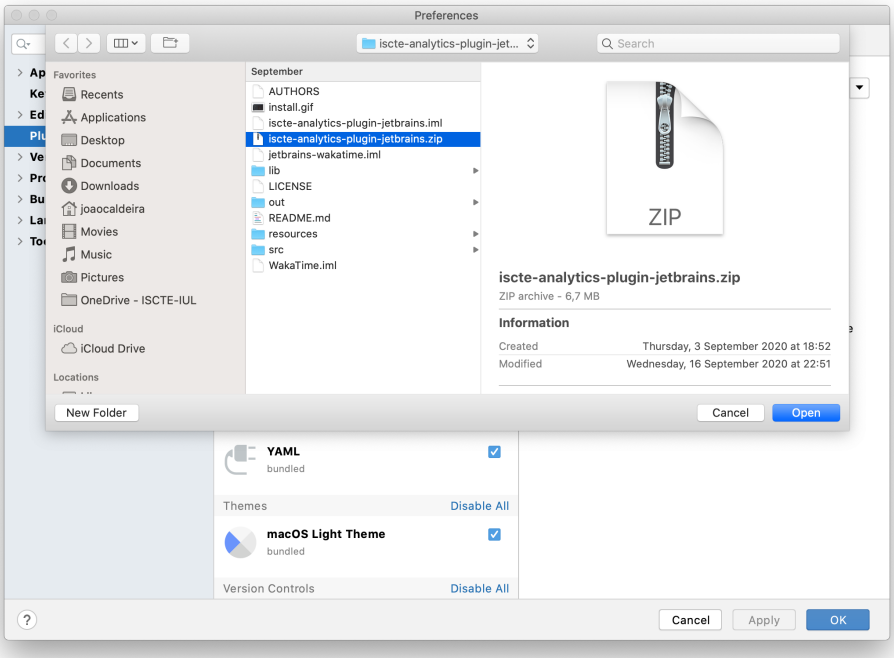

Figure A.22: Installing Software Development Process Mining Plugin for PyCharm from Disk

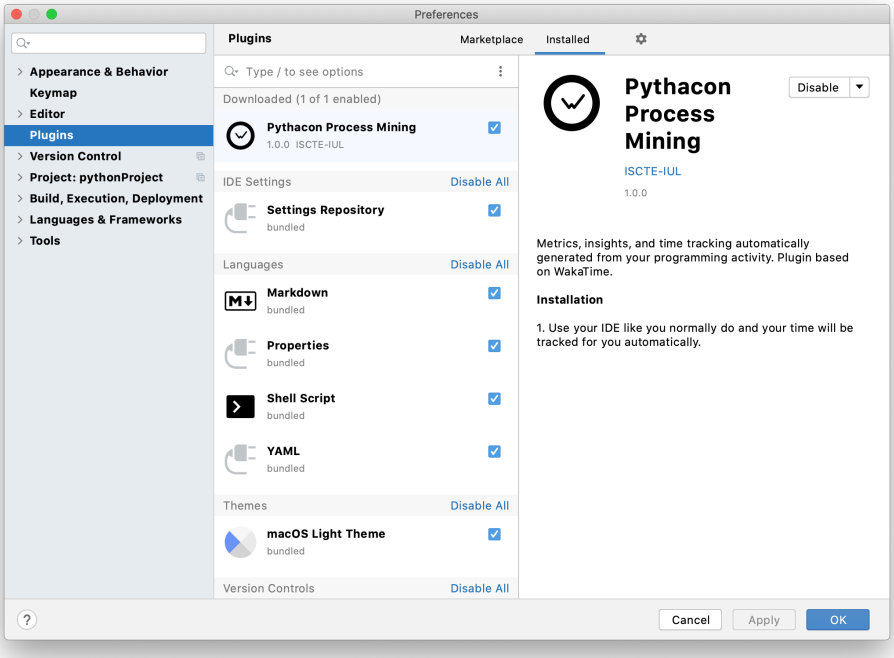

Figure A.23: Plugin Loaded - A special bundle for the Pythacon Contest

A.4 Software Development Process Mining Dashboard

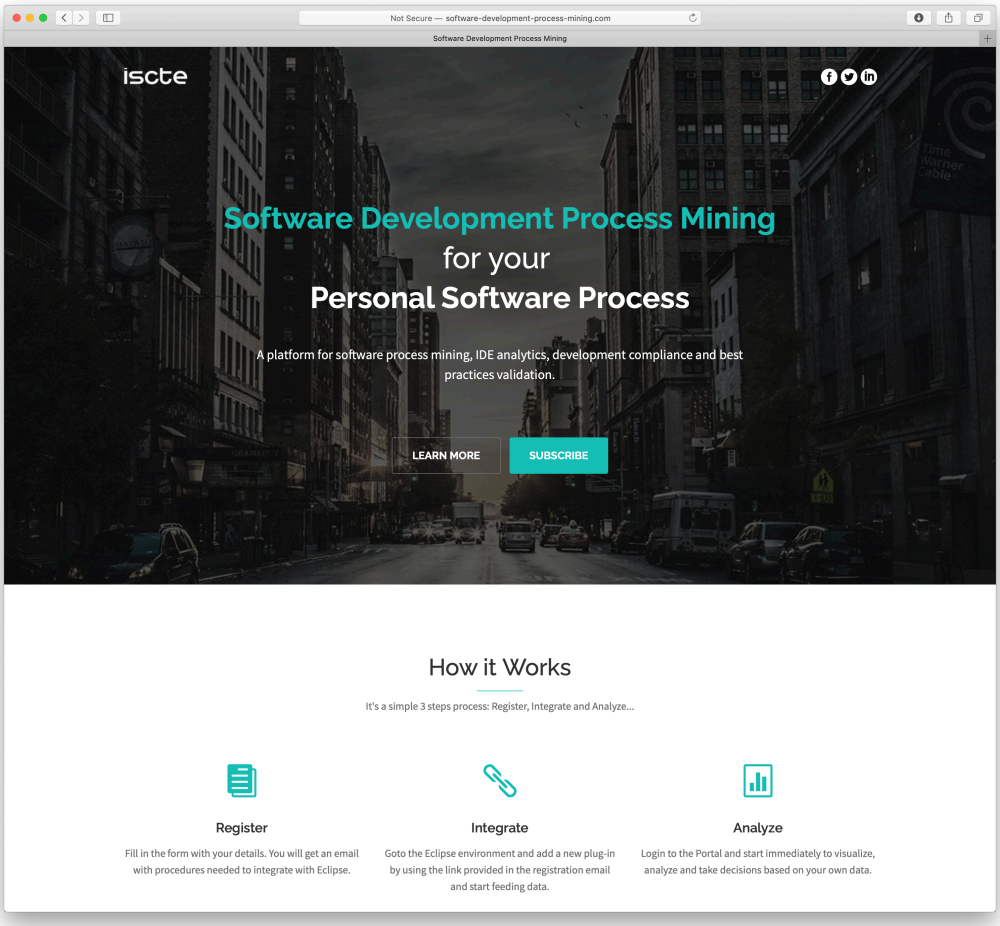

Figure A.24: SDPM Web Site

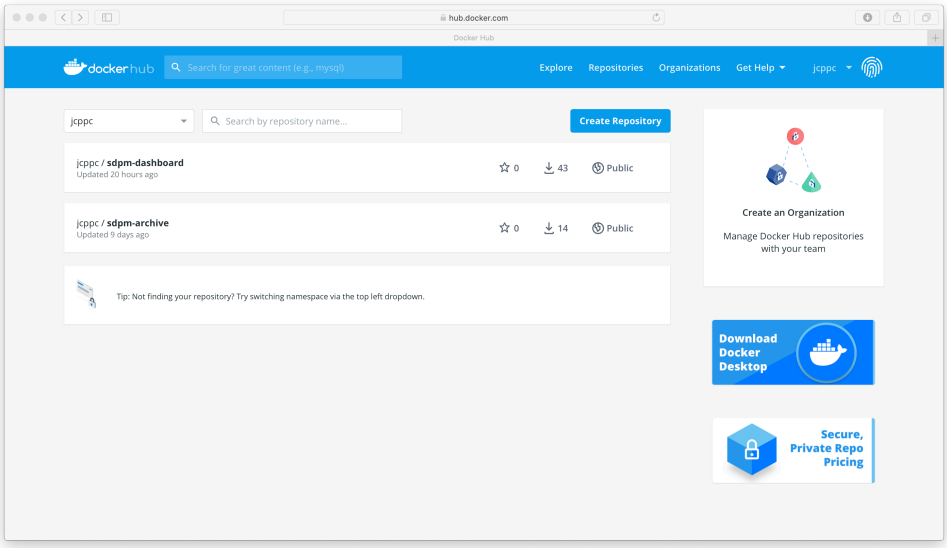

Figure A.25: SDPM Docker Hub Repository

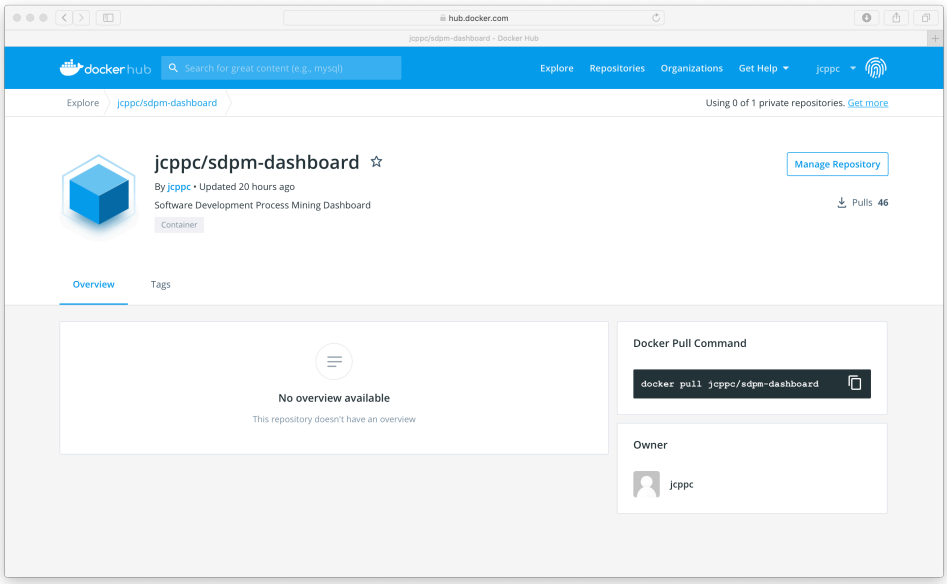

Figure A.26: SDPM Docker Hub Download Details

Listing A.3: Docker Compose File (docker-compose.yml)

```
/**
* This file downloads the full stack(web & db images) for SDPM Dashboard
*   - Install Docker
*   - Create a folder named sdpm-stack
*   - Go on sdpm-stack folder and create a file named docker-compose.yml into it
*   - Copy from line 10 to the end and paste it into file docker-compose.yml
*   - Run : docker-compose up -d
*   - SDPM Dashboard will be running on http://localhost:9000
*/

version: "3.8"

services:

  archive:
    container_name: sdpm-db
    image: mysql:latest
    command: --default-authentication-plugin=mysql_native_password
    restart: on-failure
    environment:
      MYSQL_ROOT_PASSWORD: sdpm2020 #You can change this. It must match with
          ↪ MYSQL_PASSWORD below on the "app" service.
      MYSQL_DATABASE: analytics

  app:
    container_name: sdpm-web
    image: jcppc/sdpm-dashboard:latest
    command: sh -c "yarn install && yarn run prod"
    ports:
      - 9000:9000
    working_dir: /sdpm-dashboard
    environment:
      MYSQL_HOST: sdpm-db
      MYSQL_USER: root
      MYSQL_PASSWORD: sdpm2020 #You can change this. It must match with
          ↪ MYSQL_ROOT_PASSWORD above on the "archive" service.
      MYSQL_DB: analytics
```

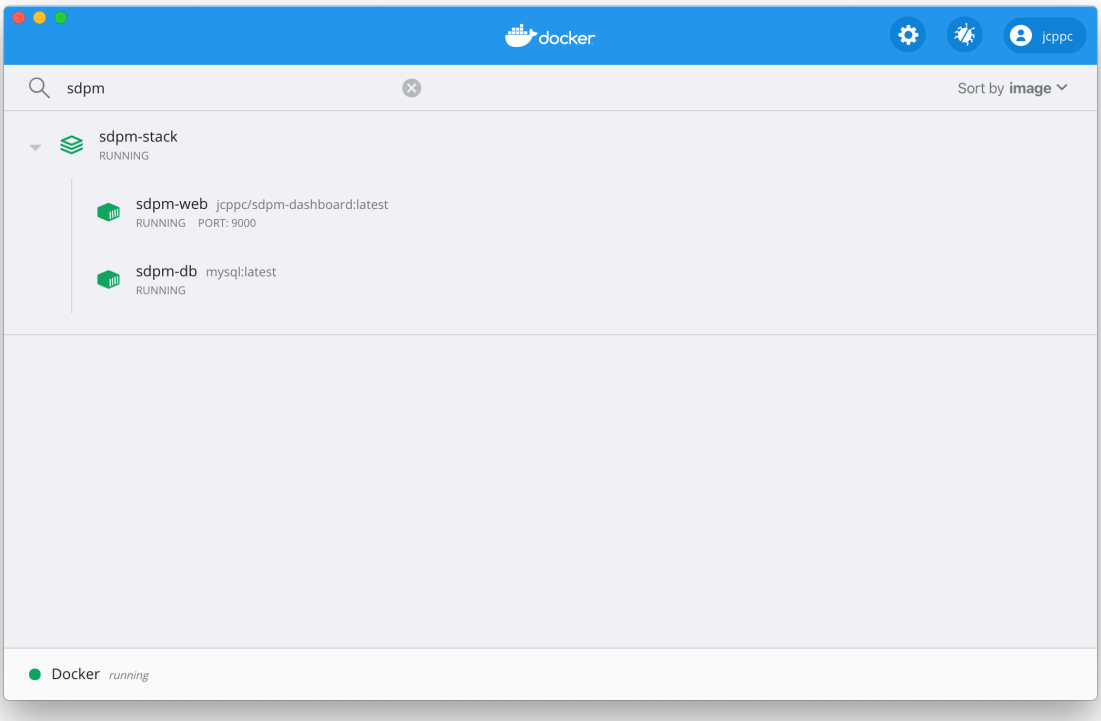

Figure A.27: SDPM Running in Docker Desktop

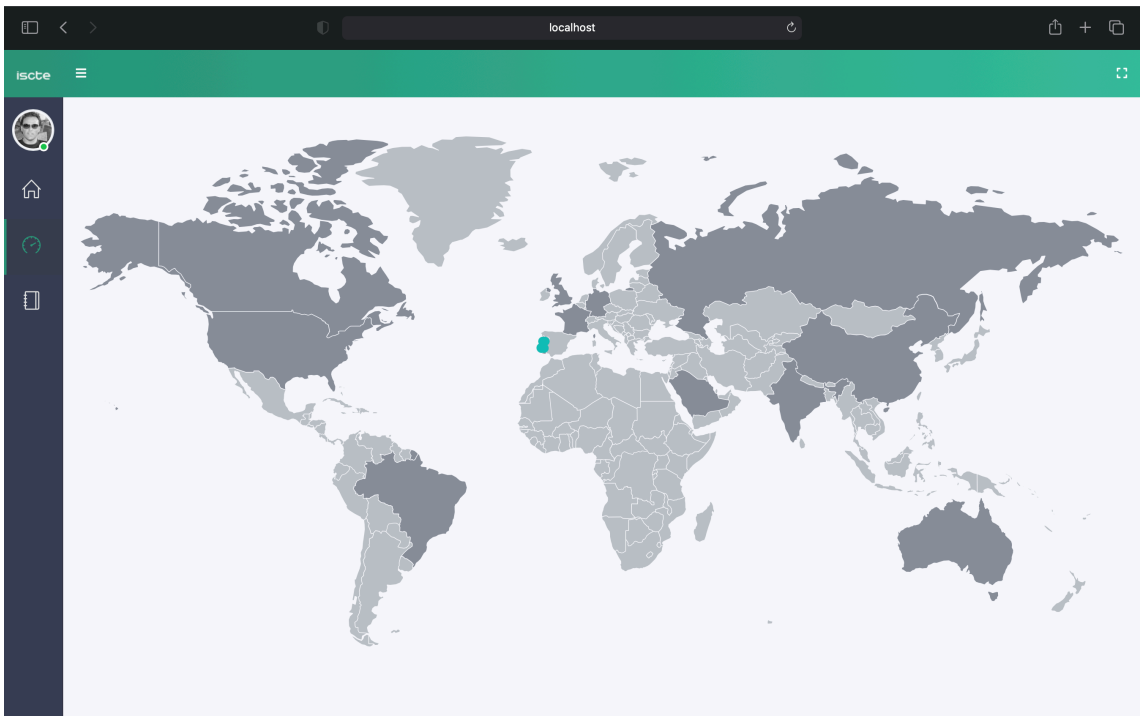

Figure A.28: SDPM Team Dashboard - Geographic View

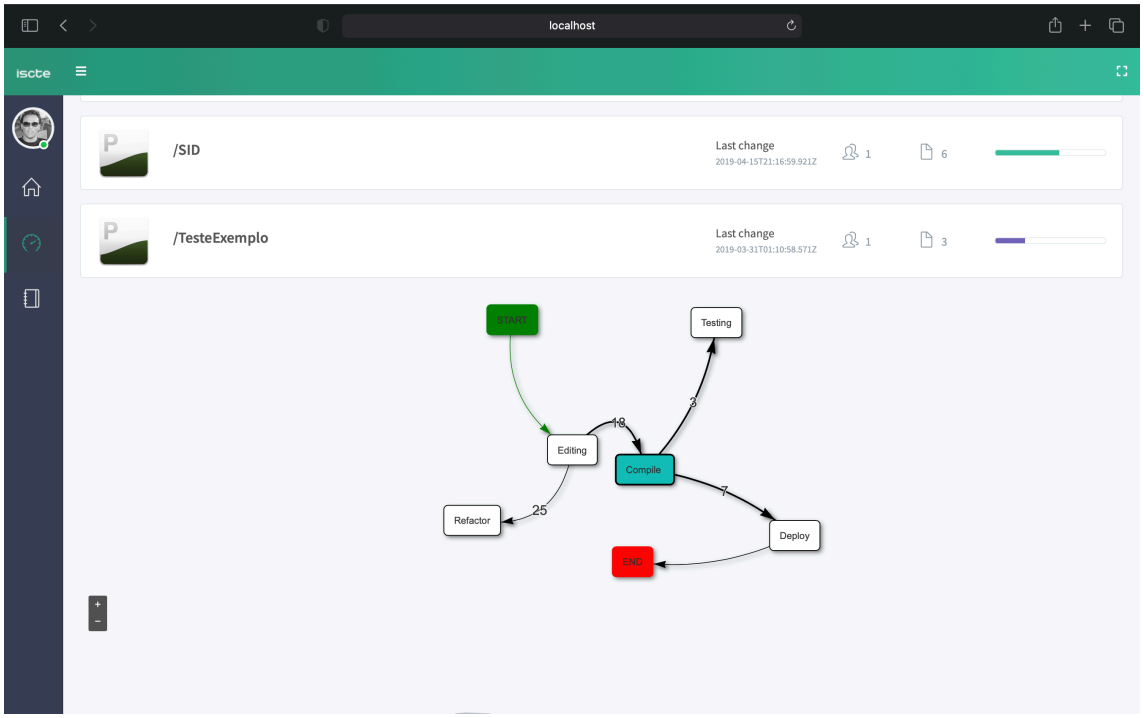

Figure A.29: SDPM Personal Dashboard - Process View

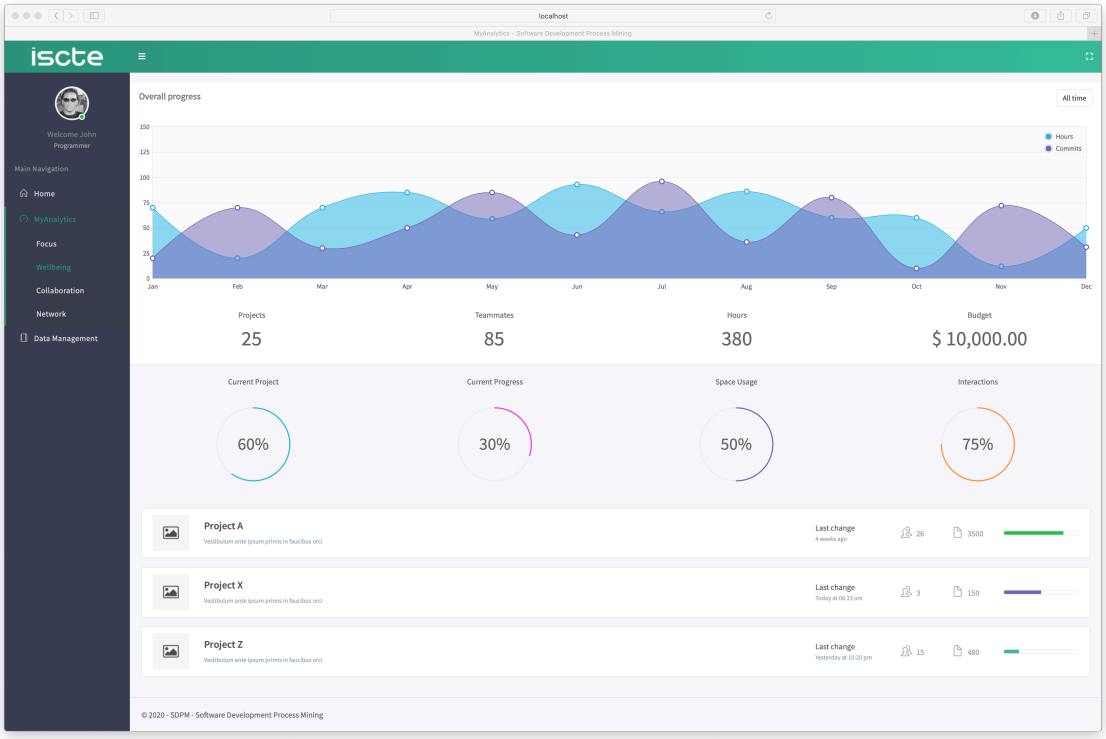

Figure A.30: SDPM Personal Dashboard - Generic KPIs (Prototype)

[This page has been intentionally left blank]

Appendix B.

SLR Complementary Materials

Contents

This Appendix presents the SLR complementary materials, including data demographics, studies comments and assessment.

B.1 Data Extraction

Table B.1: Data Collection Form

#	Item	Description
	General Information	
1	Publication Id	A sequential identifier for each publication.
2	Extraction Date	Date/Time when the data was extracted.
3	Bibliography Reference	The references of each publication.
4	Publication Date	The Date/Time of publishing.
5	Publication Type	The type publication (eg: Journal, Conference, etc).
6	Publisher Name	The name of the publisher.
7	Publication Author(s)	The author(s) of the publication.
	Addressing RQs	
8	Study Type(s)	Extracting the type of empirical study as defined in [144, 223].
9	Data Source(s)	The different types of data sources used in the publications. Admissible values are open.
10	Process Perspective	The timing of when the study was conducted (eg: **Pre-Mortem**, if study was executed before project/product was finished, **Post-Mortem**, if it was conducted after).
11	SDLC Activity(ies)	We followed the SWEBOK to build our list of admissible values [95]. **Implementation** - refers to the activity of constructing artifacts for a new product based on new defined requirements and design. **Maintenance** - refers to the task of maintaining, by changing or evolving an existing software under operation according to early defined specifications. **Testing** - refers to the automated or manual task of finding bugs and/or errors. **Debugging** - is the effort of fixing those known bugs. **Operations** - is related with the phase where the software is under exploration by the end-users. Our approach extends the taxonomy used by [55].
12	Study Stakeholder(s)	The publication outcomes should be targeted to specific individuals in the software development process. We identify them here.
13	Mining Method(s)	The identification of the methods used for data mining/analysis.
14	Analytics Scope(s)	Identifies what type of analytics was performed. We used the valid options identified in [57].

Continued on next page

Table B.1: continued from previous page

#	Item	Description
15	Contribution(s) to SDLC	We framed the admissible options to the following assessment dimensions of software: **Technical Debt/Quality**, **Time**, **Costs**, **Risk** and **Security**. Our approach adapt and extends some of the dimensions and concerns identified earlier in section 2.2.1.
	Findings	
16	Findings and Conclusions	What were the interpretation of the results obtained.
17	Validity	Identifying the threats to the validity of the publication.
18	Relevance	What other relevant outcomes could be inferred from the publication other then the ones in item 15.

B.2 Studies List

Table B.2: Systematic Literature Review Studies.

#	Score	Year	Author	Title	Publication
S01	12	2019	Sultana et al.	A study examining relationships between micro patterns and security vulnerabilities	Software Quality Journal
S02	10.5	2019	Jha et al.	An empirical study of configuration changes and adoption in Android apps	Journal of Systems and Software
S03	10	2017	Hassan et al.	An empirical study of emergency updates for top android mobile apps	Empirical Software Engineering
S04	10	2016	W. Wu et al.	An exploratory study of api changes and usages based on apache and eclipse ecosystems	Empirical Software Engineering
S05	10.5	2017	Taba et al.	An exploratory study on the usage of common interface elements in android applications	Journal of Systems and Software
S06	10	2019	Prana et al.	Categorizing the Content of GitHub README Files	Empirical Software Engineering
S07	10	2018	R. Wu et al.	ChangeLocator: locate crash-inducing changes based on crash reports	Empirical Software Engineering
S08	10.5	2019	Yan et al.	Characterizing and identifying reverted commits	Empirical Software Engineering
S09	10	2018	Salza et al.	Do developers update third-party libraries in mobile apps?	International Conference on Program Comprehension
S10	10	2019	Liu et al.	DroidLeaks: a comprehensive database of resource leaks in Android apps	Empirical Software Engineering
S11	11	2018	Fan et al.	Early prediction of merged code changes to prioritize reviewing tasks	Empirical Software Engineering
S12	10	2016	McIlroy et al.	Fresh apps: an empirical study of frequently-updated mobile apps in the Google play store	Empirical Software Engineering
S13	11	2018	Saborido et al.	Getting the most from map data structures in Android	Empirical Software Engineering

Continued on next page

Table B.2: continued from previous page

#	Score	Year	Author	Title	Publication
S14	10	2017	Guerrouj et al.	Investigating the relation between lexical smells and change- and fault-proneness: an empirical study	Software Quality Journal
S15	10	2014	Fucci et al.	On the role of tests in test-driven development: a differentiated and partial replication	Empirical Software Engineering
S16	10	2014	Mittal et al.	Process mining software repositories from student projects in an undergraduate software engineering course	International Conference on Software Engineering
S17	10.5	2018	Rakha et al.	Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval	Empirical Software Engineering
S18	10	2018	Morales-Ramirez et al.	Speech-acts based analysis for requirements discovery from online discussions	Information Systems Journal
S19	10.5	2018	Li et al.	Studying software logging using topic models	Empirical Software Engineering
S20	11	2018	Hassan et al.	Studying the dialogue between users and developers of free apps in the Google Play Store	Empirical Software Engineering
S21	10	2016	Rakha et al.	Studying the needed effort for identifying duplicate issues	Empirical Software Engineering
S22	10.5	2017	Ye et al.	The structure and dynamics of knowledge network in domain-specific Q&A sites: a case study of stack overflow	Empirical Software Engineering
S23	10.5	2019	Sawant et al.	To react, or not to react: Patterns of reaction to API deprecation	Empirical Software Engineering
S24	10	2019	Cruz et al.	To the attention of mobile software developers: guess what, test your app!	Empirical Software Engineering
S25	10	2017	Li et al.	Towards just-in-time suggestions for log changes	Empirical Software Engineering
S26	10	2017	Izquierdo-Cortazar et al.	Using Metrics to track code review performance	International Conference on Evaluation and Assessment in Software Engineering
S27	10	2016	Munaiah et al.	Vulnerability severity scoring and bounties: Why the disconnect?	International Workshop on Software Analytics

Continued on next page

Table B.2: continued from previous page

#	Score	Year	Author	Title	Publication
S28	10	2015	Tian et al.	What are the characteristics of high-rated apps? A case study on free Android Applications	International Conference on Software Maintenance and Evolution
S29	11	2016	Yang et al.	What security questions do developers ask? a large-scale study of stack overflow posts	Journal of Computer Science and Technology
S30	10.5	2019	Chen et al.	What's Spain's Paris ? Mining analogical libraries from Q&A discussions	Empirical Software Engineering
S31	10	2017	Jiang et al.	Why and how developers fork what from whom in GitHub	Empirical Software Engineering
S32	10.5	2019	Thongtanunam et al.	Will this clone be short-lived? Towards a better understanding of the characteristics of short-lived clones	Empirical Software Engineering

B.3 Comments on Studies

[**S01**] explores the correlation between software vulnerabilities and code-level constructs called micro patterns. The authors analyzed the correlation between vulnerabilities and micro patterns from different viewpoints and explored whether they are related. The conclusion shows that certain micro patterns are frequently present in vulnerable classes and that there is a high correlation between certain patterns that coexist in a vulnerable class [190].

[**S02**] presents an empirical study to analyze commit histories of Android manifest files of hundreds of apps to understand their evolution through configuration changes. The results is a contribution to help developers in identifying change-proneness attributes, including the reasons behind the changes and associated patterns and understanding the usage of different attributes introduced in different versions of the Android platform. In summary, the results show that most of the apps extend core functionalities and improve user interface over time. It detected that significant effort is wasted in changing configuration and then reverting back the change, and that very few apps adopt new attributes introduced by the platform and when they do, they are slow in adopting new attributes. Configuration changes are mostly influenced by functionalities extension, platform evolution and bug reports [103].

[**S03**] studied updates in the Google Play Store by examining more than 44,000 updates of over 10,000 mobile apps, from where 1,000 were identified as emergency updates. After studying the characterirstics of the updates, the authors found that the emergency updates often have a long lifetime (i.e., they are rarely followed by another emergency update) and that updates preceding emergency updates often receive a higher ratio of negative reviews than the emergency updates [84].

[**S04**] analyzed and classified API changes and usages together in 22 framework releases from the Apache and Eclipse ecosystems and their client programs. The authors conclude that missing classes and methods happen more often in frameworks and affect client programs more often than the other API change types do, and that missing interfaces occur rarely in frameworks but affect client programs often. In summary, framework APIs are used on average in 35% of client classes and interfaces and most of such usages could be encapsulated locally and reduced in number. Around 11% of APIs usages could cause ripple effects in client programs when these APIs change. Some suggestions for developers and researchers were made to mitigate the impact of API evolution through language mechanisms and design strategies [214].

[**S05**] extracted commonly used UI elements, denoted as Common Element Sets (CESs), from user interfaces of applications. The highlight the characteristics of CESs that can result in a high user-perceived quality by proposing various metrics. From an empirical study on 1292 mobile applications, the authors observed that CESs of mobile applications widely occur among and across different categories, whilst certain characteristics of CESs can provide a high user-perceived quality. A recommendation is made, aiming to improve the quality of mobile applications, consisting on the adoption of reusable UI templates that are extracted and

summarized from CESs for developers [192].

[**S06**] performed a qualitative study involving the manual annotation of 4,226 README file sections from 393 randomly sampled GitHub repositories and design and evaluate a classifier and a set of features that can categorize these sections automatically. The findings show that information discussing the 'What' and 'How' of a repository hapens very often, while at the same time, many README files lack information regarding the purpose and status of a repository. A classifier was built to predict multiple categories and the F1 score obtained encourages its usage by software repositories owners. The approach presented is said to improve the quality of software repositories documentation and it has the potential to make it easier for the software development community to discover relevant information in GitHub README files [169].

[**S07**] conducted an empirical study on characterizing the bug inducing changes for crashing bugs (denoted as crash-inducing changes). ChangeLocator was also proposed as a method to automatically locate crash-inducing changes for a given bucket of crash reports. The study approach is based on a learning model that uses features originated from the empirical study itself and a model was trained using the data from the historical fixed crashes. ChangeLocator was evaluated with six release versions of the Netbeans project. The analysis and results show that it can locate the crash-inducing changes for 44.7%, 68.5%, and 74.5% of the bugs by examining only top 1, 5 and 10 changes in the recommended list, respectively, which is said to outperform other approaches [213].

[**S08**] explored if one can characterize and identify which commits will be reverted. The authors characterized commits using 27 commit features and build an identification model to identify commits that will be reverted. Reverted commits were identified by analyzing commit messages and comparing the changed content, and extracted 27 commit features that were divided into three dimensions: change, developer and message. An identification model (e.g., random forest) was built and evaluated on an empirical study on ten open source projects including a total of 125,241 commits. The findings show that the 'developer' is the most discriminative dimension among the three dimensions of features for the identification of reverted commits. However, using all the three dimensions of commit features leads to better performance of the created models [217].

[**S09**] conducted an empirical study on the evolution history of almost three hundred mobile apps, by investigating whether mobile developers actually update third-party libraries, checking which are the categories of libraries with respect to the developers' proneness to update their apps, looking for what are the common patterns followed by developers when updating a software library, and whether high- and low-rated apps present any particular update patterns. Results showed that mobile developers rarely update their apps with respect to the used libraries, and when they do, they mainly tend to update the libraries related to the Graphical User Interface, with the aim of keeping the mobile apps updated with the latest design trends. In some cases developers ignore updates because of a poor awareness of the

benefits, or a too high cost/benefit ratio [184].

[**S10**] extracted real resource leak bugs from a bug database named DROIDLEAKS. It consisted in mining 34 popular open-source Android apps, which resulted in a dataset having a total of 124,215 code revisions. After filtering and validating the data, the authors found, on 32 analyzed apps, 292 fixed resource leak bugs, which cover a diverse set of resource classes. To fully comprehend these bugs, they performed an empirical study, which revealed the characteristics of resource leaks in Android apps and common patterns of resource management mistakes made by developers [124].

[**S11**] built a merged code change prediction tool leveraging machine learning techniques, and extracted 34 features from code changes, which were grouped into 5 dimensions: code, file history, owner experience, collaboration network, and text. Experiments were executed on three open source projects (i.e., Eclipse, LibreOffice, and OpenStack), containing a total of 166,215 code changes. Across three datasets, the results show statistically significantly improvements in detecting merged code changes and in distinguishing important features on merged code changes from abandoned ones [65].

[**S12**] studied the frequency of updates of 10,713 mobile apps (the top free 400 apps at the start of 2014 in each of the 30 categories in the Google Play store). It was found that only ~1% of the studied apps are updated at a very frequent rate - more than one update per week and 14% of the studied apps are updated on a bi-weekly basis (or more frequently). Results also show that 45% of the frequently-updated apps do not provide the users with any information about the rationale for the new updates and updates exhibit a median growth in size of 6%. The authors conclude that developers should not shy away from updating their apps very frequently, however the frequency should vary across store categories. It was observed that developers do not need to be too concerned about detailing the content of new updates as it appears that users are not too concerned about such information and, that users highly rank frequently-updated apps instead of being annoyed about the high update frequency [131].

[**S13**] studied the use of map data structure implementations by Android developers and how that relates with saving CPU, memory, and energy as these are major concerns of users wanting to increase battery life. The authors initially performed an observational study of 5713 Android apps in GitHub and then conducted a survey to assess developers' perspective on Java and Android map implementations. Finally, they performed an experimental study comparing HashMap, ArrayMap, and SparseArray variants map implementations in terms of CPU time, memory usage, and energy consumption. The conclusions provide guidelines for choosing among the map implementations: HashMap is preferable over ArrayMap to improve energy efficiency of apps, and SparseArray variants should be used instead of HashMap and ArrayMap when keys are primitive types [183].

[**S14**] detected 29 smells consisting of 13 design smells and 16 lexical smells in 30 releases of three projects: ANT, ArgoUML, and Hibernate. Further, the authors analyzed to what extent

classes containing lexical smells have higher (or lower) odds to change or to be subject to fault fixing than other classes containing design smells. The results obtained bring empirical evidence on the fact that lexical smells can make, in some cases, classes with design smells more fault-prone. In addition, it was empirically demonstrated that classes containing design smells only are more change- and fault-prone than classes with lexical smells only [79].

[**S15**] examined the nature of the relationship between tests and external code quality as well as programmers' productivity in order to verify/refute the results of a previous study. With the focus on the role of tests, a differentiated and partial replication of the original study and related analysis was conducted. The replication involved 30 students, working in pairs or as individuals, in the context of a graduate course, and resulted in 16 software artifacts developed. Significant correlation was found between the number of tests and productivity. No significant correlation found between the number of tests and external code quality. For both cases we observed no statistically significant interaction caused by the subject units being individuals or pairs. Results obtained are consistent with the original study although, as the authors admit, there were changes in the timing constraints for finishing the task and the enforced development processes [70].

[**S16**] presented an application of mining three software repositories: team wiki (used during requirement engineering), version control system (development and maintenance) and issue tracking system (corrective and adaptive maintenance) in the context of an undergraduate Software Engineering course. Visualizations, metrics and algorithms to provide an insight into practices and procedures followed during various phases of a software development life-cycle were proposed and these provided a multi-faceted view to the instructor serving as a feedback tool on development process and quality by students. Event logs produced by software repositories were mined and derived insights such as degree of individual contributions in a team, quality of commit messages, intensity and consistency of commit activities, bug fixing process trend and quality, component and developer entropy, process compliance and verification. Experimentation revealed that not only product but process quality varies signicantly between student teams and mining process aspects can help the instructor in giving directed and specific feedback. Authors, observed that commit patterns characterizing equal and un-equal distribution of workload between team members, patterns indicating consistent activity in contrast to spike in activity just before the deadline, varying quality of commit messages, developer and component entropy, variation in degree of process compliance and bug fixing quality [141].

[**S17**] investigated the impact of the just-in-time duplicate retrieval on the duplicate reports that end up in the ITS of several open source projects, namelly Mozilla-Firefox, Mozilla-Core and Eclipse-Platform. The differences between duplicate reports for open source projects before and after the activation of this new feature were studied. Findings showed that duplicate issue reports after the activation of the just-in-time duplicate retrieval feature are less textually similar, have a greater identification delay and require more discussion to be retrieved as duplicate reports than duplicates before the activation of the feature [172].

[**S18**] exploited a linguistic technique based on speech-acts for the analysis of online discussions with the ultimate goal of discovering requirements-relevant information. The datasets used in the experimental evaluation, which are publicly available, were taken from a widely used open source software project (161120 textual comments), as well as from an industrial project in the home energy management domain. The approach used was able to successfully classify messages into Feature/Enhancement and Other, with significant accuracy. Evidence was found to support the rationale, that there is an association between types of speech-acts and categories of issues, and that there is correlation between some of the speechacts and issue priority, which could open other streams of research [146].

[**S19**] studied the relationship between the topics of a code snippet and the likelihood of a code snippet being logged (i.e., to contain a logging statement). The intuition driving this research, was that certain topics in the source code are more likely to be logged than others. To validate the assumptions a case study was conducted on six open source systems. The analysis gathered evidences that i) there exists a small number of "log-intensive"topics that are more likely to be logged than other topics; ii) each pair of the studied systems share 12% to 62% common topics, and the likelihood of logging such common topics has a statistically significant correlation of 0.35 to 0.62 among all the studied systems. In summary, the findings highlight the topics containing valuable information that can help guide and drive developers' logging decisions [122].

[**S20**] revisits a previous work in more depth by studying 4.5 million reviews with 126,686 responses for 2,328 top free-to-download apps in the Google Play Store. One of the major findings is that the assumption that reviews are static is incorrect. In particular, it is found that developers and users in some cases use this response mechanism as a rudimentary user support tool, where dialogues emerge between users and developers through updated reviews and responses. In addition, four patterns of developers were identified: 1) developers who primarily respond to only negative reviews, 2) developers who primarily respond to negative reviews or to reviews based on their contents, 3) developers who primarily respond to reviews which are posted shortly after the latest release of their app, and 4) developers who primarily respond to reviews which are posted long after the latest release of their app. To perform a qualitative analysis of developer responses to understand what drives developers to respond to a review, the authors analyzed a statistically representative random sample of 347 reviews with responses for the top ten apps with the highest number of developer responses. Seven drivers that make a developer respond to a review were identified, of which the most important ones are to thank the users for using the app and to ask the user for more details about the reported issue. In summary, there were significant evidences found, that it can be worthwhile for app owners to respond to reviews, as responding may lead to an increase in the given rating and that studying the dialogue between user and developer can provide valuable insights which may lead to improvements in the app store and the user support process [85].

[**S21**] empirically examined the effort that is needed for manually identifying duplicate reports

in four open source projects, i.e., Firefox, SeaMonkey, Bugzilla and Eclipse-Platform. Results showed that: (i) More than 50% of the duplicate reports are identified within half a day. Most of the duplicate reports are identified without any discussion and with the involvement of very few people; (ii) A classification model built using a set of factors that are extracted from duplicate issue reports classifies duplicates according to the effort that is needed to identify them with significant values for precision, recall and ROC area; and (iii) Factors that capture the developer awareness of the duplicate issues' peers (i.e., other duplicates of that issue) and textual similarity of a new report to prior reports are the most influential factors found. The results highlight the need for effort-aware evaluation of approaches that identify duplicate issue reports, since the identification of a considerable amount of duplicate reports (over 50%) appear to be a relatively trivial task for developers. As a conclusion, the authors highlight the fact that, to better assist developers, research on identifying duplicate issue reports should put greater emphasis on assisting developers in identifying effort-consuming duplicate issues [173].

[**S22**] analyzed URL sharing activities in Stack Overflow. The approach was to use open coding method to analyze why users share URLs in Stack Overflow, and develop a set of quantitative analysis methods to study the structural and dynamic properties of the emergent knowledge network in Stack Overflow. The findings show: i) Users share URLs for diverse categories of purposes. ii) These URL sharing behaviors create a complex knowledge network with high modularity, assortative mixing of semantic topics, and a structure skeleton consisting of highly recognized knowledge units. iii) The structure of the knowledge network with respect to indegree distribution is scale-free (i.e., stable), in spite of the ad-hoc and opportunistic nature of URL sharing activities, while the outdegree distribution of the knowledge network is not scale-free. iv) The indegree distributions of the knowledge network converge quickly, with small changes over time after the convergence to the stable distribution. The conclusions highlight the fact that the knowledge network is a natural product of URL sharing behavior that Stack Overflow supports and encourages, and proposed an explanatory model based on information value and preferential attachment theories to explain the underlying factors that drive the formation and evolution of the knowledge network in Stack Overflow [219].

[**S23**] questioned if there was really a strong argument for the Java 9 language designers to change the implementation of the deprecation warnings feature after they notice no one was taking seriously those and continued using outdated features. The goal was to start by identifying the various ways in which an API consumer can react to deprecation and then to create a dataset of reaction patterns frequency consisting of data mined from 50 API consumers totalling 297,254 GitHub based projects and 1,322,612,567 type-checked method invocations. Findings show that predominantly consumers do not react to deprecation and a survey on API consumers was done to try to explain this behavior and by analyzing if the APIs deprecation policy had an impact on the consumers' decision to react. The manual inspection of usages of deprecated API artifacts lead to the discovery of six reaction patterns. Only 13% of API consumers update their API versions and 88% of reactions to deprecation is doing nothing. However the survey got a different result, where 69% of respondents say they replace it with

the recommended repalcement. Over 75% of the API barelly affect consumers with deprecation and 15% of the consumers are affected only by 2 APIs(hibernate-core and mongo-java-driver) [186].

[**S24**] investigated working habits and challenges of mobile software developers with respect to testing. A key finding of this exhaustive study, using 1000 Android apps, demonstrates that mobile apps are still tested in a very ad hoc way, if tested at all. However, it is shown that, as in other types of software, testing increases the quality of apps (demonstrated in user ratings and number of code issues). Furthermore, there is evidence that tests are essential when it comes to engaging the community to contribute to mobile open source software. The authors discuss reasons and potential directions to address the findings. Yet another relevant finding of this study is that Continuous Integration and Continuous Deployment (CI/CD) pipelines are rare in the mobile apps world (only 26% of the apps are developed in projects employing CI/CD) - authors argue that one of the main reasons is due to the lack of exhaustive and automatic testing [49].

[**S25**] tries to understand the reasons for log changes and, proposes an approach that can provide developers with log change suggestions as soon as they commit a code change, which is referred to as "just-in-time"suggestions for log changes. A set of measures is derived based on manually examining the reasons for log changes and individual experiences. Those measures were used as explanatory variables in random forest classifiers to model whether a code commit requires log changes. These classifiers can provide just-in-time suggestions for log changes and was evaluated with a case study on four open source projects: Hadoop, Directory Server, Commons HttpClient, and Qpid. Findings show that: i) the reasons for log changes can be grouped along four categories: block change, log improvement, dependence-driven change, and logging issue; ii) the random forest classifiers can effectively suggest whether a log change is needed; iii) the characteristics of code changes in a particular commit and the current snapshot of the source code are the most influential factors for determining the likelihood of a log change in a commit [123].

[**S26**] designed and conducted, with the continuous feedback of the Xen Project Advisory Board, a detailed analysis focused on finding problems associated with the large increase over time in the number of messages related to code review. The increase was being perceived as a potential signal of problems with their code review process and the usage of metrics was suggested to track the performance of it. As a result, it was learned how in fact the Xen Project had some problems, but at the moment of the analysis those were already under control. It was found as well how diferent the Xen and Netdev projects were behaving with respect to code review performance, despite being so similar from many points of view. A comprehensive methodology, fully automated, to study Linux-style code review was proposed [99].

[**S27**] analyzed the Common Vulnerability Scoring System (CVSS) scores and bounty awarded for 703 vulnerabilities across 24 products. CVSS is the de facto standard for vulnerability severity measurement today and is crucial in the analytics driving software fortification. It

was found a weak correlation between CVSS scores and bounties, with CVSS being more likely to underestimate bounty. Such a negative result is suggested to be a cause for concern. The authors, investigated why the measurements were so discordant by i) analyzing the individual questions of CVSS with respect to bounties and ii) conducting a qualitative study to find the similarities and diferences between CVSS and the publicly-available criteria for awarding bounties. It was found that the bounty criteria were more explicit about code execution and privilege escalation whereas CVSS makes no explicit mention of those. Another lesson learnt was that bounty valuations are evaluated solely by project maintainers, whereas CVSS has little provenance in practice [147].

[**S28**] through a case study on 1,492 high-rated and low-rated free apps mined from the Google Play store, investigated 28 factors along eight dimensions to understand how high-rated apps are different from low-rated apps. The search for the most influential factors was also addressed by applying a random-forest classifier to identify high-rated apps. The results show that high-rated apps are statistically significantly different in 17 out of the 28 factors that we considered. The experiment also presents eveidences for the fact that the size of an app, the number of promotional images that the app displays on its web store page, and the target SDK version of an app are the most influential factors [201].

[**S29**] conducted a large-scale study on security-related questions on Stack Overflow. Two heuristics were used to extract from the dataset the questions that are related to security based on the tags of the posts. Later, to cluster different security-related questions based on their texts, an advanced topic model, Latent Dirichlet Allocation (LDA) tuned using Genetic Algorithm (GA) was used. Results show that security-related questions on Stack Overflow cover a wide range of topics, which belong to five main categories: web security, mobile security, cryptography, software security, and system security. Among them, most questions are about web security. In addition, it was found that the top four most popular topics in the security area are "Password", "Hash", "Signature"and "SQL Injection", and the top eight most difficulty security-related topics are "JAVA Security", "Asymetric Encryption", "Bug", "Browser Security", "Windows Authority", "Signature", "ASP.NET"and "Password", suggesting these are the ones in need for more attention [218].

[**S30**] present an approach to recommend analogical libraries based on a knowledge base of analogical libraries mined from tags of millions of Stack Overflow questions. The approach was implemented in a proof-of-concept web application and more than 34.8 thousands of users visited the website from November 2015 to August 2017. Results show evidences that accurate recommendation of analogical libraries is not only possible but also a desirable solution. Authors validated the usefulness of their analogical-library recommendations by using them to answer analogical-library questions in Stack Overflow [41].

[**S31**] explored why and how developers fork what from whom in GitHub. This approach was supported by collecting a dataset containing 236,344 developers and 1,841,324 forks. It was also validated by a survey in order to analyze the programming languages and owners of forked

repositories. Among the main findings we have: i) Developers fork repositories to submit pull requests, fix bugs, add new features and keep copies etc. Developers find repositories to fork from various sources: search engines, external sites (e.g., Twitter, Reddit), social relationships, etc. More than 42% of developers that were surveyed agree that an automated recommendation tool is useful to help them pick repositories to fork, while more than 44.4% of developers do not value a recommendation tool. Developers care about repository owners when they fork repositories. ii) A repository written in a developers' preferred programming language is more likely to be forked. iii) Developers mostly fork repositories from creators. In comparison with unattractive repository owners, attractive repository owners have higher percentage of organizations, more followers and earlier registration in GitHub. The results show that forking is mainly used for making contributions of original repositories, and it is beneficial for OSS community. In summary, there is evidence of the value of recommendation and provide important insights for GitHub to recommend repositories [104].

[**S32**] designed and executed an empirical study on six open source Java systems to better understand the life expectancy of clones. A random forest classifier was built with the aim of determining the life expectancy of a newly-introduced clone (i.e., whether a clone will be short-lived or longlived) and it was confimed to have good accuracy on that task. Results show that a large number of clones (i.e., 30% to 87%) lived in the systems for a short duration. Moreover, it finds that although short-lived clones were changed more frequently than long-lived clones throughout their lifetime, short-lived clones were consistently changed with their siblings less often than long-lived clones. Findings show that the churn made to the methods containing a newly-introduced clone, the complexity and size of the methods containing the newly- introduced clone are highly influential in determining whether the newly-introduced clone will be short-lived. Furthermore, the size of a newly-introduced clone shares a positive relationship with the likelihood that the newly introduced clone will be short-lived. Results suggest that, to improve the efficiency of clone management efforts, such as the planning of the most effective use of their clone management resources in advance, practitioners can leverage the presented classifiers and insights in order to determine the life expectancy of clones [199].

B.4 General Statistics

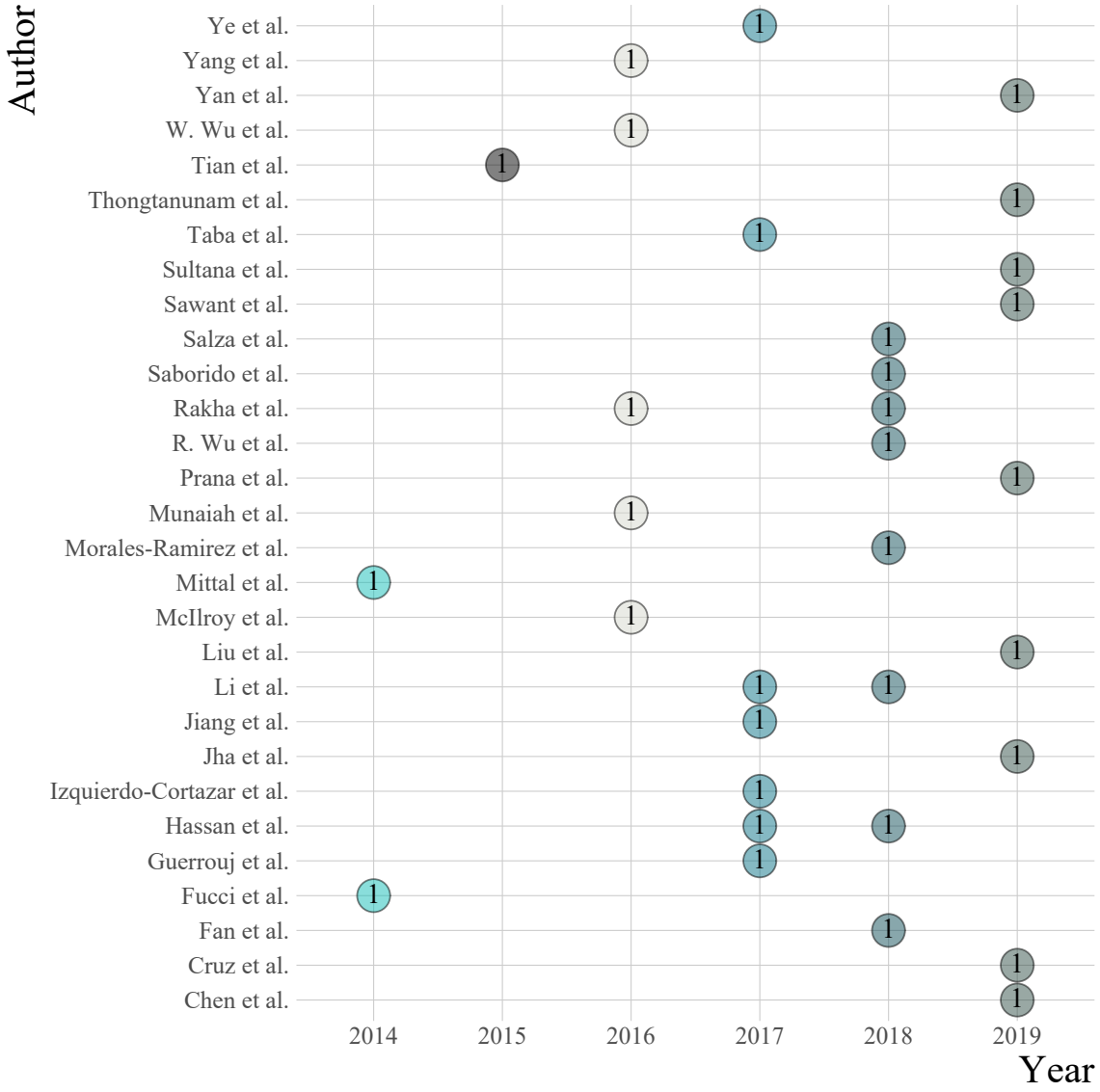

Figure B.1: Number of studies published by each main author over the years

Table B.3: List of all Contributors

Name	Freq.	Perc.	Ref.
Ahmed E. Hassan	10	31.25%	[S03], [S08], [S12], [S17], [S19], [S20], [S21], [S25], [S28], [S32]
David Lo	7	21.88%	[S06], [S08], [S11], [S24], [S28], [S29], [S31]
Weiyi Shang	5	15.62%	[S03], [S19], [S21], [S25], [S32]
Xin Xia	4	12.5%	[S08], [S11], [S29], [S31]
Foutse Khomh	3	9.38%	[S04], [S13], [S14]

Continued on next page

Table B.3: continued from previous page

Name	Freq.	Perc.	Ref.
Giuliano Antoniol	3	9.38%	[S04], [S13], [S14]
Yann-Gael Guéhéneuc	3	9.38%	[S04], [S13], [S14]
Cor-Paul Bezemer	2	6.25%	[S17], [S20]
Heng Li	2	6.25%	[S19], [S25]
Mohamed Sami Rakha	2	6.25%	[S17], [S21]
Safwat Hassan	2	6.25%	[S03], [S20]
Shanping Li	2	6.25%	[S08], [S11]
Shing-Chi Cheung	2	6.25%	[S07], [S10]
Ying Zou	2	6.25%	[S05], [S25]
Zhenchang Xing	2	6.25%	[S22], [S30]
Ajay Kumar Jha	1	3.12%	[S02]
Alberto Bacchelli	1	3.12%	[S23]
Anand Ashok Sawant	1	3.12%	[S23]
Andrea De Lucia	1	3.12%	[S09]
Andrew Meneely	1	3.12%	[S27]
Anna Perini	1	3.12%	[S18]
Ashish Sureka	1	3.12%	[S16]
Benjamin C. M. Fung	1	3.12%	[S14]
Bram Adams	1	3.12%	[S04]
Burak Turhan	1	3.12%	[S15]
Byron J. Williams	1	3.12%	[S01]
Chakkrit Tantithamthavorn	1	3.12%	[S20]
Chang Xu	1	3.12%	[S10]
Christoph Treude	1	3.12%	[S06]
Chunyang Chen	1	3.12%	[S30]
Cosmo D'Uva	1	3.12%	[S09]
Daniel Izquierdo-Cortazar	1	3.12%	[S26]
Dario Di Nucci	1	3.12%	[S09]
Davide Fucci	1	3.12%	[S15]
Deheng Ye	1	3.12%	[S22]
Fabio Palomba	1	3.12%	[S09]
Ferdian Thung	1	3.12%	[S06]
Filomena Ferrucci	1	3.12%	[S09]
Fitsum Meshesha Kifetew	1	3.12%	[S18]
Gede Artha Azriadi Prana	1	3.12%	[S06]
Hongyu Zhang	1	3.12%	[S07]
Iman Keivanloo	1	3.12%	[S05]
Itzel Morales-Ramirez	1	3.12%	[S18]
Jesus M. Gonzalez-Barahona	1	3.12%	[S26]
Jiahuan He	1	3.12%	[S31]
Jian-Ling Sun	1	3.12%	[S29]
Jian Zhang	1	3.12%	[S10]
Jing Jiang	1	3.12%	[S31]
Jue Wang	1	3.12%	[S10]

Continued on next page

Table B.3: continued from previous page

Name	Freq.	Perc.	Ref.
Jun Yan	1	3.12%	[S10]
Kazi Zakia Sultana	1	3.12%	[S01]
Lars Kurth	1	3.12%	[S26]
Latifa Guerrouj	1	3.12%	[S14]
Li Zhang	1	3.12%	[S31]
Lili Wei	1	3.12%	[S10]
Luis Cruz	1	3.12%	[S24]
Megha Mittal	1	3.12%	[S16]
Meiyappan Nagappan	1	3.12%	[S28]
Meng Yan	1	3.12%	[S08]
MingWen	1	3.12%	[S07]
Nachiket Kapre	1	3.12%	[S22]
Nasir Ali	1	3.12%	[S12]
Nelson Sekitoleko	1	3.12%	[S26]
Nuthan Munaiah	1	3.12%	[S27]
Pasquale Salza	1	3.12%	[S09]
Patanamon Thongtanunam	1	3.12%	[S32]
Pavneet Singh Kochhar	1	3.12%	[S31]
Rodrigo Morales	1	3.12%	[S13]
Romain Robbes	1	3.12%	[S23]
RongxinWu	1	3.12%	[S07]
Rubén Saborido	1	3.12%	[S13]
Rui Abreu	1	3.12%	[S24]
Seyyed Ehsan Salamati Taba	1	3.12%	[S05]
Shaohua Wang	1	3.12%	[S05]
Stuart McIlroy	1	3.12%	[S12]
Sunghee Lee	1	3.12%	[S02]
Tanmay Bhowmik	1	3.12%	[S01]
Thushari Atapattu	1	3.12%	[S06]
Tianyong Wu	1	3.12%	[S10]
Tse-Hsun (Peter) Chen	1	3.12%	[S19]
Venera Arnaoudova	1	3.12%	[S14]
Wei Wu	1	3.12%	[S04]
Woo Jin Lee	1	3.12%	[S02]
Xin-Li Yang	1	3.12%	[S29]
Yang Liu	1	3.12%	[S30]
Yepang Liu	1	3.12%	[S10]
Yuan Tian	1	3.12%	[S28]
Yuanrui Fan	1	3.12%	[S11]
Zeinab Kermansaravi	1	3.12%	[S14]
Zhi-Yuan Wan	1	3.12%	[S29]

Table B.4: Statistics per Institution

Institution	Freq.	Perc.	Ref.
Queen's University	11	34.38%	[S03], [S05], [S08], [S12], [S17], [S19], [S20], [S21], [S25], [S28], [S32]
Singapore Management University	7	21.88%	[S06], [S08], [S11], [S24], [S28], [S29], [S31]
Concordia University	4	12.5%	[S03], [S19], [S25], [S32]
Zhejiang University	4	12.5%	[S08], [S11], [S29], [S31]
École Polytechnique de Montréal	3	9.38%	[S04], [S13], [S14]
Monash University	3	9.38%	[S08], [S11], [S30]
Hong Kong University of Science and Technology	2	6.25%	[S07], [S10]
Nanyang Technological University	2	6.25%	[S22], [S30]
Rochester Institute of Technology	2	6.25%	[S27], [S28]
University of Adelaide	2	6.25%	[S06], [S20]
University of Zurich	2	6.25%	[S09], [S23]
Australian National University	1	3.12%	[S30]
Beihang University	1	3.12%	[S31]
Bitergia	1	3.12%	[S26]
Citrix	1	3.12%	[S26]
Delft University of Technology	1	3.12%	[S23]
École de Technologie Supérieure	1	3.12%	[S14]
Fondazione Bruno Kessler	1	3.12%	[S18]
Free University of Bozen-Bolzano	1	3.12%	[S23]
Indraprastha Institute of Information Technology	1	3.12%	[S16]
INESC ID	1	3.12%	[S24]
INFOTEC	1	3.12%	[S18]
Kyungpook National University	1	3.12%	[S02]
McGill University	1	3.12%	[S14]
Mississippi State University	1	3.12%	[S01]
Nanjing University	1	3.12%	[S10]

Continued on next page

Table B.4: continued from previous page

Institution	Freq.	Perc.	Ref.
Southern University of Science and Technology	1	3.12%	[S10]
Universidad Rey Juan Carlos	1	3.12%	[S26]
Universitá della Svizzera Italiana	1	3.12%	[S09]
University of Chinese Academy of Sciences	1	3.12%	[S10]
University of Lisbon	1	3.12%	[S24]
University of Melbourne	1	3.12%	[S32]
University of Newcastle	1	3.12%	[S07]
University of Oulu	1	3.12%	[S15]
University of Salerno	1	3.12%	[S09]
University of Waterloo	1	3.12%	[S12]
Vrije Universiteit Brussel	1	3.12%	[S09]
Washington State University	1	3.12%	[S14]

Table B.5: Statistics per Continent and Country

	Freq.	Perc.	Ref.
Continent			
North America	18	56.25%	[S01], [S03], [S04], [S05], [S08], [S12], [S13], [S14], [S17], [S18], [S19], [S20], [S21], [S25], [S26], [S27], [S28], [S32]
Asia	13	40.62%	[S02], [S06], [S07], [S08], [S10], [S11], [S16], [S22], [S24], [S28], [S29], [S30], [S31]
Oceania	7	21.88%	[S06], [S07], [S08], [S11], [S20], [S30], [S32]
Europe	6	18.75%	[S09], [S15], [S18], [S23], [S24], [S26]
Country			
Canada	14	43.75%	[S03], [S04], [S05], [S08], [S12], [S13], [S14], [S17], [S19], [S20], [S21], [S25], [S28], [S32]
Singapore	9	28.12%	[S06], [S08], [S11], [S22], [S24], [S28], [S29], [S30], [S31]
Australia	7	21.88%	[S06], [S07], [S08], [S11], [S20], [S30], [S32]
China	6	18.75%	[S07], [S08], [S10], [S11], [S29], [S31]
USA	5	15.62%	[S01], [S14], [S26], [S27], [S28]
Italy	3	9.38%	[S09], [S18], [S23]
Switzerland	2	6.25%	[S09], [S23]
Belgium	1	3.12%	[S09]
Finland	1	3.12%	[S15]
India	1	3.12%	[S16]
Mexico	1	3.12%	[S18]
Portugal	1	3.12%	[S24]
Republic of Korea	1	3.12%	[S02]
Spain	1	3.12%	[S26]
The Netherlands	1	3.12%	[S23]

B.5 Studies Appraisal

The following acronyms were used for SLR results interpretation:

- **Study Type**

  CS-Case Study, **ECS**-Exploratory Case Study, **QE**-Quasi-Experiment, **S**-Survey

- **SDLCActivities**

  D-Debugging, **I**-Implementation, **M**-Maintenance, **O**-Operations, **T**-Testing

- **Project Stakeholders**

  D-Developers, **E**-Educators, **EU**-End-Users, **T**-Testers, **PM**-Product Managers

  PjM-Project Managers, **R**-Researchers, **RE**-Requirements Engineers

- **Analytics Scope**

  Des-Descriptive Analytics, **Dia**-Diagnostics Analytics

  Pred-Predictive Analytics, **Pres**-Prescriptive Analytics

The following taxonomy was used to assess the SDLC contributions:

- **The benefit is:**

Absent (0)	○	Not addressed
Weak (0.25)	◔	Implicitly addressed
Moderate (0.5)	◑	Explicitly addressed (not detailed)
Strong (0.75)	◕	Explained with details and implications
Complete (1)	●	Fully explained, validated and replicable

Table B.6: Systematic Literature Review Results.

Study	Study Type	Data Sources	Process Perspective	SDLC Activities	Project Stakeholders	Mining Methods	Analytics Scope	Contributions to SDLC				
								Technical Debt	Time Management	Costs Control	Risks Assessment	Security Analysis
S01	CS	Vulnerability Reports,Apache Tomcat Archive,SecuriBench Archive	Post-Mortem	I,T	D,T	Descriptive Statistics,Pattern Extraction,Correlation Analysis	Des,Dia	◕	◔	◔	◔	◕
S02	CS	F-Droid Repository,GitHub Repositories	Post-Mortem	I	D	Descriptive Statistics,Pattern Extraction,Correlation Analysis	Des,Dia	◕	◔	◔	○	○
S03	CS	F-Droid Repository,Google Play Store	Post-Mortem	O	D,PM	Descriptive Statistics,Pattern Extraction	Des,Dia	◑	○	○	○	○
S04	ECS	Maven Repositories	Post-Mortem	I	D	Descriptive Statistics,Hyphotesis Testing,Correlation Analysis	Des,Dia	◕	○	◔	○	○
S05	ECS	Google Play Store	Post-Mortem	O	D	Descriptive Statistics,Hyphotesis Testing,Correlation Analysis	Des,Dia	◕	○	○	○	○
S06	QE,S	GitHub Repositories	Post-Mortem	I	D,PM	Descriptive Statistics,Pattern Extraction,Classifier Learning	Des,Dia,Pred	◕	○	○	○	○
S07	QE	NetBeans Source Code Repository,BugZilla,Exception Reports	Post-Mortem	I,D,M	D	Descriptive Statistics,Pattern Extraction,Heuristic Features,Classifier Learning	Des,Dia	◕	○	○	○	○
S08	CS	Git Repositories	Post-Mortem	I,D,M	D	Descriptive Statistics,Feature Extraction,Correlation Analysis,Redundancy Analysis,Classifier Learning	Des,Dia,Pred	◕	◔	◔	○	○
S09	ECS	F-Droid Repository,SVN Repositories,GitHub Repositories,BinTray,JCenter,Maven Repositories,Google	Post-Mortem	I,D,M	D	Descriptive Statistics,Pattern Extraction	Des,Dia	◕	○	○	○	○
S10	ECS	F-Droid Repository,GitHub Repositories,Google Play Store	Post-Mortem	I,D,M	D	Descriptive Statistics,Pattern Extraction	Des,Dia	◕	○	○	○	○

Continued on next page

Table B.6: continued from previous page

Study	Study Type	Data Sources	Process Perspective	SDLC Activities	Project Stakeholders	Mining Methods	Analytics Scope	Contributions to SDLC				
								Technical Debt	Time Management	Costs Control	Risks Assessment	Security Analysis
S11	QE	Gerrit	Post-Mortem	I,D,M	D	Descriptive Statistics,Hyphotesis Testing,Redundancy Analysis,Feature Extraction,Correlation Analysis,Classifier Learning	Des,Dia,Pred	◕	◔	◔	○	○
S12	QE	Google Play Store	Post-Mortem	I,D,M	D,PM	Descriptive Statistics	Des,Dia	◕	○	○	○	○
S13	ECS,QE,S	GitHub Repositories,Google Forms	Post-Mortem	I,M	D	Descriptive Statistics,Pattern Extraction	Des,Dia	◕	○	○	○	○
S14	QE	Git Repositories,SVN Repositories,BugZilla,JIRA	Post-Mortem	I,M	D	Descriptive Statistics,Hyphotesis Testing,Correlation Analysis	Des,Dia	◕	○	○	○	○
S15	QE	Online Survey,Lab Computers	Post-Mortem	I	D	Descriptive Statistics,Hyphotesis Testing,Correlation Analysis	Des,Dia	◕	◑	○	○	○
S16	ECS	Team Wiki (BitBucket),Mercurial Repositories,Git Repositories,BugZilla	Post-Mortem	I	D	Descriptive Statistics,Process Mining	Des,Dia	◕	○	○	○	○
S17	QE	BugZilla	Post-Mortem	I,M	D,R	Descriptive Statistics,Hyphotesis Testing,Correlation Analysis	Des,Dia	◕	○	○	○	○
S18	QE	Apache OpenOffice Issue Tracking System,SEnerCON Feedback Gathering System	Post-Mortem	I,M,O	D,PM,RE	Descriptive Statistics,Hyphotesis Testing,Correlation Analysis,Classifier Learning	Des,Dia,Pred	◕	◔	○	○	○
S19	QE	Git Repositories	Post-Mortem	I	D	Descriptive Statistics,Correlation Analysis,Topic Modeling,Regression Models	Des,Dia,Pred	◕	◔	○	○	○

Continued on next page

Table B.6: continued from previous page

Study	Study Type	Data Sources	Process Perspective	SDLC Activities	Project Stakeholders	Mining Methods	Analytics Scope	Contributions to SDLC				
								Technical Debt	Time Management	Costs Control	Risks Assessment	Security Analysis
S20	ECS	Google Play Store	Post-Mortem	I,M,O	D,EU,PM,R	Descriptive Statistics,Correlation Analysis,Mixed-Effect Models,Cluster Analysis,Regression Models	Des,Dia,Pred	◕	○	○	○	○
S21	QE	BugZilla	Post-Mortem	I,M	D	Descriptive Statistics,Correlation Analysis,Classifier Learning	Des,Dia,Pred	◔	◑	◔	○	○
S22	ECS	StackOverflow	Post-Mortem	I,M	D	Descriptive Statistics,Correlation Analysis,Topic Modeling,Cluster Analysis	Des,Dia	◑	○	○	○	○
S23	ECS,S	GitHub Repositories,Online Survey	Post-Mortem	I,M	D	Descriptive Statistics,Pattern Extraction	Des,Dia	◑	◔	○	○	○
S24	QE,S	F-Droid Repository,GitHub Repositories,Google Play Store,Online Survey	Post-Mortem	I,M,T	D,T	Descriptive Statistics,Correlation Analysis	Des,Dia	◕	○	○	○	○
S25	ECS	Version Control Repositories	Post-Mortem	I,M	D	Descriptive Statistics,Correlation Analysis,Classifier Learning,Cluster Analysis	Des,Dia,Pred	◕	○	○	○	○
S26	ECS	Mailing List,Git Repositories	Post-Mortem	I,M	D,PjM	Descriptive Statistics	Des,Dia	◑	◑	○	○	○
S27	ECS	Android Issue Tracker,Chrome Releases Blog,Chromium Issue Tracker,HackerOne Bug Bounty Platform	Post-Mortem	I,M	D,PM	Descriptive Statistics,Correlation Analysis	Des,Dia	◔	○	○	○	◑
S28	ECS	Google Play Store	Post-Mortem	I,M,O	D,PM	Descriptive Statistics,Correlation Analysis	Des,Dia	◑	○	○	○	○
S29	ECS	StackOverflow	Post-Mortem	I,M	D,R,PjM,E	Descriptive Statistics,Topic Modeling,Genetic Algorithms	Des,Dia	◔	○	○	○	◑

Continued on next page

Table B.6: continued from previous page

Study	Study Type	Data Sources	Process Perspective	SDLC Activities	Project Stakeholders	Mining Methods	Analytics Scope	Contributions to SDLC				
								Technical Debt	Time Management	Costs Control	Risks Assessment	Security Analysis
S30	ECS	StackOverflow	Post-Mortem	I,M,T	D	Descriptive Statistics,Association Rules,Natural Language Processing	Des,Dia,Pres	◕	◔	○	○	◔
S31	ECS,S	GitHub Repositories	Post-Mortem	I	D	Descriptive Statistics	Des,Dia	◑	○	○	○	○
S32	ECS	Git Repositories	Post-Mortem	I	D	Descriptive Statistics,Generalized Suffix Trees,Correlation Analysis,Cluster Analysis,Classifier Learning	Des,Dia,Pred	◕	○	○	○	○

Appendix C.

Unveilling Process Insights Materials

Contents

This Appendix presents additional contents to provide context to chapter 4.

C.1 Algorithms shown in Model Evaluations

RandomCommittee. Method for building an ensemble of randomizable base classifiers. Each base classifier is built using a different random seed number (but based one the same data). The final prediction is a straight average of the predictions generated by the individual base classifiers.

RandomSubSpace. This method constructs a decision tree based classifier that maintains highest accuracy on training data and improves on generalization accuracy as it grows in complexity. The classifier consists of multiple trees constructed systematically by pseudo-randomly selecting subsets of components of the feature vector, that is, trees constructed in randomly chosen sub-spaces.

RandomForest. Method for constructing a forest of random trees. It consists of a learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

RepTree. Fast decision tree learner. Builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning (with back-fitting). Only sorts values for numeric attributes once. Missing values are dealt with by splitting the corresponding instances into pieces.

LMT. 'Logistic Model Trees' are classification trees with logistic regression functions at the leaves. The algorithm can deal with binary and multi-class target variables, numeric and nominal attributes and missing values.

Logistic Regression. Method for building and using a multinomial logistic regression model with a ridge estimator. Logistic regression is a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although more complex extensions exist.

LWL. The Locally Weighted Learning method uses an instance-based algorithm to assign instance weights which are then used by a specified WeightedInstancesHandler. Can do classification (e.g. using naive Bayes) or regression (e.g. using linear regression).

LinearNNSearch. This method implements the brute force search algorithm for nearest neighbour search.

DecisionTable. Builds and uses a simple decision table majority classifier.

Bagging. Method for bagging a classifier to reduce variance. Can do classification and regression depending on the base learner.

KStar. Is an instance-based classifier, that is, the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function. It differs from other instance-based learners in that it uses an entropy-based distance function.

C.2 Models Performance Metrics

Several metrics are employed in machine learning classification problems for the assessment of models' performance. The list bellow serves as a brief introduction to those metrics and how they are calculated.

True Positive (TP). The outcome prediction was positive, and so was the actual value (e.g., the refactoring was predicted as likely to be manual has been effectively manual).

False Positive (FP). The outcome prediction was positive, but the actual value was negative (e.g., refactoring was predicted as likely to be automatic, but it was indeed not automatic(manual)). **Type I error.**

True Negative (TN). The outcome prediction was negative, and so was the actual value (e.g., the booking was predicted as likely not to be manual and has been effectively not manual).

False Negative (FN). The outcome prediction was negative, but the actual value was positive (e.g., refactoring was predicted as likely not to be automatic(manual), but it was indeed automatic). **Type II error.**

Area Under the Curve (AUC). Measure of success calculated from the area under the plot of true positive rate (TPR) against false positive rate (FPR).

Accuracy. Measure of outcome correctness. Measures the proportion of true results among the total number of predictions and is defined as follows:

$$\textbf{Accuracy} = \frac{\sum TP + \sum TN}{\sum TP + \sum TN + \sum FP + \sum FN} \tag{C.1}$$

False Positive Rate (FPR or Fall-out). Measures the probability of a positive prediction result and the actual value being negative (e.g., probability of a refactoring being predicted as likely to be automatic and effectively it was not). It is defined as follows:

$$\textbf{FPR} = \frac{\sum FP}{\sum FP + \sum TN} \tag{C.2}$$

Precision (Pre.). Measures the proportion of correct positive predictions. It is defined as follows:

$$\textbf{Precision} = \frac{\sum TP}{\sum TP + \sum FP} \tag{C.3}$$

True Negative Rate (TNR or Specificity). Measures the probability of a negative prediction result and the actual value being negative (e.g., probability of a refactoring being identified as not automatic and effectively it was not). It is defined as follows:

$$\textbf{TNR} = \frac{\sum TN}{\sum TN + \sum FP} \tag{C.4}$$

True Positive Rate (TPR, Recall or Sensitivity). Measures the probability of a positive prediction result and the actual value being positive (e.g., probability of a refactoring being identified as likely to be automatic and effectively was automatic). It is defined as follows:

$$\textbf{TPR} = \frac{\sum TP}{\sum TP + \sum FN} \tag{C.5}$$

F1 Score. Is the harmonic mean of precision and sensitivity. It is defined as follows:

$$\textbf{F1} = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{C.6}$$

$$\textbf{F1} = 2 * \frac{\frac{\sum TP}{\sum TP + \sum FP} * \frac{\sum TP}{\sum TP + \sum FN}}{\frac{\sum TP}{\sum TP + \sum FP} + \frac{\sum TP}{\sum TP + \sum FN}} \tag{C.7}$$

Appendix D.

Practices and Fingerprints Materials

Contents

This Appendix presents additional contents to support chapter 5 context and contributions.

D.1 Development Sessions Fingerprints

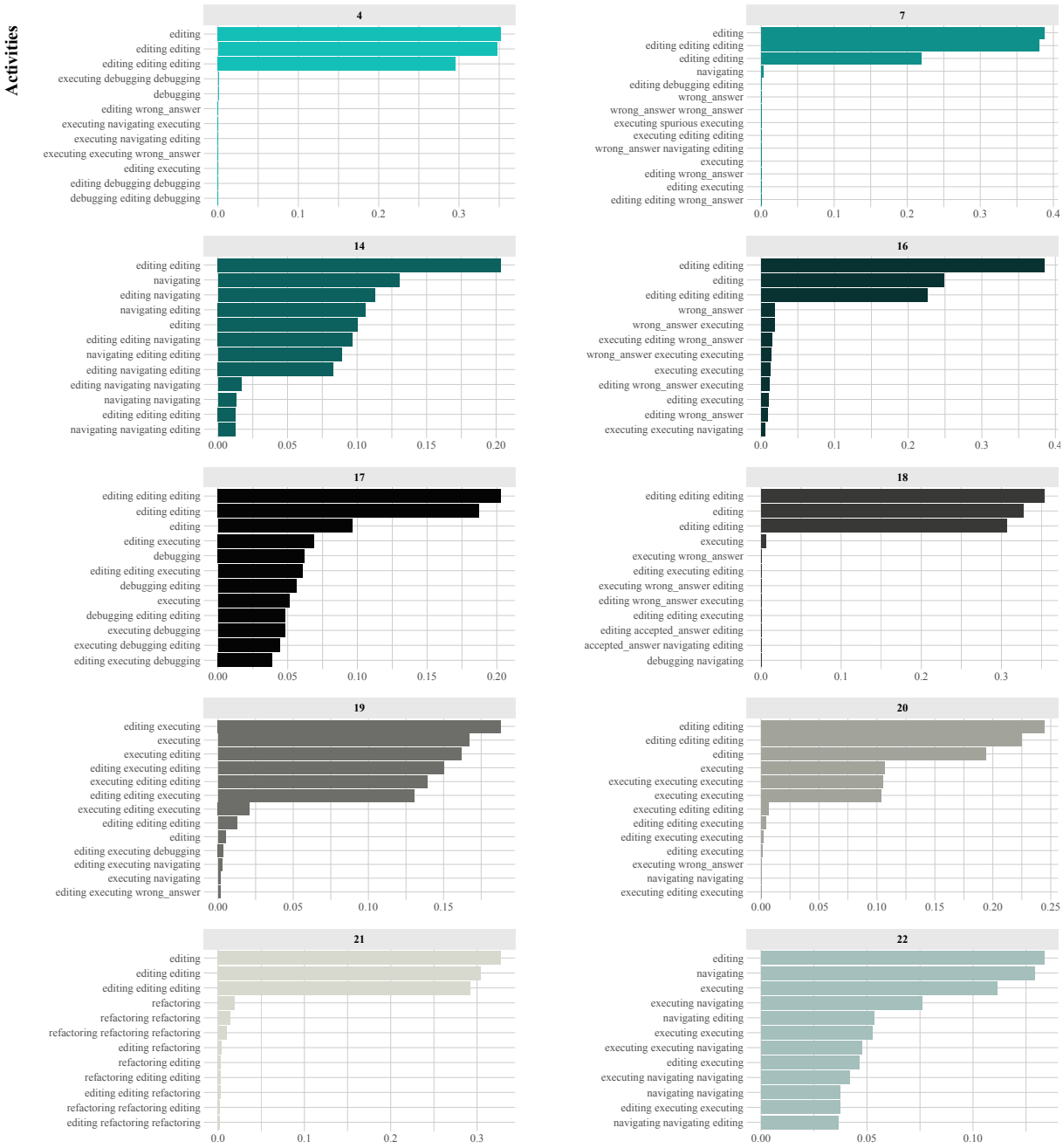

Figure D.1: All Development Sessions Fingerprints

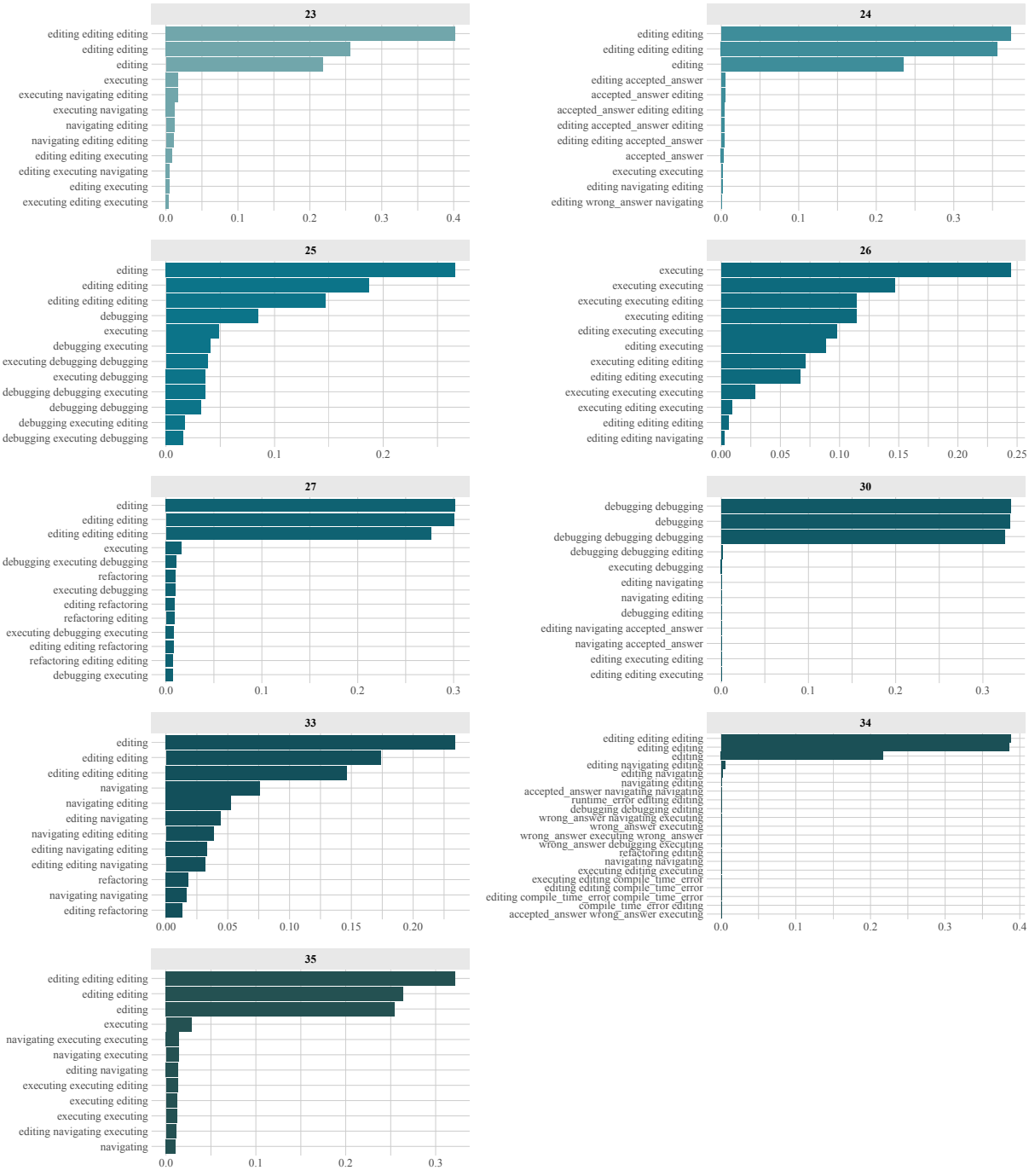

Figure D.2: All Development Sessions Fingerprints - continued

D.2 Frequency of Participants per Fingerprint

Figure D.3: Frequency of participants per development fingerprint

Figure D.4: Frequency of participants per development fingerprint - continued

[This page has been intentionally left blank]