



UNIVERSIDADE D  
COIMBRA

Rita Maria Gonçalves Caeiro da Fonseca

## MINING SOFTWARE PROJECT REPOSITORIES

Dissertation in the context of the Masters in Informatics Engineering, specialization in Software Engineering, advised by Professor Mário Alberto da Costa Zenha Rela and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023





UNIVERSIDADE D  
**COIMBRA**

Rita Maria Gonçalves Caeiro da Fonseca

## MINING SOFTWARE PROJECT REPOSITORIES

Dissertation in the context of the Masters in Informatics Engineering, specialization in Software Engineering, advised by Professor Mário Alberto da Costa Zenha Rela and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023



# Acknowledgements

First of all, I would like to thank my family: my brother Manuel Fonseca for constantly reminding me to "stay cool", but especially my parents, Eduardo Fonseca and Helena Caeiro, for all the effort they made so that I could get here, and for always supporting and believing in me. None of this would be possible without them.

I would also like to thank my dear closest friends (you know who you are) and my boyfriend for always being there for me during these 5 years of college, through the good and the bad, and for making Coimbra my new home.

I thank the University of Coimbra and especially my advisor, Professor Mário Relá, for presenting me with such a great research area and for giving me the opportunity and the support to write this thesis alongside him.

I would like to thank Isabel Batista for always being such a big reference for me and one of my main supporters in the world of IT.

Last but not least, I would like to thank the students from the Software Engineering class of the 2022–2023 edition for letting me use their work to test my own.



## Sumário

A mineração de processos é um método que analisa e melhora os processos de negócio a partir dos dados gerados durante a execução dos mesmos. Para aumentar a sua compreensão e descobrir áreas propícias a melhorias, estes métodos usam um conjunto de técnicas tal como a descoberta de processos, verificação de conformidade, e análise de desempenho. A mineração de processos permite que empresas obtenham uma maior compreensão relativa aos seus processos de maneira a que sejam capazes de tomar as melhores decisões e aumentar a sua eficácia e desempenho.

Nesta tese, vamos explorar o potencial que a mineração de processos tem na extração de informação de repositórios de software. O objetivo será assim desenvolver um sistema que irá modelar um processo, destacar as diferentes atividades que tiveram lugar no respetivo repositório, e como é que estas se relacionaram ao longo do tempo.

Numa primeira iteração, foi desenvolvido um sistema preliminar de acordo com um certo conjunto de requisitos identificados, tal como a concretização de uma primeira arquitetura para o nosso sistema e um primeiro momento de validação dos resultados.

De seguida, a arquitetura do nosso sistema foi repensada, tendo agora um conjunto de regras com o fim de percorrerem os diversos eventos retirados de um repositório. Assim, fomos capazes de os caracterizar em diferentes tipos de atividades que, por sua vez, foram utilizadas para modelar o processo final. Esta tese termina com uma validação final dos modelos obtidos.

## Palavras-Chave

Mineração de Processos em Repositórios de Software, GitLab, GitHub, Modelação de Processos, Engenharia de Software





## Abstract

Process mining is a method for analyzing and enhancing business processes by learning from data generated during process execution. To discover insights and identify areas for development, it makes use of a span of techniques, including process discovery, conformance verification, and performance analysis. Process mining enables companies to obtain a clear understanding of their processes and make wise decisions about how to enhance them by combining data from many sources.

In this thesis, we are going to explore the potential of process mining when it comes to software process repositories. Given this, our goal is to develop a system that will model a process, highlighting the different activities that took place in the same repository, and how they related with each other throughout the time of the project.

In the first iteration, a preliminary system was developed according to a set of requirements that were identified previously, as well as the designing of the first architecture for the preliminary system and a first moment of validation of the results obtained.

Following these results, we designed the architecture for the final version of our system, with a new rules engine interface thought to traverse through the events that were retrieved from a repository. This way, we were able to characterise the different types of activities that were then used to model the final process of the project from the repository. This thesis ends with a final validation of the model obtained.

## Keywords

Process Mining Software Repositories, GitLab, GitHub, Process Modeling, Software Engineering



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background and Context . . . . .	1
1.2	What we are trying to solve . . . . .	2
1.3	How we are going to solve it . . . . .	2
1.4	Structure of the thesis . . . . .	4
<b>2</b>	<b>State of the Art</b>	<b>5</b>
2.1	Process Mining . . . . .	5
2.1.1	Types of Process Mining . . . . .	6
2.1.2	In Practice . . . . .	9
2.2	Process Mining Software Repositories . . . . .	11
2.2.1	Mining Software Repositories . . . . .	11
2.2.2	Mining Software Repositories into Processes . . . . .	11
2.2.3	PMSR Usage Examples . . . . .	12
2.3	Summary . . . . .	13
<b>3</b>	<b>Methodology</b>	<b>15</b>
3.1	Research Questions . . . . .	15
3.1.1	RQ1: Is it possible to turn raw events into activities? . . . . .	16
3.1.2	RQ2: Is it possible to retrieve processes from a set of activities? . . . . .	17
3.2	Design of the study . . . . .	17
3.2.1	Planning . . . . .	18
3.2.2	User Roles . . . . .	19
3.2.3	Testing . . . . .	20
3.2.4	Tools . . . . .	21
3.3	Risk Analysis . . . . .	23
3.4	Summary . . . . .	24
<b>4</b>	<b>Preliminary Analysis</b>	<b>25</b>
4.1	Functional Requirements . . . . .	26
4.2	Non-Functional Requirements . . . . .	27
4.3	Constraints and Limitations . . . . .	28
4.4	Architecture . . . . .	29
4.5	Development of the alpha-version . . . . .	31
4.6	Other useful sources . . . . .	33
4.7	Problems Encountered . . . . .	34
4.8	Verification and Testing . . . . .	37
4.9	Discussion of the results . . . . .	39
4.10	Summary . . . . .	42

<b>5</b>	<b>System's Architecture</b>	<b>43</b>
5.1	Playground . . . . .	43
5.2	Architecture . . . . .	44
5.2.1	Fetching Data . . . . .	45
5.2.2	Aggregator . . . . .	46
5.2.3	Data Fusion . . . . .	46
5.2.4	Rules Engine . . . . .	47
5.2.5	Process Miner . . . . .	48
5.3	Summary . . . . .	48
<b>6</b>	<b>Discussion of Results</b>	<b>49</b>
6.1	Analysis . . . . .	49
6.1.1	Quirked Up Software . . . . .	50
6.1.2	KahUC . . . . .	51
6.1.3	byDeiCrowd . . . . .	52
6.2	Final Remarks . . . . .	53
6.3	Threats to Validity . . . . .	54
6.4	Summary . . . . .	56
<b>7</b>	<b>Conclusion</b>	<b>57</b>
7.1	Overall View . . . . .	57
7.2	Future Work . . . . .	59
<b>Appendix A Gantt Diagrams</b>		<b>67</b>
<b>Appendix B User Stories</b>		<b>71</b>
<b>Appendix C Other Non-Functional Requirements</b>		<b>79</b>
<b>Appendix D Student Feedback Form</b>		<b>81</b>
<b>Appendix E Rules Table</b>		<b>85</b>
<b>Appendix F Modelled Processes</b>		<b>89</b>

# List of Figures

2.1	Petri net of an agile workflow process. . . . .	7
2.2	Spaghetti Process of a Dutch Hospital (from [1]). . . . .	9
2.3	Lasagna Process of a Dutch Municipality (from [1]). . . . .	10
3.1	Flow of data thought for our system. . . . .	17
4.1	Architecture of the system to be developed. . . . .	30
4.2	Template for the reports generated. . . . .	35
4.3	Example of a generated report for KahUC team. In Requirement 4, the code names BUS, REQ and DEV refer to the separate development stages of business, requirements elicitation and development, respectively. . . . .	36
4.4	Number of commits in relation to the time of execution. . . . .	39
5.1	Architecture of the final system. . . . .	45
6.1	Simplified version of the final model for Quirked Up Software. . . . .	50
6.2	Simplified version of the final model for KahUC. . . . .	51
6.3	Simplified version of the final model for byDeiCrowd. . . . .	53
A.1	Gantt Chart from the 1st Semester (from September 2022 to January 2023) - The blue activities are related to research activities, the brown ones are related to development activities, and the bordeaux ones are related to the writing of this thesis . . . . .	68
A.2	Gantt Chart from the 2nd Semester (from the end of January 2023 to July 2023) - The blue activities are related to research activities, the brown ones are related to development activities, and the bordeaux ones are related to the writing of this thesis . . . . .	69
D.1	Introduction of the forms and request of the students' username . . . . .	81
D.2	Questions regarding the stage of development and the class they belonged to . . . . .	82
D.3	Display of the results of the Quizipedia team . . . . .	83
D.4	A space for some additional comments from the participant . . . . .	84
D.5	Final Note . . . . .	84
F.1	Real process modeled for Quirked Up Software . . . . .	90
F.2	Real process modeled for KahUC . . . . .	91
F.3	Real process modeled for byDeiCrowd . . . . .	92



# List of Tables

4.1	User Story 2 - See raw information. . . . .	26
4.2	User Story 3 - Mapping objects. . . . .	27
4.3	User Story 14 - Task Monitoring. . . . .	27
4.4	Public repositories execution in seconds. . . . .	38
6.1	Number of <i>Housekeeping</i> activities compared with the total number of events. . . . .	54
B.1	User Story 1 - Multiple Repositories . . . . .	71
B.2	User Story 2 - See raw information . . . . .	71
B.3	User Story 3 - Mapping objects . . . . .	72
B.4	User Story 4 - Author's Development Stage . . . . .	72
B.5	User Story 5 - Files Per Author . . . . .	72
B.6	User Story 6 - Authors Per File . . . . .	73
B.7	User Story 7 - New Sprint . . . . .	73
B.8	User Story 8 - Critical Files . . . . .	73
B.9	User Story 9 - Active Members . . . . .	74
B.10	User Story 10 - Files with more Authors . . . . .	74
B.11	User Story 11 - Files with only one author . . . . .	74
B.12	User Story 12 - More Commits . . . . .	75
B.13	User Story 13 - Less Commits . . . . .	75
B.14	User Story 14 - Task Monitoring . . . . .	75
B.15	User Story 15 - Completed Issues . . . . .	76
B.16	User Story 16 - Daily Commit . . . . .	76
B.17	User Story 17 - Issues with no Assignee . . . . .	76
B.18	User Story 18 - Development Type . . . . .	77
B.19	User Story 19 - Commits Per User . . . . .	77





# Acronyms

**API** Application Programming Interface.

**BPMN** Business Process Modeling Notation.

**CLI** Command Line Interface.

**FR** Functional Requirement.

**IS** Information System.

**IT** Information Technology.

**MSR** Mining Software Repositories.

**NFR** Non-Functional Requirement.

**PMSR** Process Mining Software Repositories.

**RQ** Research Question.

**SE** Software Engineering.

**US** User Story.



# Chapter 1

## Introduction

Since the beginning of this century, process mining has been a growing area in Information Technology (IT). Much research has already been done in this area, and the number of applications continues to grow.

In this thesis, we will focus on the application of process mining to software repositories. By extracting the data in a single software repository, after a further examination of the artefacts, the different activities taking place in the project, and how or when each member contributed, we will try to model the process adopted by the software development team.

In this chapter, we will present the problem that will be addressed during this research thesis. First, there's a simple introduction to give the reader some background and some context of the area of process mining. Then, we will elaborate further on what we are trying to accomplish and on what our action plan is to reach that goal.

Finally, at the end of this chapter, we present an overview of how this thesis is structured.

### 1.1 Background and Context

Process mining is still a young topic with a lot of potential for research as well as for helping businesses obtain some feedback regarding their work processes. The challenges posed by process mining, and how to apply it, have been growing as time goes on and technology and working practices advance.

Process mining was already successfully used in real-life companies within a vast span of sectors, such as healthcare [22] [1], auditing [14] [13], logistics [1] and manufacturing [15], among many others.

The goal of process mining is to be able to model the process of a company or a team by gathering and analysing a set of logs in the history of the work. On the other hand, while this has already succeeded in the examples given previously, we will try to tackle one of the next big challenges in this area: applying process

mining to software development.

Given the complexity and abstraction levels that software engineering involves, this poses a big challenge. Until now, there were many libraries and frameworks already created to support this kind of analysis. However, all the breakthroughs made to this end were less significant than what would be expected, but nevertheless a step towards the final solution. Some of these will be mentioned throughout this thesis and will be used as a reference in the development of our solution.

Last but not least, a promising application of process mining may also be found in fields requiring the certification of software, namely aerospace and health-care.

## 1.2 What we are trying to solve

Our main challenge during this research will be not only how to apply process mining techniques to software repositories, but also how to map these into modelled processes. These specific kinds of projects are incredibly complex by nature, not only because of the number of variables to be considered but also on how we can trace them down during a period to see every step and every stage it went through.

The primary goal is to create a system that can reach these models with no human involvement, using process mining exclusively on the event log data that has been extracted from the software repository (using the GitLab and GitHub Application Programming Interface (API)'s). Furthermore, the biggest issue to be dealt with is regarding the given models' accuracy to the reality of what happened, or if they can be improved even more.

## 1.3 How we are going to solve it

Given the previous problem statement, now all there's left for us to do is define an action plan towards a solution to solve our problem.

To be able to map these processes, we are going to develop a system that, by using the chosen online repository API <sup>1</sup> to collect the event logs from the software development project, uses a rule engine and process mining methods to identify the separate activities that took place in the repository up to that moment, and by using this input extract the underlying work processes.

As it is explained in section 5.2, after we get the repository for the project we want to analyze, we are planning on filtering all the data and all the commits to get a list of events. We will consider an event (see section 3.1.1) as a single action that is performed by a member of the project, as simple as the action can be. These

---

<sup>1</sup>As will be seen, even though we only interact with GitLab, our solution can be easily adapted to other types of repositories.

events will be the base structure of our system since everything will be based on that list. Later, we plan to classify these events according to their activity type (see sections 3.1.1, 5.2.4 and Appendix E) to use it as an input to model the process.

As a starting point, we will focus on the premise that every single project we'll be working with, and the ones that will be our primary goal to uncover, are software development projects. By doing this alone we are greatly reducing the scope of projects instead of considering every single repository that may be found. Given their complexity, we can state that software processes differ from more generic ones where a simple log set can directly give us a modelled process.

Given this, we can now use Software Engineering (SE) principles such as standard processes (waterfall, agile, ...) and types of activities (bug fixing, new requirements, new features, ...) as guidelines to know what to look for when analysing the repository logs. SE focuses on the development of software with all the underlying stages it requires like requirements elicitation, architecture design, development, testing, and maintenance, among others. By using all these concepts, we know what to look for and how to interpret the collected logs to model the desired process.

These methods were applied to the creation of our system's alpha version, in the first part of our thesis, which is detailed in Chapter 4. This version does not have all the requirements implemented, a visual interface built (the only interaction the system requires is done through the Command Line Interface (CLI)), or the complexity considered for our preferred solution, as it is mentioned below. For the purpose of testing, verification and enhancement of this early version, we used six distinct groups of college students and their projects for a class of the last year in the Bachelor's Degree in Informatics Engineering at the University of Coimbra.

Further ahead, along with SE techniques, we also used a set of engine rules to help us decide what kind of activities we will be looking for in the data. That way, we can easily specify how we want to define each activity, and which rules an event must obey in order to be categorized as a certain activity.

In the end, the final version of the system resulted in the merge between some concepts already used in the previous system, a rule engine that was created so that we could retrieve the activities from the events, and process mining algorithms that helped us turn those same activities into a modelled process. The testing of this final solution was applied to the projects of the same groups of students already mentioned above.

Along with the development of our solution, we are also predicting some unexpected drawbacks. One of those is related to the quality of the logs retrieved [6] and that will be used in the analysis. Another challenging problem concerns the labelling of the logs, activities or even artefacts that were retrieved from the project being studied. If these are ambiguous or don't follow some clear format stating what is their end, then some human involvement may be required. All the issues that were encountered were documented in sections 3.3, 6.3 and 4.7.

## 1.4 Structure of the thesis

As a way to ease the reader's view into this thesis, following we present its structure and what may be found in each chapter.

In Chapter 2: State of the Art, besides an overview of some basic concepts of process mining to get the reader more acquainted with this area of study and some terminology that may be used during the report, we will also narrow it down further to the main focus of this work by introducing Process Mining Software Repositories (PMSR).

In Chapter 3: Methodology, we present an extended description of how we are planning to address the Research Question (RQ)s proposed. We present a plan according to the timeline given for this research, the roles of the users to which our solution may benefit, the testing that will take place and a brief overview of some of the main frameworks and API's that will be used. In the end, we also present a risk analysis.

In Chapter 4: Alpha-version, we present the development and discussion of the results of our initial system. We propose a set of requirements identified for our system as well as its constraints, how each one of these will affect the system, and for what purpose we came up with them. The preliminary architecture of our system is presented as well with a brief description along with a risk analysis. Furthermore, we're also going to make a description of the approach we took to develop this version of our system and some other auxiliary scripts that were needed to help validate the results. Finally, we will mention some of the most relevant problems we encountered during that process and explain the techniques used to validate our current system and the results that we obtained.

In Chapter 5: Validation, we present the complete solution developed to help us answer the RQs. First, we introduce a small *Playground* script developed to help us come up with the best architecture for our system. Taking that into account, we propose an architecture of the type pipe and filter and describe every interface that will be needed.

Finally, in Chapter 6: Discussion of Results, we present the testing made for our final solution as well as a discussion and analysis of the results, as well as some known threats towards the validation of these results.

In the last chapter (Chapter 7: Conclusion) there will be a small overview of the thesis by singling out the key ideas. We are also going to propose some future work in case this research is given a follow-up.

Lastly, in the end, the reader may find some useful appendices that complement some of the information and data in this thesis.

# Chapter 2

## State of the Art

In this chapter, a state of the art will introduce the reader to the research area of process mining in section 2.1, the different methods and techniques used, and how it benefits the companies or projects that proceed to make an analysis of this kind.

Afterwards, the Process Mining Software Repositories (PMSR) area will be presented since it is the area of process mining where this research will be focused on.

### 2.1 Process Mining

Since the beginning of the century, there has been a noticeable increase in Information Technology (IT) and in data that is either exchanged between two or more entities or that gets stored in information systems. Inevitably, this growth called for new techniques to handle data, analyze it, and store it. It also allows us to discover all the possibilities this new amount of data gives companies.

At a certain point, both process science and data science had to cooperate with each other to deal with the common issue of increasing data collected by IT systems and not knowing what to do with it. After some experimenting and research, the joining of these two sciences was named "process mining," which followed the development of several algorithms and tools.

Given this brief introduction, process mining is based on the extraction of information regarding the processes of a team or company. Usually, this information is in a structured log with a given timestamp. We can obtain the workflow of a given team or company by combining all of the logs. Given this workflow, the project manager is able to see clearly how the team is working, at which stage a certain activity is, or if there is any deviation from the process.

By using process mining techniques, one can easily monitor a process in order to improve it by identifying trends and patterns and by having some more detailed information and insights on the workflow. All this information can be proven to be very useful when managing a team and inquiring about its effec-

tiveness, discovering bottlenecks, and improving the overall work being done by the team - the software being developed.

Even though it's outside our scope, it's worth mentioning that with the increase in research, there has also been an increase in the different possible ways to conduct a process mining analysis. As an example, there is already a very large range of algorithms to choose from, each improving the previous one or better adapting it to a different context. One of the first algorithms to be developed was the  *$\alpha$ -algorithm* [12] that follows each step required in process mining. Even though this algorithm is barely used nowadays because of all the flaws that were later detected, it served as a base for updated versions of itself and also helped investigators know what to avoid in new algorithms, such as the *Heuristic Miner*, *Genetic Algorithm*, *Language Based Region Algorithms* and *State Discovery Algorithms* [31] [3].

### 2.1.1 Types of Process Mining

There are many ways one can make a process mining analysis. The main goal in any kind of process mining is the same: to get a reliable model that clearly reflects the process that took place.

However, for this to work, we need an ordered, correct, and reliable set of logs. Otherwise, the model's quality may suffer and it will become even further removed from reality. The issue of log quality is likely one of the most difficult aspects of process mining.

Bose et al. address this issue in [6] by naming several parameters that may influence a log's quality, such as *the event granularity* (whether the logs have a high or low level of detail or if that level changes between logs from the same set), *the timestamp of each log* (even though the timestamp is a key factor for process mining, it's probably the hardest feature to get accurately according to reality, no matter if it has a delay of minutes, hours, or even days), the possibility of having *missing data* that wasn't logged in the system (if we have missing data, then we have a possible link missing between two events, which may mean a loophole in the model), *the ambiguity of events* (for example, when a log message or a log description is too ambiguous and doesn't tell us what happened in that log), or *the presence of noisy data* (which may disrupt the normal proceedings when modelling).

If any of these is detected, then they have to be corrected manually, which would imply human interference with the log, which is what process mining tries to avoid.

After the logs are cleaned up, we can go on and proceed with the analysis. This can be done by either using process discovery, conformance checking, or process improvement [2], as will be seen below.



## Process Discovery

**Process Discovery** is the key activity of process mining: it takes a set of time-stamped events or logs and constructs a model based on them so that every time one of the logs is given as an input to that model, it runs without problem. This exercise may result in several different models, but the idea is that they are *discovered* based on real event data instead of assumptions about what they should be. Given the discovered models, they can either be compared to find the discrepancies and see how they affect each other and the accepted logs, or new logs can be used to see if they are accepted or not.

One important aspect of process mining is that it is dynamic. This process analysis is done not when the whole process has ended but when the number of logs is considered large enough to start. That way, when an analysis is complete, there are new unknown logs to either confirm or refute the previous analysis, in which case we remodel the process again with the new logs.

These models can be represented in many different ways. In business process management, the most common way to model processes is using Business Process Modeling Notation (BPMN) diagrams, on which you can see a process represented by its different tasks and events, what happens in the process between each task, and the choices made during the execution. Another typical modelling approach is using Petri nets [3] [33] where a set of tasks (transitions) are connected to places by arcs and, with the use of tokens, they describe the behaviour of a model. The following example in Figure 2.1 describes a Petri Net of an Agile Workflow Model.

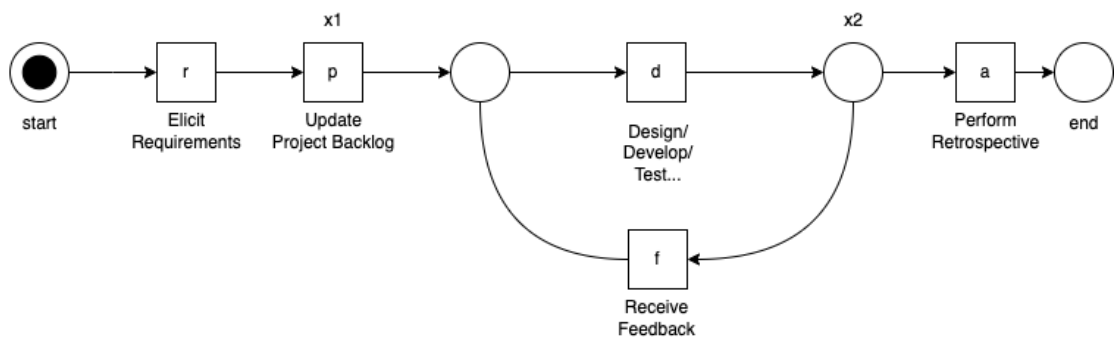


Figure 2.1: Petri net of an agile workflow process.

## Conformance Checking

Following is **conformance Checking**. The starting point for this method can either be a model that was discovered using process discovery - where the model is built based on the event logs taken from the process and from data given previously - this model can also be purely theoretical - when it is built beforehand given what it is expected to be. Then, the logs are extracted, the real process is modelled, and both models can be compared in order to detect any deviation from the theoretical model in the real one.

For instance, if a restaurant wants to know if the food is sent correctly in their takeaway service, they could model the entire process from the moment the request is registered to the instant the food is delivered to the customer. After the given model is complete, it gets a set of logged requests and runs them to see if there is any problem with its execution. If the log is accepted by the model, then the request was handled as predicted. If it's not accepted, then that specific case deviates from the common procedures and should be further analyzed.

Conformance checking is done by evaluating four different aspects of a process [3]:

- The *fitness* of a model is related to the acceptance rate of event logs by that same model, that is, the ability of the model to analyze a sequence of real events. If it succeeds when executing with every single log taken as input, then it has perfect fitness (1). The model's fitness decreases as the number of logs it fails to process increases; in the extreme case of not accepting any log, the model's fitness is 0;
- The model should be as *simple* as possible when explaining the behaviour of the process. This can be monitored by several metrics. If the process is modelled using a graph, then the number of nodes or links in the graph can be considered a measure of simplicity. The model's simplicity is important when there is a tie between models: if both are accepted, the chosen model will be the simplest.
- A *precise* model doesn't give much space for logs to behave in many different ways, it should follow a rule or a specific path and not allow many outliers or big changes in the logs to occur.
  - A model that is lacking in precision can be said to be a *underfitting* model, that is, a model that generalizes too much and allows for events that are very different from what is expected. A *daisy model* is the most underfit model that is feasible because every single example begins and ends at the same location (and therefore each case draws the petals of the daisy into its center)
- *Generalization*, unlike precision, gives the model the flexibility to generalize the case of study and accept some exceptions.
  - If a generalization isn't done correctly, the model can end up *overfitting*: when the model is so specific and detailed that only allows for some logs to run correctly, that is, when it doesn't support acceptable deviations from the real world.

Taking into account all these parameters and trying to get a perfect score on them is not an easy task, especially when it comes to both precision and generalization. However, just like Van der Aalst states in [2], even if some compromises are made between these aspects, the ideal scenario is to have a good conformance checking, meaning a *simple* model with *good fitness* and *good precision* for the case at study with *enough generalization*.

## Process Improvement

The last type of process mining is **Process Improvement** or Process Enhancement. Conformance checking is about analyzing the logs given and comparing them with a model that is assumed to be how the process works. However, there may be cases where there are traces of new behavior in the model and, after some careful analysis, it's shown that the theoretical model is outdated or that these new changes are justified by their context. Enhancement solves these cases by adapting and adjusting the model with the given logs that show a new pattern in the process.

For that purpose, the data recorded in each log can be used as a tool to get new information about how a process is being conducted, what the deviations are, what changed since the last model, and why it is malfunctioning. In the end, this inspection can either mean the *repair* of the current model (modifying the current model so that it's more accurate to reality) or the *extension* of the model (adding new paths and different views).

### 2.1.2 In Practice

Given all the technicalities, there is still one important question to be answered: *How can we use process mining in a real-life situation?*

Let's consider the following example from a study made by Wil van der Aalst [1] where he models and analyzes a Dutch hospital. These logs represented the modeling of 114.592 events, 619 different activities, and 266 separate individuals. In the end, after applying the heuristic miner algorithm, the output was a model like the one shown in Figure 2.2<sup>1</sup>.

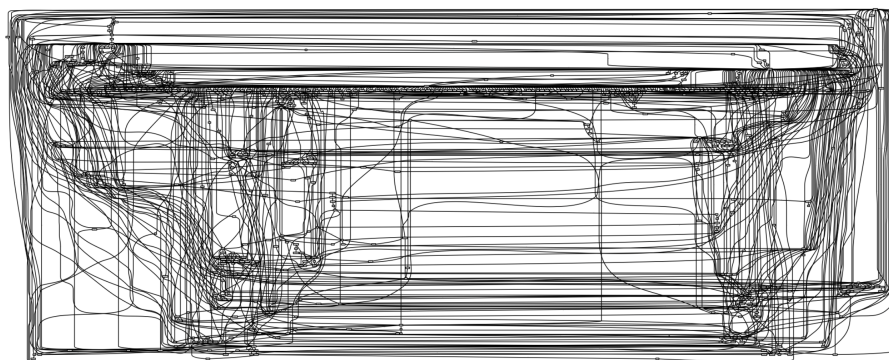


Figure 2.2: Spaghetti Process of a Dutch Hospital (from [1]).

Models such as this one, with a low structure and great disarray between activities, are informally called **Spaghetti processes**. Thanks to process mining,

---

<sup>1</sup>Even though the details of the figure are not readable, the only important remark is the visual aspect of the process and the fact that it is seen as a set of different layers organized between each other.

even though a company or workplace seems to have no issues, we can find these spaghetti processes and study them to see how they can improve, even if only slightly. One way to improve these processes is with process discovery, already mentioned in 2.1.1, by comparing the different output processes with the one in question.

However, not every process has to be a spaghetti process at first. Considering another example, taken from [1], this one analyzes how a Dutch municipality workflow handled 528 requests that generated 5498 events. With the help of ProM 5.2, one of the most advanced process mining tools to this day, the result is the model shown in Figure 2.3 <sup>1</sup>.

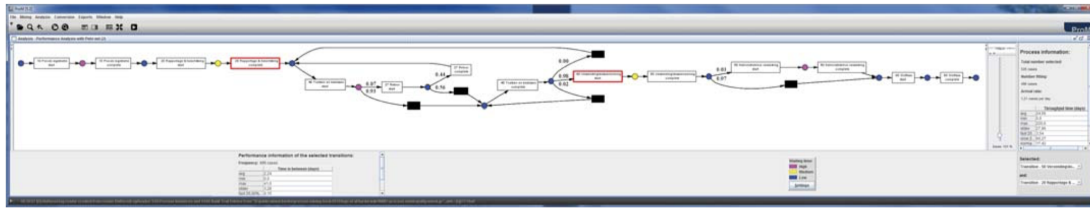


Figure 2.3: Lasagna Process of a Dutch Municipality (from [1]).

Unlike the model shown previously, this one is structured and follows a clear, clean path where we can see the decision points, the beginning, and the end. This is what is informally called a **Lasagna Process**. Of course, speaking from a process analyst's perspective, these are hardly interesting since they are already figured out and structured with no problems, and so they make very little room for improvement. However, this can be easier to analyze, and if there is any point of conflict, it's easier to find it and think of another way around it.

Over the years, process mining techniques, research, and even real-life application have increased greatly. For example, in [13], Hosseinpour and Jans suggest a method to apply process mining to help with financial auditors. Using conformance checking techniques, they propose an algorithm that, when comparing an event with the theoretical model, categorizes the possible deviations as well as the frequency at which they occur. That way, they can help auditors identify relevant deviations from the model through more context-based deviations.

Another similar example can be seen in [33], where is proposed a theoretical system that, when connected to an information system, continuously audits and checks for any business rules violations in an organization through their theoretical and discovered modelled processes.

In conclusion, there is much these models can uncover from what is happening in companies. Nevertheless, there is much more that process mining can be used for. Looking more closely into IT companies or departments, one of the next big steps in process mining is to unravel what is happening during the development of a project, what kind of methodology they are using, and if it is being used correctly.

## 2.2 Process Mining Software Repositories

Given the basics of process mining, we will now talk about the core of this thesis: PMSR. First, we will present an overview of Mining Software Repositories (MSR), and then be given examples of how we can use this approach.

### 2.2.1 Mining Software Repositories

When we talk about MSR, we are narrowing down our scope to software projects stored in repositories, open-sourced or not. Just like in process mining, with the amount of data available in repositories, there was a need to find new ways to recover, analyze, and interpret that data in a way that could benefit the team in the future.

A *software repository* is where all the data, all the files, and even some configurations are stored in a way that allows for version management, parallel coding, the management of the whole project, and even helps to take a look at the different tasks at hand. Some of the most common repository services available online, such as GitLab (3.2.4), GitHub (3.2.4) or BitBucket, also allow for the repository to be open-source so that anyone can see it and collaborate.

Even though there are a lot of similarities between process mining and MSR, the latter uses *data mining* techniques to go through the data at hand and get some powerful insights.

With all the data retrieved from the repository since the first commit down to the last one, MSR can take advantage of several techniques from process mining, data mining, and even machine learning to be able to extract some useful information about the way the team is working or even how the product is being developed - including some fatal mistakes that, on the other hand, could not be caught.

### 2.2.2 Mining Software Repositories into Processes

As mentioned in Chapter 1, in this research we are not only going to mine data from process repositories but also try to map them into processes.

One of the primary motives for process mining to transform process data into a discovered process model is not only to check if a model is well done or if it's being followed by the team: it's all about *having* a model at all. Many companies or teams that work with software repositories don't draw an initial model that needs to be followed by the team as it should be, instead, they simply assign tasks with deadlines and expect to have the work done.

On the other hand, there's a typical example of when a shortcut is taken more often than the real path. By modelling the processes stored in repositories, process mining is also able to detect these deviations in the processes that are being

followed by the team. Of course, this works mostly in cases where there is already a model *a priori*, but even if there isn't, the team leader can identify some critical behaviour that should not exist. After this discovery, process mining techniques can also be used to improve the model, either by restoring it to what is expected of it or by trying new and more promising alternatives.

### 2.2.3 PMSR Usage Examples

The smaller a company or a team, the easier it is to predict failures in the development of a product and the easier it is to prevent them. When we talk about large organizations, sometimes the processes are so complex and intricate that a problem is only detected after it has already happened, making it harder to fix it. Process mining prevents this by analyzing logs and obtaining a *real* model of the process from which we can detect problems such as bottlenecks.

In the study presented in [26], the authors use process mining techniques to extract hidden information from a big company's process logs. They present all the steps towards their goal: the definition of a theoretical process model, the data preparation to make sure it's arranged in a standard way, the analysis, and the conclusions drawn from it. After extracting all the data, they were able to not only model the real process but also draw a social network showing the flow of information in that same company and analyze the performance of the model by getting some related statistics. With all this data, they concluded, among other things, that there were some differences between the real and theoretical processes previously modeled and that there were tasks that could be optimized but didn't add any value for the company.

Another approach taken by Santos et al. in [25] was the development of a technique that tells the user which activities are candidates to be tailored according to what is expected. By having both the theoretical and the real model of the process, they were able to use process mining techniques in order to identify which activities were either missing from the real one or were added to it. After that analysis, they leave it to the project manager to look further into it and decide how to best tailor the model according to the company's needs.

In [24], Rubin et al. present to us two different use cases where process mining was used to improve two different reservation systems in the tourism sector. In both use cases, the authors were able to detect performance issues, monitor the success cases, identify the actions performed with greater frequency, and, above all, identify the failure patterns and how frequently they happened. With this insight, they proceeded to write down more accurate test cases for testers and developers to improve the system.

Another relevant piece of literature is [5], where they focus on developing a technique to monitor the progress of a project and discover which activity is being done at the moment. To this end, they focus on the artifact's name and path to find any pattern related to what kind of activity they are involved in (if

it's documentation, configuration, building, etc.). In the end, they give a few real examples using their technique in practice by identifying in what kind of activity each file was modified on which date, they were able to distinguish if the team was following a *Waterfall Model* or if they were using *Agile Methods*.

## 2.3 Summary

This chapter begins by giving an overview of process mining. This area emerged to answer the need to handle the amount of data that started increasing. This may be done by either **process discovery**, where we use directly the retrieved logs to model the process; by **conformance checking**, where we compare two different models to see the deviations that may be occurring; or even by **process improvement**, where we try to enhance the modelled process according to some deviations presented in the logs. Along with these three techniques, we also provided several examples of successful process mining applications in many sectors, including healthcare, logistics, and auditing.

After that, we move on to PMSR, the main scope of our work. By retrieving data from software repositories, we can mine them with the help of process mining and machine learning techniques in order to get more detail about the work team. Afterwards, we introduce some examples of research and advances already made into PMSR.

In the following chapter, we will make an overview of some methods and the design of our thesis and how we plan to proceed.





# Chapter 3

## Methodology

In this chapter, we are going through the methods and the design of the experience with all the steps taken during this thesis.

First of all, we'll try to better elaborate on our Research Question (RQ)s so that we can be sure of what we are looking for exactly in the development of this experience. We will also specify some important definitions that need to be clear throughout this thesis for a more reasonable understanding of the results.

After that, we will go over the study that we designed in order to reach a conclusion and an answer to the RQs posed in the context of this thesis. We'll take a closer look into the planning of the study and each step taken, as well as go over the tests designed and the tools we intend to use.

At the end of the chapter we plan to present a risk analysis for our system.

### 3.1 Research Questions

Research Questions explicitly ask what we are looking for in a thesis. If they can be answered positively or if their answer can provide us with new information that we were looking for, then we may conclude that our thesis is correct.

As mentioned before at the beginning of this section, we are trying to find out if it is possible to find the workflow of a team given only the interactions they had with their respective repository.

Since there are various abstract ideas and phrases that we are dealing with in the context of our thesis, we find the need to try to find a definition that suits our purposes. For that purpose, before we move on to our RQs, we believe it's essential to find an accurate definition for a *repository*.

This term is already familiar to anyone that works in Information Technology (IT) or Software Engineering (SE), and therefore a set of ideas may come to mind, such as files stored in a cloud among a team, or even versioning control systems that ease the work between team members working in the same file.

Nonetheless, let's take a look at the next example <sup>1</sup>. For someone that uses GitLab, besides the file management system and the versioning control already mentioned, there's also the possibility to assign tickets <sup>2</sup> to each member, to see the different deployments, or even write different testing scripts in order to see if the code added in a commit fails or is working successfully (among others).

Following the same line of thought, if we look into GitHub, it also has some features found in GitLab such as a file management system and tickets <sup>2</sup>. However, if we were to commit new code and test the pipeline to see if there are any problems, GitHub only allows it if we link the repository to a third-party app, whilst GitLab has it already pre-configured [32]. As a matter of fact, it may also happen that users of either GitHub or GitLab also take advantage of other third-party apps, either connected or not to their Git repositories, for multiple possible reasons.

Given this example, if both GitLab and GitHub store repositories, but have different functionalities available, *what exactly is a repository then?* Since in our thesis the repositories of each group are the most important source of data from which we will take the conclusions, we consider a *Repository* the set of systems or apps that the team in question uses to store the project source-code and perform version management amongst themselves.

Since repositories are at the centre of our work, it is important to keep this idea in mind from here onward.

### 3.1.1 RQ1: Is it possible to turn raw events into activities?

Before we jump into this RQ, we need to ask ourselves what are events, what are activities, and most importantly, what is the difference between both of them. As we see it, for the interest of our thesis, we will settle that:

- **Events** can be defined as every single action that is performed in the repository. These can be the modification of files, a commit, or even a new ticket. When a commit has more than one modified file, then we can say that more than one event occurred.
- **Activities** are the type of event that took place. These may differ according to the group that is being tested and the type of activities that they consider. For example, if a file with the extension *.tst* is modified in a event, then it corresponds to a testing activity.

After this clarification, our initial question is rather straightforward: after we retrieve the events from the data of the repository, is it possible to automatically

---

<sup>1</sup>In this example we will talk about two apps (GitLab and GitHub) that work on top of the Git system - one of the biggest and most well-known references for online repositories.

<sup>2</sup>Even though the word used in GitLab and GitHub is "*issues*", we will settle with the term "*tickets*" as a standard word in our thesis for this type of functionality.

infer which type of activity took place? This is one of our core challenges that, if answered positively, will then bring us halfway to our final RQ.

We may<sup>3</sup> infer that the answer to this question is positive if, after turning the data collected from the repository into events, we can automatically map these same events into a specific type of activity according to the rules given to the system (see more about the rules in section 5.2.4 and appendix E). For a better understanding, the flow of the data described is represented in the figure below.

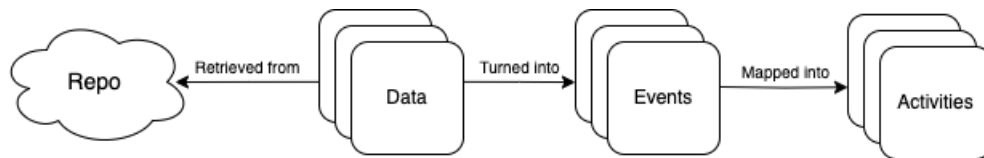


Figure 3.1: Flow of data thought for our system.

Keep in mind that a positive answer to this RQ is necessary for the thesis to be able to proceed and for the second RQ to be taken into account. Otherwise, we may say that this study is unfeasible.

### 3.1.2 RQ2: Is it possible to retrieve processes from a set of activities?

This question is the one that sums up our main goal for the thesis: given a group of events retrieved from a repository, when turned into activities (according to the findings from section 3.1.1), is it possible to get the process of the workflow followed by the team?

If the previous question presented in section 3.1.1 (*Is it possible to turn raw events into activities?*) verifies, then our goal for this question is to use those activities as references to model the process of the repository and see the traces and paths taken by the team between each activity, as well as each frequency.

We may<sup>3</sup> infer that the answer to this question is positive if, given the activities extracted from the previous RQ in 3.1.1, we can automatically trace the workflow of the process that was followed by the team.

## 3.2 Design of the study

Before we delve into how we designed the experience to prove this thesis, it's worth noting that we will be working on an *empirical software research* [20] [35] [34].

As the reader may have already noticed from the Introduction in Chapter 1, our research is empirical since it is based on real-life events and our conclusions

---

<sup>3</sup> Given the threats to validity presented in section 6.3, we cannot safely say that the answer is completely accurate.

will be based on the results of several empirical tests. Our work is related to and directly concerns specific real-life cases regarding software development.

Regarding the target audience, we will consider all software developers that work using software repositories. Even though this is a research thesis and we don't intend to turn our system into a product, it's always useful to imagine if our results and our means to them could be of any use, and to who.

As previously mentioned, even though our main testers were students in an academic scope, given the software development's complexity, to have a perfect system, we would need to develop one solution for each case, which is not doable. However, our system is thought to retrieve software repositories and model their processes at the moment, so that the team can see the bigger picture and look for possible conflicting areas. As such, despite some minor modifications that it may (or may not) require, we plan to develop our system as flexible as possible in order to try to cover all the most known cases so that it could be applied to every type of development scope.

### 3.2.1 Planning

Since at the beginning of this thesis' work process mining was still an unknown area for us, the first step was to spend some time doing some research, learning new tools that could be useful, and getting acquainted with this scope of IT. This way, we have become better prepared to tackle the problem at hand and design a strategic plan on how to proceed next.

Also because process mining was new to us, we thought it best to start our thesis with a simple program (the alpha-version described in Chapter 4) that simply retrieved the data from an online repository and analyzed the data according to some requirements established. These requirements were thought to take into account some parameters and types of information that may interest someone who wants to know how the project is going and if there are any abnormal results. Some of these may be which team members worked in a specific file, the number of commits that each user made, or even if there is any ticket with no one assigned.

After this version was complete, we developed two small scripts to aid us in the validation of the system. The first one was used to generate a report to help us see the results in a more user-friendly way. The second script was to help us find multiple users in a repository. There were many cases of users that during the project changed their usernames, which caused past commits to have the author with the previous name, and more recent commits to have the new name. As such, we found the need to find these multiple names to have more accurate results. Even though our script couldn't resolve all the cases and may need some human help to decipher the final ones, a good amount of names were mapped in the end.

With all the data gathered, all that was left to do to finish this first milestone of our thesis was to validate the results we got with the students that had been

working on the repository (more insights about the testing with this team of students in sections 3.2.3 and 4.8). To that end, we created a form with Google Forms with some of the results and some of our conclusions to see their responses, if they agree or not and how accurate the results are.

The development of this first system was a great opportunity for us to study the data that could be retrieved from Git repositories, as well as leaving us with the idea of what to look for and how to proceed in the next phase.

At the start of the second iteration, our first step was to determine which architecture would be best for our solution, and how to enhance the one we already had. For that reason, we developed a small script named *Playground* where we sorted and displayed each event that took place in a certain repository. This allowed us to add all the interfaces needed to our architecture.

Afterwards, all that was left for us to do was start implementing each interface needed to get the results we needed. Eventually, there was some parallel writing and research needed to be done, especially when we started to implement the rules engine, which called for an increased effort in studying them and trying to understand them.

Even though the rules engine was the most challenging task, it also helped us to validate our first RQ, which allowed us to quickly move on to the final interface to model the process.

When the development was done, before we started writing this final report of the thesis, we still went through a period of refactoring of the code to improve the rules applied to the data as well as the system altogether. After this, we tested our solution on three of the groups of students already mentioned above: Quirked Up Software, KahUC and byDeiCrowd.

Both iterations of the thesis may be seen as a Gantt Diagram in appendix A for better visualization.

### 3.2.2 User Roles

Further on, in case our system ends up being used as a final product (or is incorporated into one), it's important to understand who are the users that would be using it and what roles they could play out.

All the interactions with the system are thought to be through a simple dashboard. Even though the development of this dashboard is out of the scope of this thesis, it's good to remember the possibilities and the likely interactions that would take place.

### **Admin**

An *Admin* is a user that, besides viewing the simple display that will be shown with all the data and statistics gathered until that moment, is also granted access to change the configurations of the system, as well as the activities which the events are categorized.

As mentioned before, software development is so customized, unique for each project and out of the ordinary that there isn't an equal set of software projects. Either the organization among the team is done differently, the systems used are different, or even the rules to know when to accept an event may change (further regarding the rules in section 5.2.4 and Appendix E).

Because of all these specifications, we are going to need an *Admin* that is able to change the configurations of the system in order for it to be tailored to what is asked for in that specific project.

### **Team members**

A *team member* is any person who belongs to the project that is not an *Admin*. That is, every person working on the project will have access to view the data collected and the statistics shown there, but will not be allowed to change any settings.

### **3.2.3 Testing**

Besides the small debugging tests made during the development of the system, there were two crucial moments of validation during this thesis. These tests were made not only to make sure our system agrees with the purposed requirements but also to validate or disprove our proposed thesis, that is, our RQs.

As already mentioned in section 1.3, to test our systems we used six distinct groups of students from the Bachelor's Degree in Informatics Engineering at the University of Coimbra that were taking the Software Engineering class (from September 2022 until January 2023). For this class, they had to develop a system according to the requirements and needs of a real client following the agile model for software development. These groups are Quizipedia, KahUC, byDeiCrowd, QuizzES, Quirked Up Software and App.

Since these students are still in a learning environment and feel the need to comply with SE practices, we figured that they matched the requirements and the type of Repository we were looking for to test our system.

### **$\alpha$ -version**

After running our system with the data retrieved from the repositories of each group, we collected and analysed the output data. In order to make sure our

results are according to reality, we created an online Google Forms (3.2.4) that may be seen in Appendix D with a series of questions for each student to answer and to give his own opinion regarding the results shown.

The results of this test shall be presented and discussed in section 4.8.

## Final System

For our final system, unfortunately, there was no direct feedback we could rely on as we did before. As explained, the projects we used for testing were made in the scope of a college class that lasted during the first semester (from September until January). Since our final version was only developed during the second semester, there was no possibility for us to get into contact with the students responsible.

As such, our strategy was to test the reaction of our program, specifically our rules, to new repositories. We chose the three groups of students that seemed to have the most relevant results to support our thesis (see more regarding the process of selection in 6.1). The results and discussion shall be presented and discussed in Chapter 6.

### 3.2.4 Tools

In this section, as mentioned before, the reader will be presented with an overview of the tools that were either used during this research or taken into account.

#### Camunda

As will be mentioned in the following chapters, the architecture (5.2) that was thought for our system will require a rules engine system to go through the events with the data extracted. After researching the available research systems, one possibility that came to us was Camunda<sup>4</sup>.

Camunda allows us to design, monitor and improve processes from organizations. By using models designed with Business Process Modeling Notation (BPMN), it allows us to create *rule tables* that will read our input data and give the output according to the rule(s) that match.

Even though we successfully implemented a small program to test the rule tables through Camunda, we concluded that Camunda wasn't the best option for our system. Since Camunda uses Java to run, the amount of data we were expecting to be running would cause a continuous overhead of the system while testing each rule. Therefore, since this is only a research thesis and it's not expected to be commercialized, the effort wasn't worth it.

---

<sup>4</sup><https://camunda.com>

### Git

Git [19] is probably one of the most known and most used software tools among developers. Not only it allows storing projects in online software repositories, but it also offers versioning control tools, branching of the project, the possibility to make our repository open-source and, in some of the available Git frameworks such as GitLab and GitHub, a ticketing system to manage project issues.

Even though Git is the main repository system with all the functionalities, there are many hosting services available to manage these project repositories. In this research, we will be working with two of them: GitLab and GitHub.

Even though GitHub [9] is more popular among developers and open-source code interested parties, GitLab [11][21] has recently been increasingly adopted by companies to store their software systems and monitor their development, as well as to aid in the integration of new features.

These services are the ones where, in this first stage of our research, we will fetch repositories to use for testing to validate our solution. By using their Application Programming Interface (API)'s we will be able to retrieve not only the commits and members of the project but also the list of all the issues created.

### Google Forms

In order to validate our results with the different teams mentioned in 1.3, we used Google Forms <sup>5</sup>.

This tool allows anyone to create an online survey, share it easily using a link or by sending it via email and see the insights of the answers according to different views.

### pm4py

Pm4py <sup>6</sup> is a Python library with many process mining algorithms already developed and ready to use in our own script.

It was the first process mining framework we looked into. Since its focus was only on the use of the algorithms and not on fetching data from outside systems like we needed to, we decided to use this library only in the final stages of our system. That way, we can exploit this tool to get access to some important process mining algorithms (like, for example, the  $\alpha$ -algorithm or the heuristic miner), to help us model the processes according to the input data.

---

<sup>5</sup><https://www.google.com/forms/about/>

<sup>6</sup><https://pm4py.fit.fraunhofer.de>



## ProM

ProM <sup>7</sup> is currently one of the most advanced open-source process mining tools. It's a framework that supports all the process mining algorithms and techniques to date and from them models the processes of the data used as input. ProM already has many different versions and is constantly updated.

Even though this tool was considered at the very beginning of this thesis as a possible external visual system, its complexity quickly made us discard it as an option. However, we think it's still a relevant tool to be mentioned.

## PyDriller

Pydriller [29][30] will be used in addition to the GitLab and GitHub API's. This library is particularly useful when mining software repositories since it allows us to retrieve commits from a given repository with all the kinds of data that come attached to them (such as modified files, authorship of the commit, and date of the commit, among others).

It's important to state that the latest version of PyDriller (v2.3) that was used in the system has some bugs that were discovered during development and were mentioned amongst other problems encountered during the validation of the first system (section 4.7). However, since there were some turnarounds to avoid and mitigate these problems, we decided to keep working with PyDriller and take advantage of its potential.

## pyscript

Pyscript <sup>8</sup> is a very recent framework that allows us to run Python scripts in HTML code. Since the user interface of our solution is out of the scope of this thesis, we are not going to use it. However, the connection pyscript establishes between Python and HTML seems to be an interesting tool for this future work.

## 3.3 Risk Analysis

Even though this is only research work, it's always important to make a risk assessment to make sure we're ready for all the potential risks that may appear unexpectedly. These may cause great harm and damage to what is being done, or they may even end up being beneficial to us. Either way, it's good practice to be ready for whatever may come up.

A risk assessment is important to be carried out now in the planning phase of our research to make sure we have the resources and the conditions needed for

---

<sup>7</sup><https://promtools.org>

<sup>8</sup><https://pyscript.net>

any event that may come. For this assessment, we will be using the guidelines from [18].

To begin with, one of the most obvious risks everyone takes when starting research work is that, in the end, the work is proven to be fruitless. That is, if, after all of our efforts to find a way to map software project repositories activities or events, we come to the conclusion that it is not currently possible. However, even such an outcome adds to the community's knowledge and might lead to further research while preventing other researchers from going into the same dead-end.

Another risk is the lack of skills or knowledge among the members of the team. Thankfully, with the existence of experts in the field, these gaps can be easily filled, harming the project only by delaying the amount of time needed to gain experience and enhance the skills needed.

In the eventual incident of a computer breaking down and leaving a member without a functional working device, the team is also prepared by saving all the work already done and all the artefacts produced in a GitLab repository and in the Overleaf<sup>9</sup> cloud so that they can be saved and accessed by any machine with an online connection.

### 3.4 Summary

In the present chapter, we start by detailing the RQs addressed in this thesis. First, we want to know if it's possible to turn raw events into activities with which we can categorize these events according to their purpose in the project. If answered positively, then we want to know if these same activities may be turned into process models by means of process mining techniques. That way, we would be able to see the workflow taken by a software team during the development of a system, and our thesis would be correct.

From there on, we present the way that our experiment was designed in order to prove our thesis. We introduce the planning that was made throughout this research, as well as the tests thought to validate our conclusions, and a brief overview of the main frameworks and tools that we will be using during our research, such as Git and PyDriller, as well as some of the tools that were considered.

At the end of the chapter we make a risk analysis.

In the following chapter, we will present the first system developed for this thesis. We will introduce some of the requirements identified, the preliminary architecture for our system and some of the constraints. Afterwards, we will see how this first system was developed along with the validation of these preliminary results.

---

<sup>9</sup>Overleaf (<https://www.overleaf.com/project>) is a cloud-based Latex editor that was used to produce all the documentation for this thesis such as this report.

# Chapter 4

## Preliminary Analysis

In this chapter the user will be given an overview of the alpha-version and alpha-validation that took place in the first half of this thesis.

At the beginning of the chapter, will be presented all the requirements to be implemented in this first system, that is, the functionalities and components that are expected (section 4.1), specifications that the system needs to comply with (section 4.2), and the constraints and limitations regarding the development and use of the system (section 4.3).

Furthermore, the architecture thought up for this system in order to materialize the requirements proposed (section 4.4) will also be presented.

Going back to section 1, we identify the main research problem we are trying to answer. By getting the raw data from a repository, we want to model the process and single out not only team members, but also the activities performed by them, and how each and the other relate through the workflow of the process.

Afterwards, we will describe the development of the alpha version of our system, some other small programs that helped us validate the results, and the problems encountered during this stage. We are also going to describe the testing and validation process, and finally, we will make a brief discussion of the results achieved.

During this preliminary phase of our work, we collaborated with six different groups of students. These are in the last year of the Bachelor's Degree in Informatics Engineering at the University of Coimbra.

In the past semester (from September 2022 until January 2023), they attended the course Software Engineering, where they were asked to build a software system at the request of a real client. Their primary challenge was to work as a software development team in an agile model by segregating themselves into different smaller teams (requirements team, development team, and business team, among others).

Since they used GitLab to store and share their repository among themselves, we were able to get their permission to access and use their repositories to test, debug, and validate our current version of the system.

## 4.1 Functional Requirements

A Functional Requirement (FR) is a functionality that the system is expected to perform. These requirements may not only be functionalities, but may also specify behaviours the system should have according to certain conditions, or even services he should provide [28]. They can also be seen as a bridge between what the client wants and what the developers should implement. Therefore, they should be written in the clearest and most explicit way possible.

For this system there were some FR's identified, more explicitly what kind of data we want to retrieve from the repositories and even some further analysis into them. All of the requirements found may be seen in Appendix B.

These requirements are presented in the form of a User Story (US). A US is a scenario describing the user experience that may be expected when a FR is correctly implemented in a system. These scenarios are usually accompanied by a US id, a title describing the FR that it is answering to, a priority (normally going from high to low), and its validation criteria, that is, the way we can validate if our program is working accordingly to what is expected of it.

The FR's with high priority are the ones that answer directly the problem we are facing and trying to resolve. The first one is presented in Table 4.1 *US 2 - See raw information*:

US #2	Title See raw information	Priority High
<b>User Story</b>		
As an activities tracker,		
I want to see the raw information of each commit		
In order to be able to perform debugging in my data collection		
And create new rules to transform raw events into semantically rich activities		
<b>Validation Criteria:</b>		
If want to be able to see the raw information of a repository		
When I insert new input data in the target repository		
Then I will see a raw view of the data inserted.		

Table 4.1: User Story 2 - See raw information.

Before the mapping *per se*, it's relevant to extract a report with the raw information collected from the data of the repository analyzed. These reports could help to clear out some doubts with more detailed information regarding what actually happened or simply show some underlining knowledge that can't be seen when mapping the objects. Taking this into account, it's expected that the system is able to generate a log with all the data retrieved and the information asked from it in other requirements.

US #3	Title Mapping objects	Priority High
<b>User Story</b> As a process engineer, I want to see the mapping between raw events and activities in a project In order to be able to understand, debug and see the process the team is following		
<b>Validation Criteria:</b> If I want to be able to see how my team is working, When generating the mapping from raw event to activity Then I will see how my team is operating according to the retrieved logs.		

Table 4.2: User Story 3 - Mapping objects.

In the previous table, also presented in 4.2, we can see the *US - Mapping Objects*, where it is expected to be able to see a network of the whole development process. Also related to this requirement, we have the table 4.3 *US 14 - Task Monitoring*:

US #14	Title Task Monitoring	Priority High
<b>User Story</b> As a project manager, I want to know which activity/task was developed/worked on in each commit In order to monitor an issue evolution		
<b>Validation Criteria:</b> If I want to monitor a task, When loading the event data I will be able to know if it was completed or if it's being taken care of.		

Table 4.3: User Story 14 - Task Monitoring.

This one is probably the most challenging and the most important FR - to see if we are able to single out a task from the project we are analyzing and see how, when and with whom it interacts with through each stage of the development process.

## 4.2 Non-Functional Requirements

A Non-Functional Requirement (NFR) describes how a system should operate or which characteristics it must obey to fulfil the needs of the stakeholders or of the business it belongs to. Unlike FR's, which specify functionalities and components of the system, these cannot be developed directly into the code, but rather be *emergent* properties of the system, which means these have to be converted into actionable FR's or drive the software architecture design.

Although there are many definitions and specifications of possible NFR for a system [23], not all of them have to be applied to the product in development. This section will present the NFR's for this research.

## Maintainability

In the lifecycle of a software system, the longest and most important phase is its maintenance after its release to the public or to the client. Every time an error is detected or the system starts malfunctioning, it should be further analysed and tested to try to fix those problems and return them back to its users as fast as possible. This is what defines **Maintainability** [23]: prepare the software so that the moment problems arise they can be easily and quickly fixed.

This is also important for research work such as the present one so that, in case there will be further improvements and examination involved, the code can be easily understood and updated with new functionalities and components, or even integrated with new systems. In case problems arise, as already explained, it's also useful to have it clearly implemented to discover easily what is the trigger for each problem.

Since our system belongs to the second case and is being developed as a means to answer a problem, our maintainability will require us to produce documentation such as this report describing what the system does and how it works. Another way to allow our system to be as maintainable as possible would be to add documentation in the shape of comments throughout the code of the system so that a specific method can be addressed and explained even further.

## Extensibility

**Extensibility** is also a key feature that will require some attention to attend to. It states how a system reacts or how easily it can adapt to new technologies, what are the specifications that the software needs to run the product, or even more specifically, to which software is it bound so that it works with no problem [4].

Regarding the system that is going to be developed, extensibility is important when it comes to which systems it's prepared to retrieve data from. For this first version, the system is able to get data from both Gitlab [10] and GitHub [9].

Even though these are the high priority NFR's for the system to be developed - *Maintainability and Extensibility* - and the ones that will be more focused on, there were also other NFR that were identified. These other NFR may be seen in Appendix C.

## 4.3 Constraints and Limitations

A constraint is a non-negotiable requirement that our system absolutely needs to comply with. They are the obligations that will define the architecture of our system, settle down some of the design choices, and condition both the development and the usage of the system being created [16].

Constraints can be split into business constraints and technical constraints. The prior ones are not straightforward, since they only imply what must be accomplished without stating how it must be done. However, these constraints can still have a great impact on the design of a system. Since the work being developed is purely for research ends, there is no business constraint attached to it, but there is the one that every research work has, which is the goal and the purpose of the work being done. Just like what was stated in section 1, this research aims to find if it's possible to find activities that took place in the project only by looking into the raw data from the project's repository and, if so, if we can turn those activities into a model of the process.

Technical constraints are more clear as far as what they want to impose on the product. They can be regarded for either internal or external systems that will be used, specific design decisions that should be made, or even limitations that must be taken into account when designing the system.

Again, since the purpose of this work is research, there is some degree of liberty that we may or may not use. However, there are still some guidelines that must be taken into account. To begin with, all the data and repositories that will be analyzed must come from the Git framework (section 3.2.4), specifically from either GitLab or GitHub. This was established taking into account the many perks these frameworks offer, such as the variety of open-sourced repositories, all the data that may be retrieved from them, and the fact that these are one of the standard Information System (IS) used by developers nowadays.

Another technical constraint is related to the development language to be used. We chose Python not only for being considered a simple yet powerful language, but also to take advantage of all the libraries at hand for managing this kind of data, such as the API for both GitLab and GitHub, the pydriller library (section 3.2.4), and the process mining library for Python pm4py (section 3.2.4).

Besides constraints, there can also be some limitations to our project. These can be described as self-imposed constraints that were settled based on many reasons, like a limitation of resources or knowledge. In our case, given the limitations of the libraries used, we had to fix the limitation of only getting the data from one repository at a time. If a project is using more than one repository, these will first have to be analyzed separately, and only after all the data has been extracted we will be able to make an attempt at crossing the data and analyzing both repositories as one.

## **4.4 Architecture**

The preliminary architecture of a software system is meant to show how a system will be organized and how the different components, both internal and external, will interact with each other [28]. It's also at this phase where many important development decisions are settled according to the type of system being developed and what we want to prioritize.

In our case, the architecture is especially important regarding the NFR's and how to materialize them. One of the main challenges will be how to assure the extensibility of the system: as already discussed in 4.2, even though in the initial stages the system is expected to receive data only from GitLab and GitHub, it's desirable that, in the future, it will be portable to more systems.

With this in mind, the architecture we are looking for is of the sort of pipe and filter. After loading all the data we need from the repositories, we are going to filter it and save it in the desired format so that we can model it and see the results.

The architecture of the alpha-system is shown in Figure 4.1. However, most of the development of this preliminary system was focused on the component *Event Generator* and the *csv Converter*. First, we retrieve all the information from the project's software repository to be studied with the *Load Data*. After, we go through all the data, analyzing it according to the requirements and the information we want to find in *Analyse Data*, which will generate a queue of sorted events that show the timeline of each activity that took place in the development of the project (*Event Queue*). In the end, this event queue will be exported to a *csv* file that can be transferred to a *User Interface* for the user to have a visual representation and visual mapping of the project. For the moment, the *User Interface* can be any external tool that takes *csv* files as input.

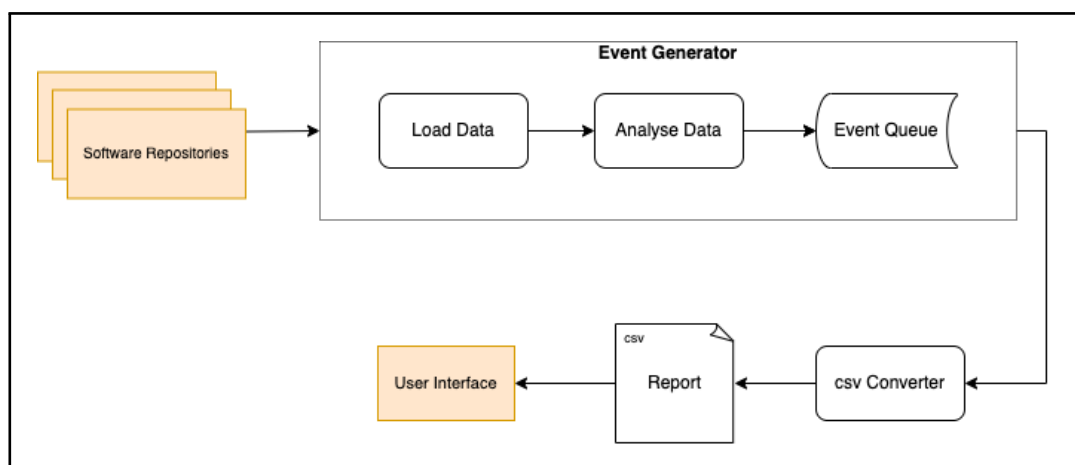


Figure 4.1: Architecture of the system to be developed.

As previously stated, the IS's that will be used during this research are GitLab (3.2.4) and GitHub (3.2.4). However, many other IS's could also be used to retrieve repositories or the ticketing systems used by the team, such as Jira <sup>1</sup>, Trello <sup>2</sup>, or Solarwinds <sup>3</sup>.

Regarding the component *Event Generator*, that is where the focus of our work was during this first iteration. With the help of the API's and the PyDriller library,

<sup>1</sup><https://www.atlassian.com/software/jira>

<sup>2</sup><https://trello.com>

<sup>3</sup><https://www.solarwinds.com>



we can extract every single commit in the repository and the tickets that were created. With all this information gathered, we can examine it even further to see the common points between each commit and each issue and track down which artefacts each user participated in, among others. In the end, we will be able to generate an event queue sorted according to the timestamps, where each event will describe an activity that took place in the project, show either the file or issue changed (if any), who performed it, and how many changes there were. We will consider an activity any kind of action performed by a single person in the repository (for example, if we have a new issue that was created and someone was automatically assigned to it, this can be split into two separate events: the creation of the issue and the assignment of the issue to someone responsible).

This event queue will then go into a converter to generate a *csv* file with all the information, showing one event per line. This file may then be used as input in several other tools such as Microsoft Excel, ProM, or Celonis to get some visual statistics or even to generate some graphs that may be used as a visual aid to see the different interactions and mappings found during the data analysis.

For the moment, the *User Interface* will be left on standby since the main purpose of this work is research and not commercial. However, for future work, a simple dashboard may be developed to better analyze the generated reports.

## 4.5 Development of the alpha-version

As mentioned before, this was an alpha-version of our program: it doesn't answer our main question nor lead us to any conclusion. This version's purpose was to start making some tests, see what we can do with each of the frameworks we will use, and slowly keep growing and adding more complexity until we discover the key to tracking down an activity throughout the workflow.

First, we loaded the data from the repository. In order to ensure the extensibility (section 4.2) of our program, we created different functions according to the system and the API used to retrieve the repository. For the moment we are able to use either GitLab or GitHub.

Since we can get the data from different systems, we need to save the data retrieved in a common format in order to be applied to the same functions with no problem. With that in mind, as we traverse through the different data from the repository, we save each author, each issue and each modified file in separate lists and dictionaries organised as follows:

```
files = { name_file: <Class> File }
authors = { name_author: <Class> Author }
issues = []

class File:
def __init__(self, name, path, file, date):
    self.name = name
```

```
self.path = path
self.file = file
self.authors = {}
self.stage = self.setStage()
self.latest_change = date

class Author:
def __init__(self, name):
    self.name = name
    self.commits = 1
```

After all the data is collected, the system runs a `cleanup()` version that traverses through the list of files to find any automatic or startup files that are created by the system itself, and discards them from the collected data. Since they were created by the repository itself and not purposefully by any team member, we decided they weren't relevant for this preliminary system.

After loading all the data we need, we implemented some of the most basic requirements. Even though some need to be further enhanced, from Appendix B we developed the requirements 2, 4 to 6, 8 to 13, and 15 to 19. As mentioned before, we are developing this system in a way that slowly leads us to our final answer and in each iteration grows its complexity. That's why we started with the smallest and easiest requirements whilst some of the requirements not developed may be considered the most important ones and with higher priority.

For the purpose of easily testing what was implemented for this first version and visualising the system's output, a small menu was built in the Command Line Interface (CLI).

First, it asks the user if it would like to see the results in the timeline of the last sprint or if the user would like to get the analysis from the full repository. Since the requirement destined to identify the beginning or end of a sprint (Table B.7) hasn't been implemented yet, we consider the last sprint as the timeline of the last 15 days, that is, the only data retrieved from the repository is the one where its timestamp falls in that period.

```
Do you want the statistics by sprint?
1 - Yes; 0 - No
```

After answering, the user will be presented with a menu listing all the requirements that were implemented as shown below:

Insert one of the options below:

```
Req 2 - Generate user report
Req 4 - In which development stage each member is (REQ
↔ , ARQ, ...)
```

- Req 5 - Which files each person authored
- Req 6 - Which authors changed each file
- Req 8 - Critical Files
- Req 9 - See active members
- Req 10 - Which files have more authors
- Req 11 - Which files have a single author
- Req 12 - Which member made the most commits
- Req 13 - Who made less commits
- Req 15 - Issues closed
- Req 16 - Which authors committed in the previous 24h
- Req 17 - See issues without assignees
- Req 18 - Which development model is being followed
- Req 19 - List of users per commit
- 0 - Exit

Lastly, as it was already mentioned in section 4.4, we don't have any dashboard or visual aid for the results besides the menu that was presented. However, by running requirement 2, the user obtains a *csv* file with the number of commits, number of modified files and number of tickets assigned to each author.

This current solution allows the user to input the *csv* file in any other user interface system such as Microsoft Excel or ProM to visualize some statistics.

## 4.6 Other useful sources

In order to support the validation of the results, we had the need to develop two auxiliary programs: **report\_generator.py** and **name\_mapper.py**. In this section, we are going to make a brief overview of each one of them saying why they were useful and supporting them with examples.

### **report\_generator.py**

The first program, **report\_generator.py**, gives us a report with all the information available from the repository that we retrieved. That is, it directly accesses the functions of our system where the requirements are specified and saves the output in a file.

We also thought it could be interesting to see the advances of each team along time. With that in mind, we enabled this program with a versioning system: when it creates a new file, it checks if there's already a report from that team or not. If there is, then it counts up the version, keeping both files saved (if we have version 2 of a report for team A, then we will save version 3 for the same team).

In Figure 4.2 we can see a template for one of these reports. In the beginning, there's a header with the name of the repository being analysed, the date the re-

port was generated and the time it took to retrieve all the commits. After that, the outputs will be listed according to the requirement they are answering (the requirements presented in the report are conditional to the requirements that were already implemented in the system).

In the following Figure 4.3, we can see an excerpt of the beginning of a generated report for the team KahUC.

### **name\_mapper.py**

During the development of the alpha-version and the consecutive testing with several groups, we noticed that there were users with more than one name in the commits. This may happen due to many reasons, such as the user contributing to the repository with more than one account, or changing its username in the middle of the project. In either case, we thought it was necessary to try to map the names and find as many associations as we could between names.

Given this, we developed the program **name\_mapper.py** where, given a *csv* file with the first column containing the list of usernames, it would try to find matches among them. Unfortunately, the current solution doesn't cover every case, and after the mapping is done it still requires a manual double-check. However, we were able to track down names where:

- The user has at least two usernames with space and the first and second names match (for example, *Rita Fonseca* and *Rita Fonseca 2018294751*)
- The user has at least one username with no spaces that contain its first name (for example, *Rita Fonseca* and *rita2018*)

These solutions only work if there is only one person with that first name. Otherwise, the program won't be able to map to which person the username belongs.

Taking as an example the Quizipedia team, from an initial list of 51 names we ended up with only 37.

## **4.7 Problems Encountered**

During the development of this first version of our system, we encountered some issues that conditioned our work and made us find the need to adapt and try to discover some new solutions.

Some of these issues were already expected to be encountered and, therefore, were left on hold for the next version of our system. These specific problems made an impact on the results obtained and, therefore, will be mentioned again in the next section 4.8.

```
===== FILE REPORT =====
REPOSITORY: A
DATE: 16-Jan-2023 (12:00:00)
REPOSITORY LOADING TIME: 43.42978811264038

~~~~ REQUIREMENT 4 > Development stage of each author
< the different stages will be listed here followed by each member username >

~~~~ REQUIREMENT 5 > In which files each author worked on
< each members username will be listed here followed by the files >

~~~~ REQUIREMENT 6 > Which author worked in each file
< the different files will be listed here followed by each member username >

~~~~ REQUIREMENT 8 > Get all critical files
< the different files will be listed here >

~~~~ REQUIREMENT 9 > Get active members
< each members username will be listed here >

~~~~ REQUIREMENT 10 > Files with more authors
< each file will be listed here >

~~~~ REQUIREMENT 11 > Files with only one author
< each file will be listed here >

~~~~ REQUIREMENT 12 > Who made more commits
< each members username will be listed here >

~~~~ REQUIREMENT 13 > Who made less commits
< each members username will be listed here >

~~~~ REQUIREMENT 15 > Completed issues
< issues will be listed here >

~~~~ REQUIREMENT 16 > Who made a commit in the last 24h
< each members username will be listed here >

~~~~ REQUIREMENT 17 > If there are issues with no assignee
< issues will be listed here >

~~~~ REQUIREMENT 18 > Which development model is being followed
< development model will be stated here >
```

Figure 4.2: Template for the reports generated.

```

===== FILE REPORT - LAST 15 DAYS =====
REPOSITORY: KahUC
DATE: 12-Dec-2022 (11:35:28)
REPOSITORY LOADING TIME: 43.42978811264038

~~~~ REQUIREMENT 4 > Development stage of each author
BUS
    JoaoESmoreira
    Samuel Machado
REQ
    Diogo Tavares
    João Pinto
    Ricardo Guegan
    Rui Santos
    eduC21
DEV
    Andre Moreira
    Eduardo22
    Emanuel Roque
    Filipe Figueiredo
    Gonçalo Almeida
    Guilherme Branco
    Gustavo Lima
    João Santos
    Rikes8
    Tomás Pinto
    brunosequeira
    lucasmatiasanjo
    ricardoquinteladev

~~~~ REQUIREMENT 5 > In which files each author worked on
Andre Moreira
    REQ_13.1_numero_de_quizzes_por_tag.md
    REQ_14.1_exportar_quizzes_em_XML.md
    REQ_15.1_importar_quizzes_em_XML.md
    tests.py
    urls.py
    views.py
Bruno Sequeira
    REQ_07.1_rever_quiz.md
Diogo Tavares
    WIRE_5.1_1.plantuml
    WIRE_5.1_2.plantuml

```

Figure 4.3: Example of a generated report for KahUC team. In Requirement 4, the code names BUS, REQ and DEV refer to the separate development stages of business, requirements elicitation and development, respectively.

In the beginning, one problem that emerged was due to PyDriller. The repository on which the system was being tested had a commit where there was a link to a sub-module of that same repository. This can also be known as a sub-project commit:

```
Subproject commit [id]
```

However, because PyDriller can't identify these sub-modules when traversing through the commits list, the system was always exiting with the exception `ValueError`.

Since this issue is a PyDriller limitation, and because a cross-repository analysis is out of our scope, our only option was to disregard these commits and

continue with the analysis of the other ones, leaving a note that a commit failed because of this situation. This note can be left in the CLI in case the system is running manually or can be left just below the header of a report.

Another obstacle was the difficulty of trying to map each member of the team to the development stage they belonged to, that is if they were assigned to the requirements team, development, or testing, among others.

As a starting point, we tried to make this mapping using a similar technique as the one described in [5]. We looked into the path of each file in the repository and tried to find some patterns in it that may have directed us to a hint of which development stage this file belonged to. For example, if in the path of a file we could find any of the keywords `DEV`, `ENV` or `DEP`, then we would know that this is a development file. After that, all it took was to see who made any modifications to that file.

Our solution quickly turned out to be a problem. There were many reasons for which this kind of mapping was meant to fail, such as the fact that there were several people that took part in more than one development stage (and therefore, modified many files of different stages), or the fact that the keywords chosen may not be entirely accurate.

For the moment, we left it as it was. Even though most of the cases weren't completely accurate, most of them were, and this may be a starting point to find better mapping criteria in the future.

One final problem worth mentioning is the difficulty of tracking down a requirement throughout a repository.

Since these preliminary tests were made with college groups, the names chosen for either artefacts, tickets or even branches in the repository didn't let us start working on this specific problem of our goal.

An easy solution to this problem would be to impose that the team follows a certain name format for either tickets, commits or files. This obligation will allow us to better track down an activity and see how it was developed, by who and in what timeline, among other statistics. If future teams don't follow this specific format, then we won't be able to see the activities as they are, and a warning may appear saying that it's not correctly formatted.

## **4.8 Verification and Testing**

There were two different sources that we used to validate the alpha-system: the first is the set of repositories from the college groups we mentioned at the beginning of this chapter, and the second set is from public repositories from GitLab.

The set of repositories from the college groups was used to debug and verify our system and see if there was no failure or wrong results. This verification was done each time with all of the six groups, since because of their differences some of them could capture errors that others wouldn't. These tests took place whenever a new requirement was implemented. In the next section 4.9 we analyze and go through these results by cross-examining them with some feedback we received from the respective students.

The other verification using the online public repositories in GitLab was made so that we could get in mind the relation between the time it took to gather all the data from the repository and the number of commits - the size of the repository. In other words, to see how scalable our system was.

The time the system takes to get the data is mostly due to PyDriller cloning the repository every time it wants to use it to retrieve commits and, therefore, there's not much we can do to enhance its speed. However, it's always good to keep in mind the execution time. In fact, after obtaining these results, we concluded that in case we develop a dashboard or in case our system ends up being used outside of the research context, we would probably make it a real-time system that is always running and continuously updating the data as new information comes in.

In light of this, we had three successful tests with the repositories Octoapp<sup>4</sup> (> 4.000 commits), GnuTLS<sup>5</sup> (> 21.000 commits), and recalbox<sup>6</sup> (> 42.000 commits). We tested these for a 15-day time period data and for all the data in the repository. For the moment, even though it could manage a 15-day period, the fourth repository that didn't succeed in completing the extraction of all the data was wireshark<sup>7</sup> with more than 85.000 commits. Keep in mind that the number of commits presented here is the number that each repository had at the moment these tests were performed.

The following table 4.4 and figure 4.4 illustrate the results obtained that were already explained. Even though we tested only in a few repositories, as expected we can see the tendency to longer execution times as the number of commits increases.

	Octoapp	GnuTLS	recalbox	wireshark
Number of commits	> 4.000	> 21.000	> 42.000	> 85.000
In sprint (s)	63.19214630126953	114.53173303604126	436.2550573348999	1536.9174399375916
All data (s)	419.0770287513733	1178.6181738376617	3128.676027774811	-

Table 4.4: Public repositories execution in seconds.

<sup>4</sup><https://gitlab.com/realoctoapp/octoapp>

<sup>5</sup><https://gitlab.com/gnutls/gnutls>

<sup>6</sup><https://gitlab.com/recalbox/recalbox>

<sup>7</sup><https://gitlab.com/wireshark/wireshark>



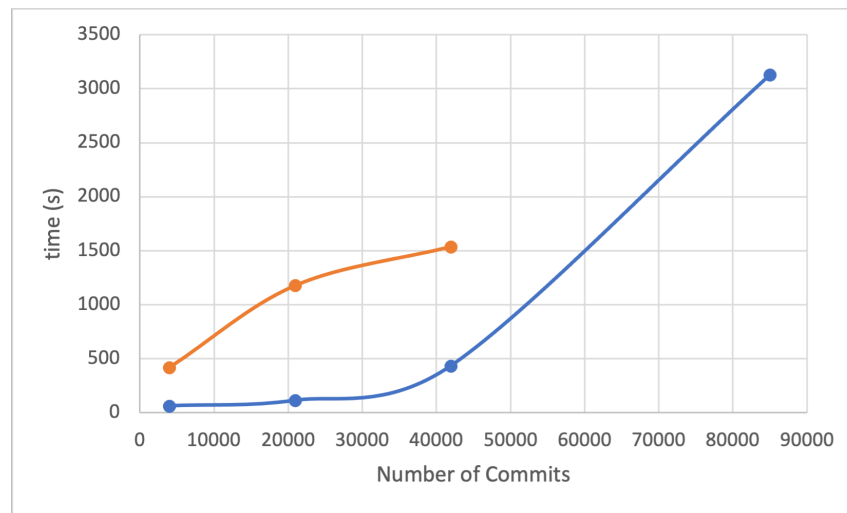


Figure 4.4: Number of commits in relation to the time of execution.

## 4.9 Discussion of the results

In order to validate our results from the college students, we created a Google Forms 3.2.4 for them to fill out. Said survey may be seen in Appendix D. In this section, we will focus on crossing their answers to what we were already expecting to happen - since this is only an alpha-version - and the results we obtained.

In the forms, we first began by asking some simple questions to help with the mapping, such as:

- What their name was in their GitLab account
- In which activities or development stages they were more involved in
- What group they belonged to

After that, they were redirected to a page showing the results obtained by the alpha-version regarding the members that worked in each development stage, the most active members and the less active members. Given this, they were then asked if they agreed with the results and if they had anything to add.

Even though the six groups combined gathered around 200 students, we only got 30 answers to the forms created by the groups KahUC, byDeiCrowd, QuizzES, and moelasware. We also got 2 answers from the Quirkep Up Software team, but since these were both positive and with no further comments they will be disregarded.

Next, we'll go through the most common comments left by the students. These answers turned out to be incredibly useful since not only did they mention some aspects that were not regarded by us before, but also mentioned on several occasions aspects that we were already counting on.

### *The information about more active members seems slightly wrong*

One of the most common answers was regarding the results of the most active members. These results were obtained through the requirement presented in table B.19 where we list the different members according to the number of commits, from highest to lowest. After that, we simply saw the members that were on the top of the list.

One type of comment was that there were members missing from the list of most active members. This may happen either because they changed their name in the middle of the project and we weren't able to map it (which would mean the real number of commits would be split between two usernames), or because the member was correctly assigned with a high number of commits, but simply not of the first top 5 members.

Even though we can't be certain if there was a member assigned with the wrong number of commits because of the name difference - since there were names we weren't able to map not even by hand - it's a possibility that we need to be aware of. The latter reason presented is not a significant problem, since we are aware that the member has a high number of commits.

Other comments mentioned that this list presented wasn't "*reflective of the volume of work done by them*". This may be related to the way the list was presented since we didn't mention the number of commits or the rest of the list with the other members. However, as mentioned before, software development is composed of a set of unstructured (manual) activities where it can be hard to see things clearly. The only way to see the amount of work in a repository would be either by commits, modifications or even comparing team members.

One final comment, still regarding more active members, said that there was a group of members that worked closely together, for example using pair programming, and so the work done by some of them could've been mingled in the commits of another member. Regarding this, there's not much we could've done. By having an online repository, each person can do their own work and submit it as their own. Even if it's a joint component that needs more than one person, they could split the work or even develop in stages completing the other person's code.

### *The statements don't reflect well the elements who worked in various areas*

Some of the feedback that we were already expecting is related to the fact that there were several members of the team that worked in several stages of the development. Since these are college groups and they were meant to see how software development works, this was something that was already expected. However, this may not be a common behaviour when we move to more professional groups.

Even though it's not something that will be paid much attention to, it's always good practice to be aware of how it works and how it can be solved. Our current

solution to these cases is to see in which development stage the member made the most commits. Of course, this isn't perfect, but it's good enough just to give us an idea.

One of the most common answers from the students in this regard was that they had a big presence in several development stages but only appeared in one of them and, therefore, the results didn't reflected accurately their role in the team. This may happen with the change of workload during the development of software in the different stages: in the beginning, requirements elicitation needs most of the attention, whilst in the end all the effort is put into development and testing.

Another comment was that there were stages missing in the results. This is due to one of the problems already mentioned in section 4.7.

### *People that didn't work much*

While in the first topic we mentioned complaints of active members that were missing or were wrongly mapped, in this one we will talk about the other cases where the less active members' information isn't accurate. Unfortunately, it's common in groups of many people to always have at least one member that works less or doesn't work at all.

The list of these members was generated in the same way as the previous one (using the results from requirement B.19). With this in mind, in the cases where a single or a few team members did all the work, it's easier for these "non-workers" to appear with a big presence as if they had a positive contribution to the project. Sometimes all it takes is to be active in assigning or completing small tasks, or even just update their profiles, and it boosts their reputation more than it should.

### *Many work was done outside git*

These kinds of comments were probably the most unexpected ones we were presented with. Since all the data we are using in the analysis is related to the work done and submitted to GitLab, and therefore all the work is expected to be uploaded there in any change, we weren't considering the work that was done outside Git and that doesn't have a direct representation in it. One of the examples a student gave was the planning of the sprints and the controlling of tickets was done using an external communication tool (which we are not expecting to embed in our system), instead of the ticketing system that GitLab provides or any other similar systems.

However, in our thesis, we are trying to single out activities by looking into the development process and trying to model it. So even though that kind of work is always important, our main concerns are the artefacts and the usage that the members give to GitLab.

The NFR regarding the extensibility of our system shall try to address this issue by being able to integrate other tools used in the same project.

## 4.10 Summary

In the present chapter we name some of the most important FR's for the alpha-system identified so far (see a complete list in Appendix B) and the NFR's that we need to attend to. We also express the constraints and limitations we need to comply with.

We also describe the architecture thought for this first version. A pipe and filter that, by receiving data from a repository, will filter the events into a formatted list. This list, after being converted into a *csv* file, can be used as input in a visual interface so that the user can see the results.

We also focused on the development and validation of this system. After a brief explanation and an overview of some other auxiliary scripts, we spoke about some of the problems we encountered, some of them out of our scope such as the limitation found in PyDriller, and others that need to be better looked into, like the mapping of the team members and the artefacts to the right development stage.

Finally, we ended our chapter by explaining the testing process in 4.8 as well as some first experiments made, and by discussing the results obtained. This reflection was based on the feedback we obtained from the college students who collaborated with us. Most of their comments were regarding the most and less active members and the fact that members that participated in more than one development stage were not accurately mapped. Even though we were aware of these limitations, we have now better insights to enhance this in the future.

In the next chapter, we will make a similar analysis for our final system and for the final results that finally helped us tackle the real problem that we are trying to solve.

# Chapter 5

## System's Architecture

For this chapter, we will start by making an overview of a script (*Playground*) that was used in order to better analyse the data we were dealing with and get a better idea of how we proceed for the final architecture.

Said architecture is then presented. Just like the one shown in the previous chapter, this one will also work as a pipe and filter, and each separate stage is described in further detail.

### 5.1 Playground

What we nicknamed as *Playground* is a script whose only purpose was to present the commits and tickets from a Repository according to their timeline. This program was developed before we designed the architecture with the goal of trying to identify patterns in the events. That way, we could plan the final version and be more confident of the best way to tackle our problem.

From the alpha-version of our system previously developed (see section 4.5), we already had the means to gather the information retrieved from the Repository, i.e. commits and tickets. Our main problem now was how to turn this data into separate events, and consequentially, how to find the activities that took place within them.

The only way we could find was the most obvious one: look into the data, try to find what we (as humans) figure out of it, and then try to write down our chain of thought and how we reached that conclusion. For that, we developed *Playground*.

All that this program does is fetch the data from the alpha-version, sort the commits and the tickets according to their timestamp, and finally save the final list in a report. That way, we could say that this report contains the history of the work developed by that team in the form of events. Like the `report_generator.py` mentioned in 4.6, we also provided these reports with a versioning system.

In the end, the start of a report would look something like this:

```
===== PLAYGROUND REPORT =====
REPOSITORY: Quirked Up Software
DATE: 14-Feb-2023 (15:43:11)

COMMIT:
{'author': 'Pedro Filipe',
 'branch': 'main',
 'date': datetime.datetime(2022, 9, 18, 10, 39, 26),
 'description': 'Initial commit',
 'modified_files': {1: {'name': 'README.md', 'path': '
    ↪ README.md'}}}}

COMMIT:
{'author': 'prpfilipe',
 'branch': 'main',
 'date': datetime.datetime(2022, 9, 18, 12, 6, 43),
 'description': 'add index.js',
 'modified_files': {1: {'name': 'index.js', 'path': '
    ↪ DEV/src/index.js'}}}}

...

```

This program was useful to help us design the architecture and not only settle for a rule engine to decipher the activity of each event but also to write some of those rules (see Appendix E). It was also a bonus help when it came to developing the Data Fusion interface of the architecture as presented next (section 5.2.3).

## 5.2 Architecture

Similarly to what was described in section 4.4 for the alpha-version, our final architecture will also be adopting a pipe and filter scheme taking into account the same Non-Functional Requirement (NFR) mentioned above in section 4.2 (maintainability and extensibility).

As mentioned earlier, with a pipe and filter architecture we can have several different stages that will be used to filter the data as it goes from one stage to another. Each stage receives at least one file as input with the data to be filtered (the first stage receives the raw unfiltered dataset) and outputs at least one file with the data filtered as expected (the last stage outputs the final version of the dataset with the final results we were looking for). As such, the architecture thought for our final system may be seen in the following figure 5.1.

After fetching the data (5.2.1) in the first stage from the several systems that compose the repository<sup>1</sup> of the project in question, we send the collected data directly to the following stage where we aggregate (5.2.2) the data as an event ac-

---

<sup>1</sup>In case of doubt, see our definition for Repository in section 3.1.

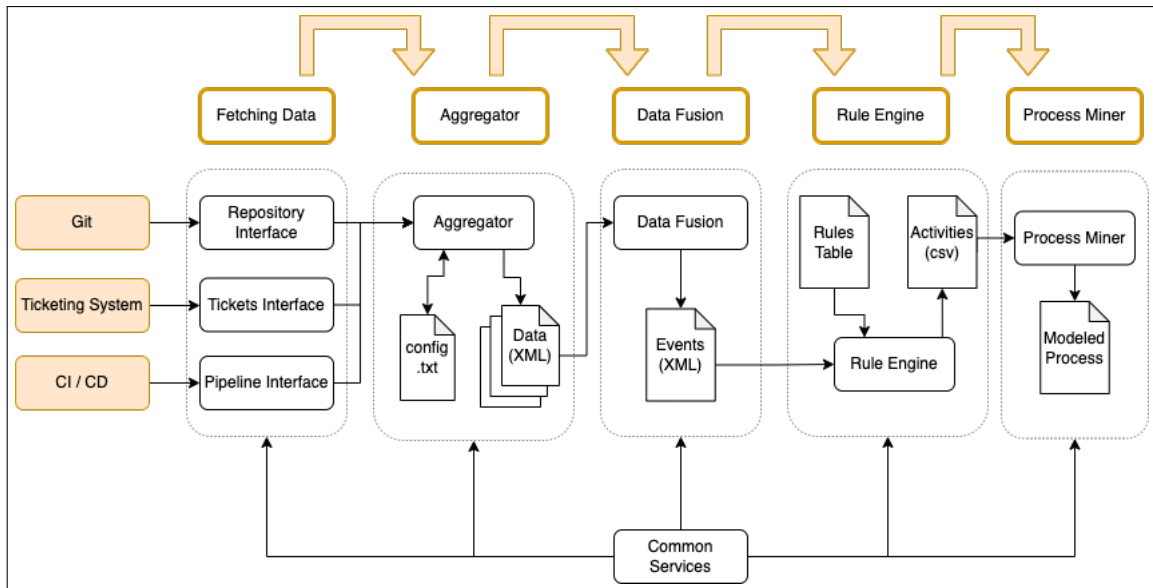


Figure 5.1: Architecture of the final system.

ording to its type (commits, tickets, ...) and saves each one in a specific file. Later, we fuse the data (5.2.3) from multiple files into a single one and sort it according to their datetime.

With all the data ready, we move on and parse it through a rule engine (5.2.4) [7] [27]. After loading both files with the data and with the rules to be loaded in the system, we check each data element to see which rules apply (if any) and that way identify the activity that took place in that event. In the end, all these activities are saved in a *csv* file to be used as input in the process miner (5.2.5), which in turn we will use to model the process as desired.

Since there are many common methods used by each interface, we decided to refactor these and join them into a single script. That way, whenever each interface needs to use one of these methods, all it needs to do is call it from the other file. This file may be seen in figure 5.1 as the Common Services box.

In the following subsections, we will explain each of these stages with further detail for a clearer understanding.

### 5.2.1 Fetching Data

This stage of our architecture, being the first one, is in charge of retrieving the data from the aggregate of systems that make up the team's Repository.

This stage is currently composed of three separate programs, each one of these in charge of fetching the data from the respective system. These are the Repository Interface, which currently retrieves data from either Github or Gitlab; the Tickets Interface [8], in case the team doesn't use the ticketing system provided by Github or Gitlab and instead uses other third-party systems like Jira or ClickUp; and the Pipeline Interface to get the pipeline from the repository through the CircleCI platform.

Each one of these interfaces uses the Application Programming Interface (API) of every single system where the data of the team is (or can be) stored, and some additional libraries (like `pydriller`<sup>2</sup>) that turned out to be useful. For the moment, only the systems mentioned above have been implemented.

In order to get our system more structured, we decided to split the code according to the type of data that we are retrieving from each one of the API's, that is, repository data (commits and tickets), tickets or pipeline information. With this division, we ended up with the three programs previously mentioned.

### 5.2.2 Aggregator

This interface has a single program that receives the data retrieved from the previous stage and saves it in separate XML files according to the type of data (*commitsXML.xml*, *issuesXML.xml*, ...).

Some may argue that this interface wasn't necessary and that we could simply save the data in an XML file straight away in the Fetching Data interface. However, in this stage, we implement a mechanism to help speed up further "data fetches". Along with the XML files that it produces, this program also updates a *config.txt* file where it saves the `datetime` of the latest piece of data present in the dataset that has been saved earlier to the file. That way, if the Repository is updated in the meantime and we want to get the new data, when the Aggregator fetches it, it previously sends the `datetime` of the last data recorded so that the respective interface knows from which point in time it needs to find the new data.

Afterwards, all that's left for the program to do is to merge both chunks of data, the one that was already saved in the file and the new one that was just fetched, save it to the XML file and update the *config.txt* file with the date of the new latest element of data.

### 5.2.3 Data Fusion

The Data Fusion is the stage where we join all the data in a single XML file sorted according to their `datetime`. At the end of this stage, we can safely say that the events have been created - each event will be stored in each leaf of the XML tree. An event is expected to have a similar format as the one that follows:

```
<data>
  <C-1234
    id="C-1234"
    author="Rita Fonseca"
    updated_at="2023-05-10 15:47:26"
    branch="['main', 'code']"
    description="Example of a event">
    <modified_files>
```

---

<sup>2</sup><https://github.com/ishepard/pydriller>



```

        <M-1 name="testing.txt"
            path="doc/testing.txt" />
    </modified_files>
</C-1234>
...
</data>

```

This program reads the files with the data saved from the previous interface, joins them and, before saving them to an XML file sorts them according to the date of each event.

## 5.2.4 Rules Engine

Here is where we mostly focused on and where we try to answer the first Research Question (RQ) (section 3.1.1): *Is it possible to turn raw events into activities?*

The best way we found to tackle this issue was to build a small rules engine [17] that parses a table with rules written by the team's manager or responsible in order to address the context of the project properly. The syntax and logic of these rules were already thought of and built by us as described in Appendix E. That way, as long as the person responsible for changing the rules follows these formulas, she can adapt them as needed.

After that, the engine will load the file produced from the previous stage - an XML with all the events found in the Repository - so that each one of them can be put to test by the rules loaded.

By doing so, we give each event as input to each rule. If it's accepted, then we label the activity of that event as the one stated by the rule. If not then the event is given as input to the following rule. In case the event isn't accepted by any of the rules, then the activity is considered *Housekeeping*.

It's important to mention that the order in which the rules are presented interferes with the activity selection. For example, if the directory where the team saves the mockups is inside the directory for the requirements and they want two rules, one for each activity - one for `Mockups` and another for `Requirements` - then the rule for the `Mockups` should come first since it's the one with the smaller scope. As such, they should be sorted not only by priority but also given their scope.

After all the events are mapped into activities, this interface creates a CSV file with all the activities found as shown below:

```

date;case_id;branches;activity
2022-09-18 11:29:18;Rita Fonseca;main;Start
2022-09-20 14:05:40;Rita Fonseca;main;Research
...

```

By giving the output as a CSV file, we assure that these activities can be provided as input to other external systems.

### 5.2.5 Process Miner

The Process Miner, as the name suggests, is where the process mining algorithms are applied and is the final stage of our pipe and filter.

This program receives the filtered activities from the CSV previously generated, loads them and runs the heuristic miner algorithm [31] to model the process. We chose the heuristic miner for the simple fact that it considers the noise present in the data and builds a model robust enough to be resilient against it.

When this stage ends, we are left with the modelled process according to the activities found and, that way, we are able to answer the last RQ 3.1.2: *Is it possible to retrieve processes from a set of activities?*

## 5.3 Summary

In the current chapter, we began by looking at *Playground* and the way it helped us understand the data we were working with, how it was organized, and how we could use it alongside some sort of set of rules to characterize each event according to their activity.

Afterwards, we presented the architecture of the final version of our system and each stage that it required, namely

- Fetching Data where we fetch the data with the events;
- Aggregator where we rearrange the different types of data and saved them on each respective file;
- Data Fusion where all the types of data are merged into the same file chronologically as events;
- Rule Engine where all the events are parsed through a set of rules to decide which kind of activity took place;
- and, finally, Process Miner, where we use the heuristic miner algorithm to model the process.

In the next chapter, we will proceed to the discussion of the results.

# Chapter 6

## Discussion of Results

In the current chapter, we describe how the testing of the system proceeded and make an analysis of the results that we were able to take from them.

At the end, we also plan to go over possible threats to the validation of our solution that have been identified.

### 6.1 Analysis

In order to validate our results, our first step was to analyze which of the groups of students already mentioned in previous sections (such as 3.2.3) were eligible for testing. This selection must be done to safely validate the results obtained as explained in section 6.3.

As stated in previous sections, testing the system only on groups of students carries its own risks attached. Despite the lack of professionalism - even though they may work correctly and finish the project, they will do it differently than if they were in a more professional environment -, in this case, we also noticed a language barrier: since the majority of students were from Portugal, when we came upon groups that only had portuguese students, their tendency was to write everything in Portuguese.

The portuguese language is not a problem for us to understand, however, the rules we wrote to parse the events were only thought for projects in english, and it didn't seem feasible to us to rewrite them all for a specific project only, since it's the most common (or even standard) language to find in this context. As such, our main criteria to choose which projects would be selected was based on the amount of events written in portuguese and english. With this in mind, the three projects selected were Quirked Up Software, KahUC and byDeiCrowd.

After selecting the desired groups, we submitted their data as input to our system. It's worth mentioning that, for a better analysis of the models given as output, any connections between activities or instances from activities that had a frequency lower than ten were not considered. Furthermore, the real models that we obtained from our final system may be seen in Appendix F. Following in this

chapter, we will present a simplified version of each model where, taking as an example the first model in Figure 6.1, we can see the start in the green ellipse and the end in the orange one. The activities are presented in the blue boxes as well as the number of times they appear (the darker the shade of blue the greater their frequency).

### 6.1.1 Quirked Up Software

The group that gave us a good model straightforward from the first iteration of the system was Quirked Up Software. This was already expected since this was the most consistent student project (that got the higher grade at the time) that was used throughout development for testing the system against possible problems. A simplified version of the resulting model may be seen in figure 6.1.

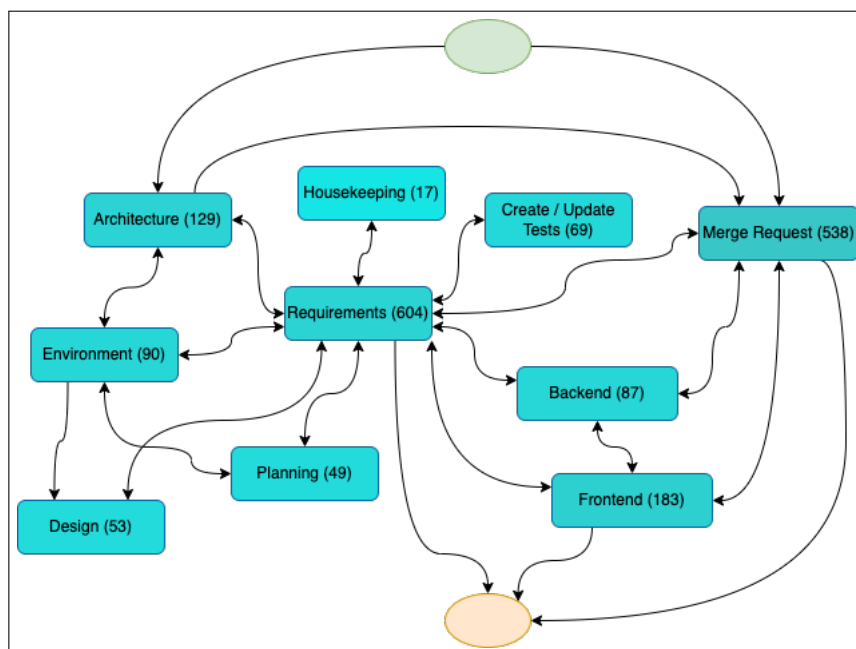


Figure 6.1: Simplified version of the final model for Quirked Up Software.

Something that may seem odd once we start analyzing the model is that the 2 first activities from the start of the model are *Architecture* and *Merge Request*. However, we need to consider that once a project starts usually the first events are simple ones such as creating directories and getting everything ready to start working, and that these smaller activities were taken out from the models.

Another positive result we get from this model is that, by looking at it, we can clearly see the team used an Agile approach. By looking into the connections between the activities, especially the one for *Requirements*, we can easily see that every activity was strongly connected and related to another (note the doubled-edge arrows present between most of the activities), which may hint for a requirements/product-backlog driven team.

Regarding the *Housekeeping* activities, which is the "default" activity for the events that couldn't be attributed to any other, this project only detected 17 *House-*

keeping events out of the 2348 total amount of events found (0.7% of the events).

One last relevant remark on this model is that, just like the starting events, some of the events made with the goal of "closing the project" may not be frequent enough to appear in the model. In the figure 6.1 shown above we can see that the closing events are *Requirements*, *Frontend* and *Merge Request*. While the latter one makes sense to be one of the last events made since it's the activity of merging together separate branches of work, the other two may seem more unconventional, but we manually confirmed that they have some hidden transitions in between.

## 6.1.2 KahUC

Whilst Quirked Up Software was the group that gave us a good model quicker, for reasons already mentioned, KahUC was the group that surprised us the most.

When we tested with this team's data, we first tried to run it with the pre-defined set of rules made according to the previous group. And, as expected, it gave us a bad model with many *Housekeeping* activities. However, after making a quick overview of the data to try to tailor the rules for this group, even with only small changes we were able to obtain the model shown in figure 6.2.

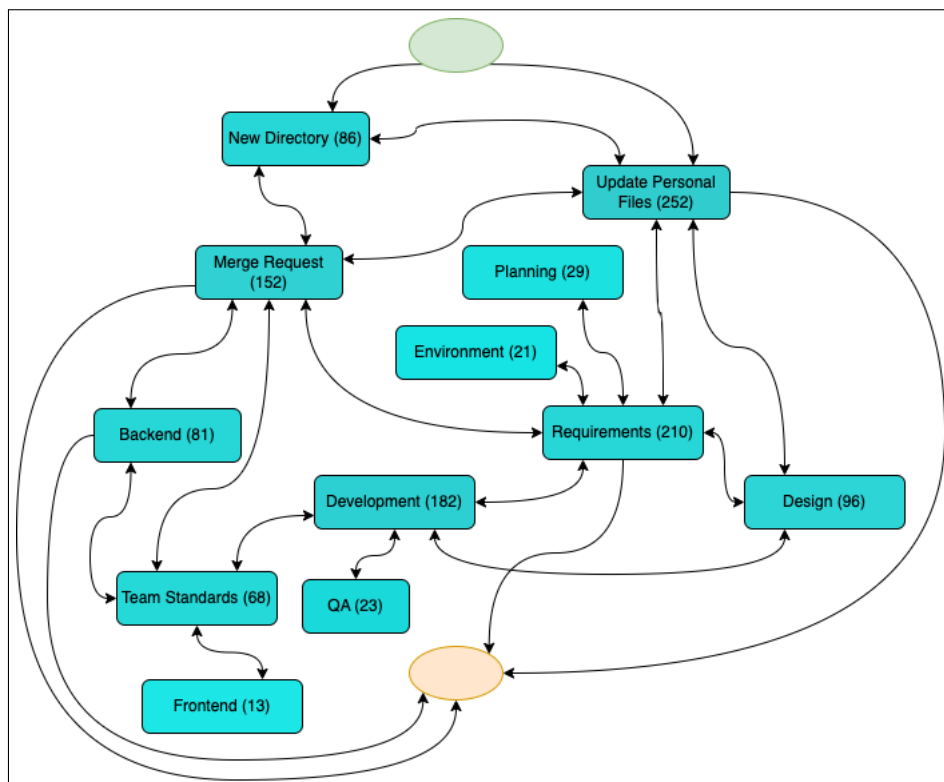


Figure 6.2: Simplified version of the final model for KahUC.

Regarding the starting activities, the *New Directory* and *Update Personal Profiles*<sup>1</sup> make sense and check out to what is expected. What this may tell us when com-

<sup>1</sup>The activity *Update Personal Files* refers to the creation or update of personal files (one file

pared to Quirked Up Software is that KahUC either had a good amount of events concerning the initial set-up of the project or that they made events of these during the whole project. The latter option may probably be the more accurate one, especially when it comes to the *Update Personal Files*, since it's something that is expected to happen during the whole project as it is being developed - this may be seen in the model through the amount of activities that go back and forth from *Update Personal Files*.

Concerning the other activities shown in the model, even though we can get a glimpse of an Agile approach, we can also see a small segmentation between development activities (*Backend*, *Frontend* and *Development*) and *Requirements*. This may point to some sort of definition of roles in the team, that is, they clearly defined amongst them who would be responsible for programming and who would take care of management (*Planning* activity), *Requirements*, among others.

When it comes to the last activities, *Merge Request*, *Backend*, *Requirements* and *Update Personal Profiles*, even though the first three could be perceived as "ending activities", we come to the same conclusion of Quirked Up Software: we can't be for sure if there isn't any hidden activity in between or not, especially between the *Requirements* and the end.

Finally, in the middle of all these remarks for KahUC's results, the most promising is regarding the *Housekeeping's*. As said in the beginning, even though the first set of rules wasn't good enough for this group, after only a few small changes to adapt them better to the group we were able to filter almost every single activity. That is, since activities with less than 10 events don't appear in the model, at most we only had 9 *Housekeeping's*, which considering the total of 1409 events seems like a great result (at most we only had 0.6% unfiltered events by the rules).

### 6.1.3 byDeiCrowd

Unlike the other two results, byDeiCrowd offered the most troubling model. Nevertheless, after we carefully analyzed it, we figured it would still be a good enough example to show another point of view for our system.

Our testing method was just like the one in KahUC, that is, we first started with the initial set of rules and afterwards looked into the data to carefully update the rules. However, we had to repeat the same process several times. The best model we could accomplish is shown in figure 6.3 below. It's also worth emphasizing that, when creating rules tailored to a specific project, we only took into account rules that we could guarantee would be used for more than one or two events. If there was an event that could have a rule but it might not be used for any other then we wouldn't consider it.

Something that strikes us right away from this model is the *Housekeeping* activities. Not only there are 264 of them, but also they seem to be related to almost ev-

---

per student) that their class's teacher asked them to maintain with the record of each student's contribution to the project. Even though it's not a very relevant activity for interpretation, it was relevant model-wise to keep it in the simplified version of the model.

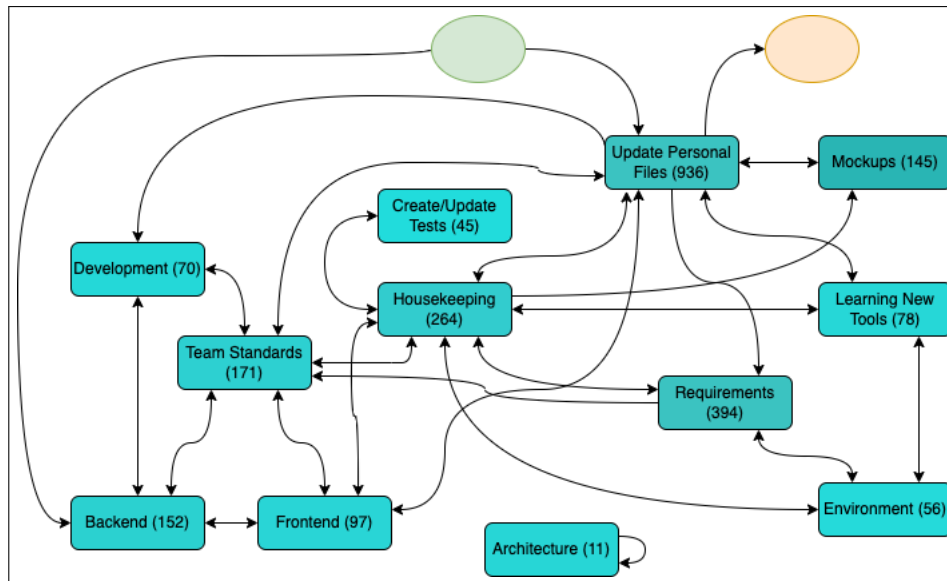


Figure 6.3: Simplified version of the final model for byDeiCrowd.

ery other activity just like the *Requirements* ones were in the previous tests. These are clearly badly filtered activities that may be due to events poorly recorded - events that didn't follow some sort of standard to help characterize them - or that were written in portuguese and couldn't be turned into a rule - this group also had some events in portuguese, not as many as the other groups that were taken out from the study, but enough to make a small compromise to the results.

In contrast with the other two groups, byDeiCrowd doesn't seem to be as ordered or as organized as them. Besides the *Housekeeping* activities, there are also others out of the ordinary. For example, the *Architecture* activity that is completely separate from the model, or the *Requirements* that, despite the amount of iterations, doesn't seem to be connected with any other relevant activity (like any *Development* activity, *Create/Update Tests* or even *Mockups*). But again, these connections cannot be trusted completely since a lot of the *Housekeeping* activities are most likely the ones that are missing for better results.

Given this, there is not much more to say regarding byDeiCrowd's results. Between the amount of *Housekeeping's* registered and all the "non-value" activities that had to be kept in the model for accuracy (like the *Update Personal Files* in KahUC), this group is a good example of the importance of using standards and recording events according to the same "formula". Nevertheless, despite having 264 *Housekeeping* events, out of the 2894 they only represent around 9%, which may indicate that we were on the right path with the rules.

## 6.2 Final Remarks

As seen, the obtained results gave us positive results when it came to our first hypothesis and our proposed Research Question (RQ)s.

When we presented the overview for our final architecture in 5.2, we were able to see how and where it was expected to answer the RQs. With the previous models presented, we were able to prove our thesis, not only by mapping events into their activity type (according to the first RQ presented in 3.1.1), and by creating a process model with those same activities (regarding the other RQ in 3.1.2).

It's important, however, to highlight the results from the last test, which revealed some frailty regarding the rules used. Or rather, revealed the importance for the team to follow the same standard and the same format when recording a new event. That way we could get a better tailoring of the rules and a better model of the process. Since we didn't work in close proximity with the group it was already expected some kind of problems such as these (as it is also mentioned in 6.3).

Nevertheless, the other two groups gave us very satisfying results. One of the units that we believe is a good measure of a good or bad mapping from the events until the final model is the number of *Housekeeping* activities. Even though the values were mentioned above, they may be seen in table 6.1. *Housekeeping* is the type of activity given to an event that didn't obey any of the other rules given to the system. Mainly these events are either ones that don't have a big relevance for the project, and so aren't given any rule; or can be events that were badly recorded in a way that didn't allow for any rule to be created or to be applied.

	Number of Events	Number of <i>Housekeeping</i> 's	% <i>Housekeeping</i> 's
Quirked Up Software	2348	17	0.7%
KahUC	1409	$\leq 9$	$\leq 0.6\%$
byDeiCrowd	2894	264	9%

Table 6.1: Number of *Housekeeping* activities compared with the total number of events.

Since our main goal is to get the best model of the process and so keep it as accurate to the real process as possible, the number of *Housekeeping*'s are desired to be as little as possible.

Another positive result was the fact that, simply by looking into the model, we could easily see the interactions between each activity and, as we expected, draw the history and the trace of information and of how they proceeded.

### 6.3 Threats to Validity

Despite all the planning and design that we are doing in this section, there are always some threats to the validity of our final conclusions that should be mentioned and taken into account when analysing our results. We are mainly seeking all the possible ways that our results may be incorrect or inaccurate.



## **Testing on groups of students**

One of our main threats is the scope on which we are testing the system developed. As mentioned before in 3.2.3, we will be using different groups of students with a still small and preliminary knowledge of software development.

The first question that comes up is: is the testing towards an audience composed of students enough? Clearly, software developed by students, no matter their degrees of knowledge, is going to be very different from industrial or more professional development made by companies.

Using groups of students is certainly an advantage since they always try to make everything right and as it should be according to the standards of software development, and at the same time, some of them find it hard to come up or even follow a team standard. In a more professional environment, this may not happen as easily, which raises some flags for our system.

With this in mind, we are giving in to the mistake of limiting ourselves to only one type of user for our system, or analysing the use of only one of our target audiences and not looking to the rest of them.

One of our main challenges for this research, which is precisely one of the main question marks we are trying to break through, is the abstract factor that comes along with software development. There is no standard nor any formula on how a system should be developed, given the number of factors that may change and influence one project in comparison to another, such as the client, the requirements, the frameworks, or even the team itself working on it.

So, with all these infinite possibilities, our true challenge is to see which tests could be enough to gather the umbrella of possible audiences. If besides the tests with students we were able to get a more professional scope to test our solution, would that still be enough? In either case, we would be neglecting many other user types.

One small solution for this problem is already incorporated into the architecture of our system. As we will see in 5.2, the system will have an engine rule integrated that will parse through the data and see what kind of activities have taken place throughout the project. These rules may be edited in a small text file in a certain type of format with clear instructions on how to write the rules in order for them to work. That way, any project manager may dictate how each activity may be identified and how each team member should make the commits and/or tickets in order to be successfully recorded.

Even though it's not a perfect solution, it's good enough so that any project manager, either a student or someone in a more professional scope may adapt these rules at will to some that fit better their purpose.

## **Misinterpreted Data**

Another threat to the validity of our system is the misinterpretation of data.

When we were testing our system, at least one of the sets of students had absolutely no contact with us during the process, that is, we tested the rule engine system when they had already finished the project, and so we had only the information of the data and no clear explanation if the interpretation we made out of it was correct or not. Not only that but there were also some cases in which we couldn't give any kind of meaning to them, leaving us with outliers with no sense.

Such a threat could've been avoided if we had worked in close proximity, but since their project and our rule engine development didn't overlap in time, there was no way of turning this around. This is also proof of the importance of teams following a standardized software development along with good organization and management among themselves.

One example may be observed with the Quirked Up Software team, one of the first sets of students and the one on which it was most tested. In the first initialization and configuration commits, they created the following directory: *'PROD/README-PROD.md'*. Unfortunately, we couldn't determine what the keyword *'PROD'* could mean neither in English nor in Portuguese (their mother tongue). Besides, after going to look into the directory to try to figure it out according to its contents, we discovered that it suffered no other changes, that is, it stood until the end as an empty directory. In the end, we decided it was a meaningless outlier and categorized it as a 'Housekeeping' activity (more on that in Appendix E).

## 6.4 Summary

In this chapter, we proceeded to make some tests and the analysis of the results.

As we could see, of the 3 groups selected, Quirked Up Software and KahUC were those with the best results and with a good model for their process. The other group, byDeiCrowd, even though the results weren't as fulfilling, were a good example to show the importance of establishing a standard or using the same format when registering events at all times during the project.

In the end of this chapter we presented some possible threats to the validity of our conclusions that have been considered.

In the following chapter we will present the concluding remarks for this thesis and further future work that we propose.

# Chapter 7

## Conclusion

After all the work and research presented in the previous chapters, in the current one we will proceed to make a summary of the most important ideas to keep in mind after the reading.

We will also take this opportunity to propose some future work that may take place in order to deepen this research.

### 7.1 Overall View

At the beginning of this thesis, after a brief introduction of the work that was followed through, we make a short overview of what is process mining (section 3.2.4) so that the reader can become acquainted with some of the terminologies that may be used throughout this report. We talk about the different techniques that can be used such as process discovery, conformance checking and process improvement, and explored some examples of real-life cases where process mining was used to improve or analyze processes. After this contextualization, we narrow down our scope to Process Mining Software Repositories (PMSR) where the main focus of our research will be. This area emphasizes trying to apply process mining to software repositories. As seen before, this may pose a great challenge, not only because of the different activities done inside and outside of the repository but also for the fact that much can be done in a single file of code.

Further ahead, we give an overview of the methodologies used (section 3). We present both Research Question (RQ)s that we are trying to solve - if it's possible to turn raw events into categorized activities and, if so, if in turn these can be used to model the process followed by that Repository - and we explain how we planned to go through these. There is also a risk analysis in order to enumerate some threats to the validity of our results that we kept in mind when designing the solution.

In light of this, we presented our aim for this thesis: not only to model processes but also to try to see some insights into the development that is being undertaken by the team. Taking advantage of the data retrieved from a GitLab

or GitHub repository, we want to be able to track down activities throughout the process and be capable of clearly identifying them in order to use them as a basis to model the process of the data extracted from the Repository.

For that purpose, in the first iteration of the thesis, we propose an initial system with a set of functional requirements already listed in Appendix B. We also point out the importance to have some Non-Functional Requirement (NFR)s, namely the maintainability of our system in order to allow the investigation to go on and be used by others. Even though it's not as important, the extensibility of our solution can also be an important factor regarding the span of systems from where it can retrieve repositories.

Given these requirements, we concluded that the architecture we are going to need is of the type pipe and filter: after we retrieved all the data from the repository in one component, we filtered this data in an event queue with all the activities formatted in order to comply to the system's needs. In the end, this event queue may be exported as a *csv* file which can be used as input in any user interface. With this, the user may be able to model the data we were able to filter and see any desirable statistics in the chosen program.

After describing the architecture, the validation steps and results were also described in chapter 4. This was done by sending a Google Forms form to the students of the college groups that we used to validate our current tool. Besides some outliers that we were not predicting, most of the feedback was already expected and highly related to the problems and concerns we described previously.

Moving on to the second iteration of our thesis, we first started by creating a system named *playground* that we used to analyze and find patterns in the raw data retrieved from a repository. This was useful for us to conclude that we would need to build some rules engine interface in the pipe-and-filter, but also to write said rules.

We then described the separate interfaces that were part of the architecture. We have the *Fetching Data* interface where we fetch the data from the repository; the *Aggregator* interface where not only do we save the data in a *xml* file, but also save the timestamp from the last recorded data to ease the next fetch; the *Data Fusion* where we merge together all the different types of events into a single file chronologically; the *Rules Engine* where we parse the events through the rules created in order to decide which activity took place; and finally the *Process Miner* interface where, using the heuristic miner algorithm, we model the process according to the activities found from the events.

In the last chapter, we proceeded to make the discussion of the final results obtained for the last system. After a brief analysis of the testing groups, we concluded that the best candidates to test the system were Quirked Up Software, KahUC and byDeiCrowd. The overall results were positive with a low rate of events that weren't caught by any rules. One particularly noteworthy finding was the importance of a team following a specific standard or pattern when saving an event.

## 7.2 Future Work

One of our first priorities in the further evolution of this work would be to go through the problems mentioned in section 4.7 and look further into them to see how we could avoid them. Once these have been taken care of and the final requirements implemented we will be close to singling out process activities and tracking them down in the timeline.

We would also advise a collaboration with some more professional repositories to get some real feedback and some more serious reference. With this, we may be able to understand the adjustments that may still be needed to make, either to our system or to the rules.

In case this system starts being used with current (as in projects that didn't come to an end) and larger repositories, or is used as a commercial tool for real-life clients, then we would advise the implementation of webhooks. These are mechanisms that can be used as triggers to fetch new data automatically from the repository once it's updated.

Since we worked on projects from students, one particularly interesting use of our system could be to aid students' projects with many members. That way, they could easily keep track of their interactions, how their work was evolving and if there was any hidden problem that they should keep an eye on.

Overall, this project was a unique opportunity to get to know and improve our knowledge not only regarding process mining, but also when it comes to understand software development processes, especially when it comes to the importance of sound and systematic adoption of good practices. This research opens the way for useful tools for teams and individuals.



# References

- [1] Wil Aalst. Process mining: Discovering and improving spaghetti and lasagna processes. *Atmospheric Environment - ATMOS ENVIRON*, pages 1–7, 2011.
- [2] Wil Van Der Aalst. Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems*, 3(2):7:1–7:17, 2012.
- [3] Wil Van Der Aalst. *Process Mining - Data Science in Action, 2nd Edition*. Springer, Eindhoven, 2 edition, 2016.
- [4] AltexSoft. Non-functional requirements: Examples, types, how to approach. <https://www.altexsoft.com/blog/non-functional-requirements/>. Accessed: 07.12.2022.
- [5] Saimir Bala, Paul Kneringer, and Jan Mendling. Discovering activities in software development processes. *PoEM'20 Forum: 13th IFIP WG 8.1 Working Conference on the Practice of Enterprise Modelling*, pages 54–63, 2020.
- [6] R. Bose, R. S. Mans, and W. M. van der Aalst. Wanna improve process mining results? *Computational Intelligence and Data Mining (CIDM)*, pages 127–134, 2013.
- [7] c3d3. Process mining with python tutorial: A healthcare application — part 2. [https://medium.com/@c3\\_62722/process-mining-with-python-tutorial-a-healthcare-application-part-2-4cf57053421f](https://medium.com/@c3_62722/process-mining-with-python-tutorial-a-healthcare-application-part-2-4cf57053421f). Accessed: 25.4.2023.
- [8] Abhinav Chola. Jira python integration: 2 easy methods. <https://hevodata.com/learn/jira-python-integration/>. Accessed: 29.1.2023.
- [9] GitHub. Github docs. <https://docs.github.com/en>. Accessed: 7.12.2022.
- [10] GitLab. Gitlab docs. <https://docs.gitlab.com/ee/>. Accessed: 7.12.2022.
- [11] GitLab. Why gitlab - the one devops platform. <https://about.gitlab.com/why-gitlab/#sdlc-features>. Accessed: 3.11.2022.
- [12] Alexey Grigorev. Alpha algorithm. [http://mlwiki.org/index.php/Alpha\\_Algorithm](http://mlwiki.org/index.php/Alpha_Algorithm). Accessed: 19.10.2022.

- [13] M. Hosseinpour and M. Jans. Categorizing identified deviations for auditing. *CEUR Workshop Proceedings*, vol. 1757, pages 125–129, 2016.
- [14] Mieke Jans, Michael Alles, and Miklos Vasarhelyi. The case for process mining in auditing: Sources of value added and areas of application. *International Journal of Accounting Information Systems*, pages 1–20, 2012.
- [15] Sabrina Joos and Thorsten Krüger. Ultimate list of process mining case studies in manufacturing. <https://workerbase.com/resources/process-mining-manufacturing-case-studies>. Accessed: 5.1.2023.
- [16] Anthony J. Lattanze. *Architecting Software Intensive Systems - A Practitioner's Guide*. Auerbach, 2009.
- [17] MARCIN NOWAK. What is a rules engine and why do you need it? <https://www.hyperon.io/blog/what-is-a-rules-engine>. Accessed: 10.5.2023.
- [18] University of Essex. Research risk assessment. <https://www.essex.ac.uk/student/postgraduate-research/research-risk-assessment>. Accessed: 19.12.2022.
- [19] Sayeda Haifa Perveez. What is git: Features, command and workflow in git. [https://www.simplilearn.com/tutorials/git-tutorial/what-is-git#what\\_is\\_git](https://www.simplilearn.com/tutorials/git-tutorial/what-is-git#what_is_git). Accessed: 22.12.2022.
- [20] PLACEMENT\_CHAHIYE. Empirical research in software engineering. <https://www.geeksforgeeks.org/empirical-research-in-software-engineering/>. Accessed: 25.4.2023.
- [21] Gauvain Pocentek. python-gitlab. <https://python-gitlab.readthedocs.io/en/stable/index.html>. Accessed: 3.11.2022.
- [22] Eric Rojas, Jorge Munoz-Gama, Marcos Sepúlveda, and Daniel Capurro. Process mining in healthcare: A literature review. *Journal of Biomedical Informatics*, pages 224–236, 2016.
- [23] Paula Rome. What are non functional requirements — with examples. <https://www.perforce.com/blog/alm/what-are-non-functional-requirements-examples>. Accessed: 19.10.2022.
- [24] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. van der Aalst. Process mining can be applied to software too! *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2014.
- [25] R. Santos, T. C. Oliveira, and et al. Mining software development process variations. *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. ACM, pages 1657–1660, 2015.



- 
- [26] M.L. Sebu and H. Ciocarlie. Applied process mining in software development. *2014 IEEE 9th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, pages 55–60, 2014.
- [27] Dr. Varshita Sher. Process mining to assess app user behavior from clickstream data. <https://towardsdatascience.com/process-mining-to-assess-app-user-behavior-from-clickstream-data-8e53a71428a4>. Accessed: 25.4.2023.
- [28] Ian Sommerville. *Software Engineering*. Pearson, Essex, England, 10 edition, 2016.
- [29] Davide Spadini. Pydriller documentation! <https://pydriller.readthedocs.io/en/latest/>. Accessed: 3.11.2022.
- [30] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. PyDriller: Python framework for mining software repositories. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018*, pages 908–911, New York, NY, USA, 2018. Association for Computing Machinery.
- [31] B. F. van Dongen, A. K. Alves de Medeiros, and L. Wen. Process mining: Overview and outlook of petri net discovery algorithms. *Transactions on Petri Nets and Other Models of Concurrency II*, pages 225–242, 2009.
- [32] Rohan Vats. Github vs gitlab: Difference between github and gitlab. <https://www.upgrad.com/blog/github-vs-gitlab-difference-between-github-and-gitlab/>. Accessed: 22.12.2022.
- [33] J. Van Der Werf, H. Verbeek, and W. Van Der Aalst. Context-aware compliance checking. *Lecture Notes in Computer Science, vol. 7481*, pages 98–113, 2012.
- [34] Claes Wohlin. Empirical software engineering research with industry: Top 10 challenges. *Proceedings of the First International Workshop on Conducting Empirical Studies in Industry (CESI)*, pages 43–46, 2013.
- [35] Claes Wohlin, Martin Höst, and Kennet Henningsson. 13 empirical research methods in web and software engineering. *Empirical Methods and Studies in Software Engineering: Experiences*, pages 7–23, 2003.
- [36] UX World. 3-click rule and usability. <https://uxdworld.com/2020/01/28/3-click-rule-and-usability/>. Accessed: 2.11.2022.



# Appendices



# Appendix A

## Gantt Diagrams

In the current appendix, the reader will be presented with two different Gantt Diagrams <sup>1</sup>.

The first chart A.1 is related to the time period of the first Semester, that is, from September 2022 to January 2023. During this interval, there was some research done at the beginning of this thesis in order to get more acquainted with process mining. After that, we started writing down the requirements and use cases for our program and, in the end, we developed our alpha-system and tested it.

During this time the development of the system was done in parallel with the writing of this thesis. Due to other classes, there may be some gaps in the chart.

The second chart A.2 is related to the second half of our thesis, that is, from the end of January 2023 to July 2023. During this period, our main focus was developing the final system, with a greater aim towards the rule engine interface. Even though there was some research and some writing of the thesis parallel with the development, most of the writing was saved to the end of the time period.

---

<sup>1</sup><https://www.onlinegantt.com/#/gantt>

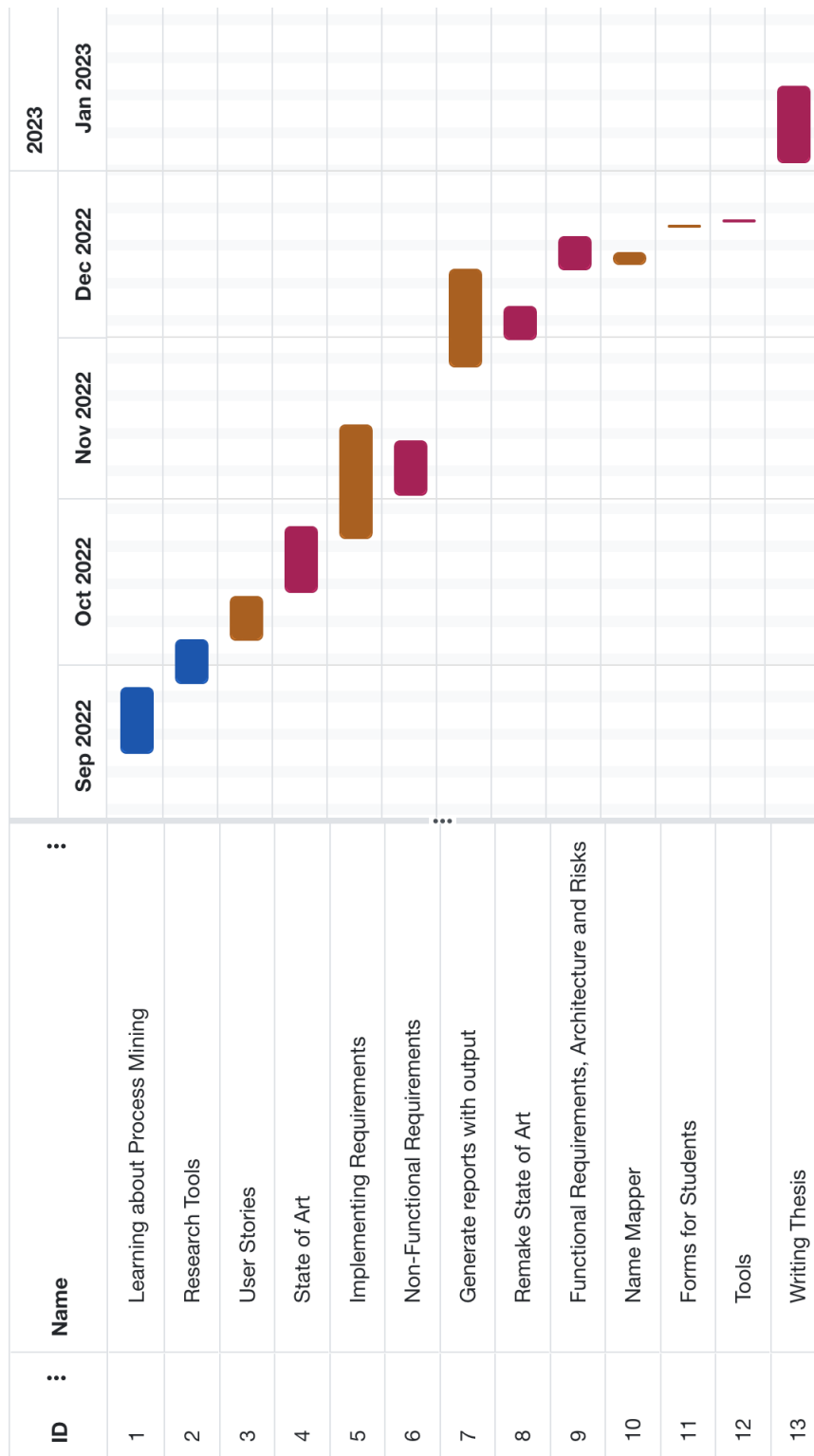


Figure A.1: Gantt Chart from the 1st Semester (from September 2022 to January 2023) - The blue activities are related to research activities, the brown ones are related to development activities, and the bordeaux ones are related to the writing of this thesis

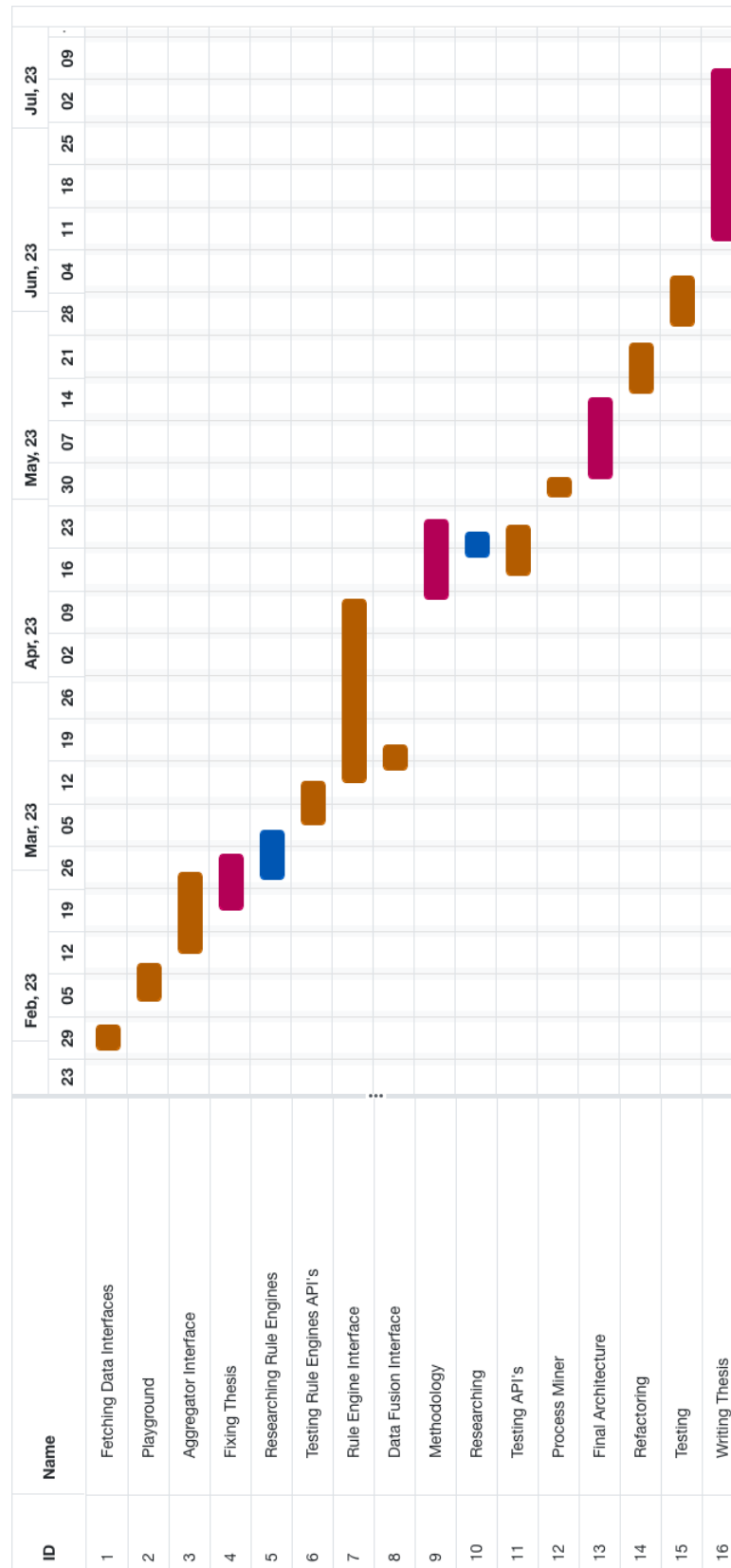


Figure A.2: Gantt Chart from the 2nd Semester (from the end of January 2023 to July 2023) - The blue activities are related to research activities, the brown ones are related to development activities, and the bordeaux ones are related to the writing of this thesis





# Appendix B

## User Stories

In the following tables are presented the user stories for all the requirements identified for the alpha-version.

### User Story 1

US #1	Title Multiple Repositories	Priority Low
<b>User Story</b> As a project manager, I want to be able to add at least a repository to the system In order to be able to get a report on more than one repository		
<b>Validation Criteria:</b> If want to be able to get data on more than one repository When I insert each input data Then I will get a report for each one of the repositories I added.		

Table B.1: User Story 1 - Multiple Repositories

### User Story 2

US #2	Title See raw information	Priority High
<b>User Story</b> As an activities tracker, I want to see the raw information of each commit In order to be able to perform debugging in my data collection And create new rules to transform raw events into semantically rich activities		
<b>Validation Criteria:</b> If want to be able to see the raw information of a repository When I insert new input data in the target repository Then I will see a raw view of the data inserted.		

Table B.2: User Story 2 - See raw information

### User Story 3

US #3	Title Mapping objects	Priority High
<b>User Story</b> As a process engineer, I want to see the mapping between raw events and activities in a project In order to be able to understand, debug and see the process the team is following		
<b>Validation Criteria:</b> If I want to be able to see how my team is working, When generating the mapping from raw event to activity Then I will see how my team is operating according to the retrieved logs.		

Table B.3: User Story 3 - Mapping objects

### User Story 4

US #4	Title Author's Development Stage	Priority Medium
<b>User Story</b> As a project manager, I want to see in which development stage <sup>1</sup> each author is working on In order to understand the team structure		
<b>Validation Criteria:</b> If I want to be able to see how the team is structured, When generating the repository report I will see how my team is organized.		

Table B.4: User Story 4 - Author's Development Stage

### User Story 5

US #5	Title Files Per Author	Priority Medium
<b>User Story</b> As a project manager, I want to see in which files each author is working on In order to map authorship to artifacts		
<b>Validation Criteria:</b> If I want to be able to see which files each author worked on, When generating the repository report I will see a list of files per author.		

Table B.5: User Story 5 - Files Per Author

---

<sup>1</sup>As *development stage* we consider the separate teams of Business, Requirements, Architecture and Design, Development, Testing, and Quality Assurance

**User Story 6**

US #6	Title Author Per Files	Priority Medium
<b>User Story</b> As a project manager, I want to see which author is working in each file In order to understand each artifact's potential entropy		
<b>Validation Criteria:</b> If I want to be able to see each artifact's entropy, When generating the repository report I will see a list of authors per artifact.		

Table B.6: User Story 6 - Authors Per File

**User Story 7**

US #7	Title New Sprint	Priority Medium
<b>User Story</b> As a project manager, I want to identify the end or the beginning of a new sprint In order to track progress And relate features to deliverables		
<b>Validation Criteria:</b> If I want to be able to identify the beginning or the end of a sprint, When loading the event data I will see the timeline on which each sprint took place.		

Table B.7: User Story 7 - New Sprint

**User Story 8**

US #8	Title Critical Files	Priority Medium
<b>User Story</b> As a quality manager, I want to identify critical files In order to pinpoint potential defect hotspots		
<b>Validation Criteria:</b> If I want to be able to identify potential hazardous files, When loading the event data I will be able to see the critical files in the project.		

Table B.8: User Story 8 - Critical Files

### User Story 9

US #9	Title Active Members	Priority Medium
<b>User Story</b> As a project manager, I want to see which members where active In order to manage my team		
<b>Validation Criteria:</b> If I want to be able to manage my team, When loading the event data I will be able to see the members that were active.		

Table B.9: User Story 9 - Active Members

### User Story 10

US #10	Title Files with more Authors	Priority Medium
<b>User Story</b> As a quality manager, I want to know which files have more authors In order to identify potential defect hotspots		
<b>Validation Criteria:</b> If I want to be able to identify potential hazardous files, When loading the event data I will be able to see the files with more authors.		

Table B.10: User Story 10 - Files with more Authors

### User Story 11

US #11	Title Files with only one Author	Priority Low
<b>User Story</b> As a quality manager, I want to know which files have only one author In order to identify potential <i>trash</i> files		
<b>Validation Criteria:</b> If I want to be able to identify potential "no adding value" files, When loading the event data I will be able to see the files with only one author.		

Table B.11: User Story 11 - Files with only one author

## User Story 12

US #12	Title More Commits	Priority Medium
<b>User Story</b> As a project manager, I want to know which members made more commits in the last sprint In order to detect possible effort spikes		
<b>Validation Criteria:</b> If I want to be able to identify potential effort spikes in the team, When loading the event data I will be able to identify the authors that made more commits.		

Table B.12: User Story 12 - More Commits

## User Story 13

US #13	Title Less Commits	Priority Medium
<b>User Story</b> As a project manager, I want to know which members made fewer commits in the last sprint In order to detect a possible <i>leech</i> in the team		
<b>Validation Criteria:</b> If I want to be able to identify who is working less in the team, When loading the event data I will be able to identify the authors that made fewer commits.		

Table B.13: User Story 13 - Less Commits

## User Story 14

US #14	Title Task Monitoring	Priority High
<b>User Story</b> As a project manager, I want to know which activity/task was developed/worked on in each commit In order to monitor an issue evolution		
<b>Validation Criteria:</b> If I want to monitor a task, When loading the event data I will be able to know if it was completed or if it's being taken care of.		

Table B.14: User Story 14 - Task Monitoring

### User Story 15

US #15	Title Completed Issues	Priority Medium
<b>User Story</b> As a project manager, I want to know which issues were completed In order to manage progress		
<b>Validation Criteria:</b> If I want to manage progress, When loading the event data I will be able to see which issues were already completed.		

Table B.15: User Story 15 - Completed Issues

### User Story 16

US #16	Title Daily Commit	Priority Low
<b>User Story</b> As a team leader, I want to know which team members made a push at the end of the day In order to make sure the day's work is being updated And check the author's daily hours of work		
<b>Validation Criteria:</b> Track the daily progress of a team member, When loading the event data I will be able to see which were the ones who made a commit in the previous day.		

Table B.16: User Story 16 - Daily Commit

### User Story 17

US #17	Title Issues with no Assignee	Priority Medium
<b>User Story</b> As an integrator, I want to know if there is any issue without a responsible In order to assign it to someone to make sure it's worked on		
<b>Validation Criteria:</b> To make sure every issue is going to be managed, When loading the event data I will be able to see if there is any issue with no assignee.		

Table B.17: User Story 17 - Issues with no Assignee

## User Story 18

US #18	Title Development Type	Priority Medium
<b>User Story</b> As a team leader, I want to know what type of software development <sup>2</sup> is being followed In order to make sure the team is working as it should		
<b>Validation Criteria:</b> To make sure the team is working accordingly as planned, When loading the event data I will be able to see what type of software development is being developed		

Table B.18: User Story 18 - Development Type

## User Story 19

US #19	Title Commits Per User	Priority Medium
<b>User Story</b> As a project manager, I want to have a sorted list with users according to their number of commits In order to see the differences between each element of the team		
<b>Validation Criteria:</b> To understand the different effort between each element of the team, When loading the event data I will be able to see a sorted list of the team members according to their number of commits		

Table B.19: User Story 19 - Commits Per User

---

<sup>2</sup>As *software development* we consider the different types of software development such as the use of Agile Methods or the Waterfall Model





# Appendix C

## Other Non-Functional Requirements

Besides the main Non-Functional Requirement (NFR) identified for the alpha-version, there were also some that, even though are not as important to have, would add even more value.

### Usability

The **Usability** [23][36] of a system describes the ease that each user has when using the system and what has an influence in it. Whether it's the interface, the styling, or even the way information is displayed, everything can be a factor in increasing or decreasing the system's Usability.

Even though this system will be developed in a research context where the analysis and interpretation of data is the number one priority, Usability is also called upon. If the data is presented in such a way that makes it unreadable for the user then there is no knowing either if the system is working correctly or what data it is showing. As such, however, if a dashboard gets to be designed to show the results, Usability is desirable with a clean interface and a clear display of information.

### Security

Just like in any other system, **Security** [23] is one of the most important factors to consider when developing software. It ensures that the system is protected against third-party attacks and that the information used is handled accordingly to prevent any leak.

As stated before, this system will be used to analyze whole projects and files from companies and teams that wish to see how their work is progressing and what is being done. If this kind of information gets intercepted by a middle-man, is leaked to the outside, or is even accessed by any third party that got access to the platform, it could be fatal to the company or give an upper hand to their competitors.

Just like Usability, given that this is research work, Security will be considered a lower priority NFR: the main focus is research and to try to achieve such process translator. But, being one of the most important NFR's and the one that usually raises more concerns, when used by final users it should be accounted for.

### **Performance**

**Performance** [23] challenges the system's speed responding to user interactions and the speed when performing operations.

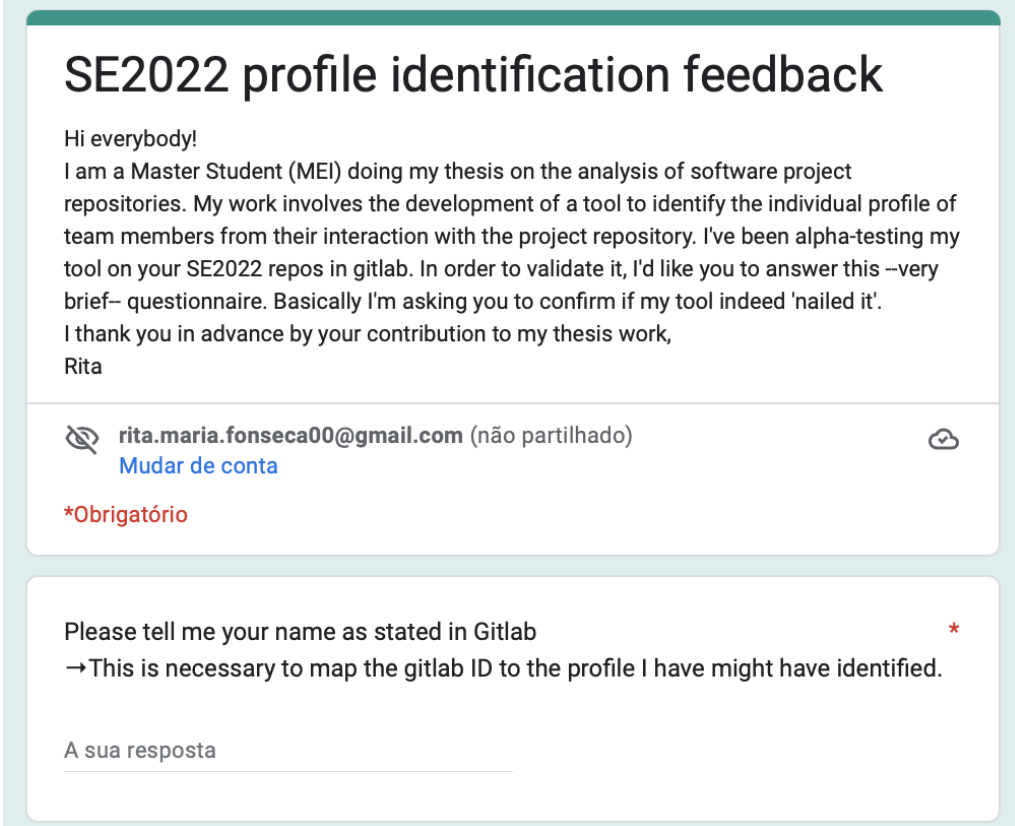
Given the amount of data that will be analyzed and then displayed in the dashboard, even if the code is as efficient and optimized as possible, it's expected that the larger the repository, the longer the wait to have the results. A possible turnaround may be that, while the system is working in the background constantly processing new data that keeps arriving, the user will be given an updated view of the process at a certain hour of the day. As mentioned before, since the focus is on the bigger picture and the history of the process instead of what is happening at that exact moment, a delay of a few hours will not drastically change the perception of the project.

# Appendix D

## Student Feedback Form

In the following figures is presented the set of questions given to the students in order to validate the results obtained by the alpha-system.

In the first section of the forms, after a brief introduction to give some context as shown in Figure D.1, we ask the students how they are identified in GitLab (the online platform where they stored the repository), in which stage of development they were more involved, and to which class they belonged to (Figure D.2).



The image shows a screenshot of a feedback form. The title is "SE2022 profile identification feedback". The text reads: "Hi everybody! I am a Master Student (MEI) doing my thesis on the analysis of software project repositories. My work involves the development of a tool to identify the individual profile of team members from their interaction with the project repository. I've been alpha-testing my tool on your SE2022 repos in gitlab. In order to validate it, I'd like you to answer this --very brief-- questionnaire. Basically I'm asking you to confirm if my tool indeed 'nailed it'. I thank you in advance by your contribution to my thesis work, Rita". Below the text is a contact information section showing an email address "rita.maria.fonseca00@gmail.com (não partilhado)" with a link "Mudar de conta" and a cloud icon. A red asterisk indicates a required field. The next section asks for the student's name as stated in GitLab, with a note: "→ This is necessary to map the gitlab ID to the profile I have might have identified." Below this is a text input field labeled "A sua resposta".

Figure D.1: Introduction of the forms and request of the students' username

The image shows two separate survey questions, each enclosed in a light blue rounded rectangular box. The first question asks about involvement in various development activities, with a list of seven options: Business, Requirements, Design, Development, Testing, Quality, and Other. Each option is preceded by an unchecked checkbox. The second question asks for the respondent's PL Class, with six radio button options labeled PL1 through PL6. Both questions include a red asterisk in the top right corner, indicating they are mandatory. Explanatory text follows each question, clarifying the purpose of the data collection.

In which activities were you more involved? \*

→ This is to compare your profile as stated by you (the "truth") with the automatically extracted profile.

- Business
- Requirements
- Design
- Development
- Testing
- Quality
- Other

What was your PL Class? \*

→ you shall be redirected to the automatically extracted profiles (my analysis was performed by PL)

- PL1
- PL2
- PL3
- PL4
- PL5
- PL6

Figure D.2: Questions regarding the stage of development and the class they belonged to

Afterwards, each student is redirected to a second section according to the class they mentioned. In the figures below, in order to present an example, we show the section related to a student from the class PL1 - team Quizipedia.

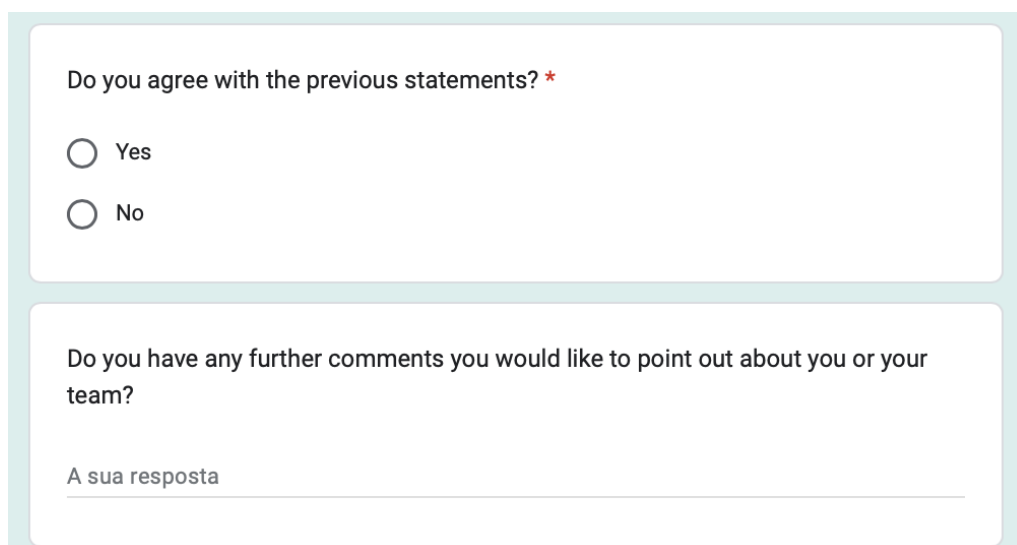
In the beginning, each student will be confronted with the results obtained by the alpha-version of our system (Figure D.3). They will be able to analyse the correlations between a person and a development stage, the more active and less active members.



Figure D.3: Display of the results of the Quizipedia team

Later, they will have some space to state if they agree or not with the present results, to give their opinion regarding what they thought about them or even

some additional comments regarding the team (Figure D.4).



Do you agree with the previous statements? \*

Yes

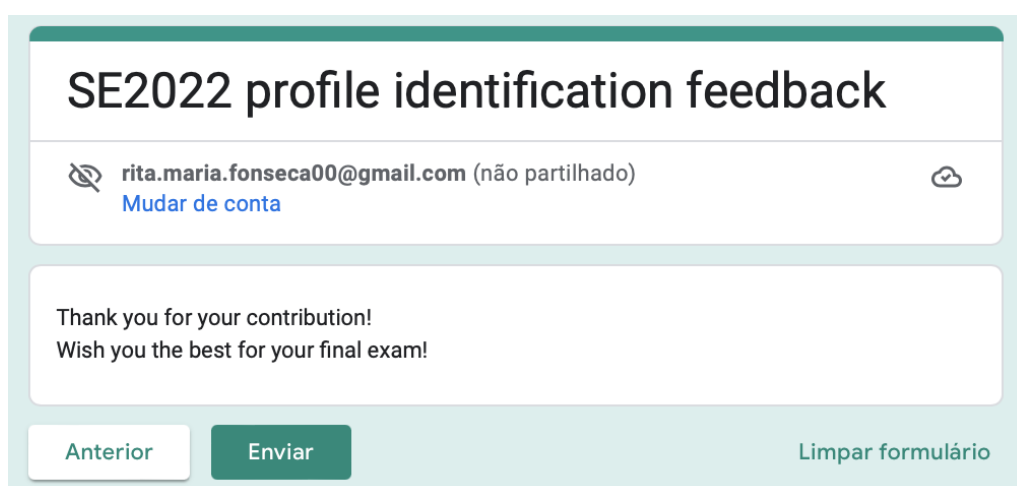
No

Do you have any further comments you would like to point out about you or your team?



A sua resposta

Figure D.4: A space for some additional comments from the participant

In the end, there's a brief note thanking the participant for their contribution to our research (Figure D.5).



**SE2022 profile identification feedback**

 rita.maria.fonseca00@gmail.com (não partilhado) 

[Mudar de conta](#)

Thank you for your contribution!  
Wish you the best for your final exam!

[Anterior](#) [Enviar](#) [Limpar formulário](#)

Figure D.5: Final Note

# Appendix E

## Rules Table

In the current appendix, we present the rules [17] created for our system to identify the different activities that took place during the project. These are saved in a separate file *Rules.txt* that is given as input to our Rule Engine interface (see section 5.2.4 and figure 5.1). We present both the syntax and the logic of each rule type as well as a basic example(s).

If none of these rules can be applied to the input, then the output will be *Housekeeping*.

### **contains(arg,S)=act**

This rule allows us to verify if the argument *arg* of the event given as input contains the string *S*. If it does then the output will be the activity *act*. If we take a look at the example below:

```
contains(path, mockup) =Mockups
```

This rule states that if the *path* of a modified file during this event contains the string *mockup*, then the activity identified is *Mockups*.

### **is(arg,S)=act**

This rule allows us to verify if the argument *arg* of the event given as input is equal to the string *S*. If it is, then the output will be the activity *act*. If we take a look at the example below:

```
is(author, Rita Fonseca) =Thesis
```

This rule states that if the *author* of the current event is *Rita Fonseca*, then the activity identified is *Thesis*.

This rule is especially useful to identify which kind of event is being analysed:

```
is(commit, ) =Commit
```

In this case, we want to know if the event is a `commit` or not. This specific use of the `is` rule may be useful when filtering events and reducing the overhead of the decision, since there may be some activities that only occur during a specific type of event (like a `commit` or an `issue`).

Another particular use of this rule may be seen below:

```
is(files, None)=act
```

In this scenario, we are merely looking for events where no file was modified.

### **not(arg,S)=act**

This rule allows us to verify if the argument `arg` of the event given as input doesn't contain or isn't equal to the string `S`. If it doesn't, then the output will be the activity `act`. If we take a look at the example below:

```
not(desc, merge)=There was no merge
```

This rule states that if the `desc` (also known as the description of the event) does not contain the string `merge`, then we know that there was no merging of branches during this event.

### **ends(arg,S)=act**

This rule allows us to verify if the argument `arg` of the event given as input ends with the string `S`. If it does then the output will be the activity `act`.

As may be seen in the following example, this rule is particularly useful when we want to verify the type of the file that was modified:

```
ends(path, .html)=Html file updated
```

That way, since the `path` of a modified file during this event ends in `.html`, then we may infer that an `Html` file was updated. A similar approach is used by Saimir Bala et al. in [5].

### **r1-AND-r2=act or r1-OR-r2=act**

Besides the *atomic* rules mentioned above, we also use the logic `and` and the logic `or` operations to create more complex rules.

When considering the example below, we have an `is` rule and a `contains` rule both connected by a logic `and`. In this particular case, we want to check if the event is an `issue` and if the `title` of the issue contains the string `bug`. If both these cases are true (according to the rules of the logic `and` operator) then the activity is `Fixing Bugs`.

```
is(issue, )-AND-contains(title, bug)=Fixing Bugs
```



Similarly, if we take a look into a rule with a logic *or*:

`contains (path, profile) -OR- contains (path, personal) =`  
`↔ Personal Profiles`

we may observe that if at least one of these is true (according to the rules of the logic *or* operator) then the activity is `Personal Profiles`.

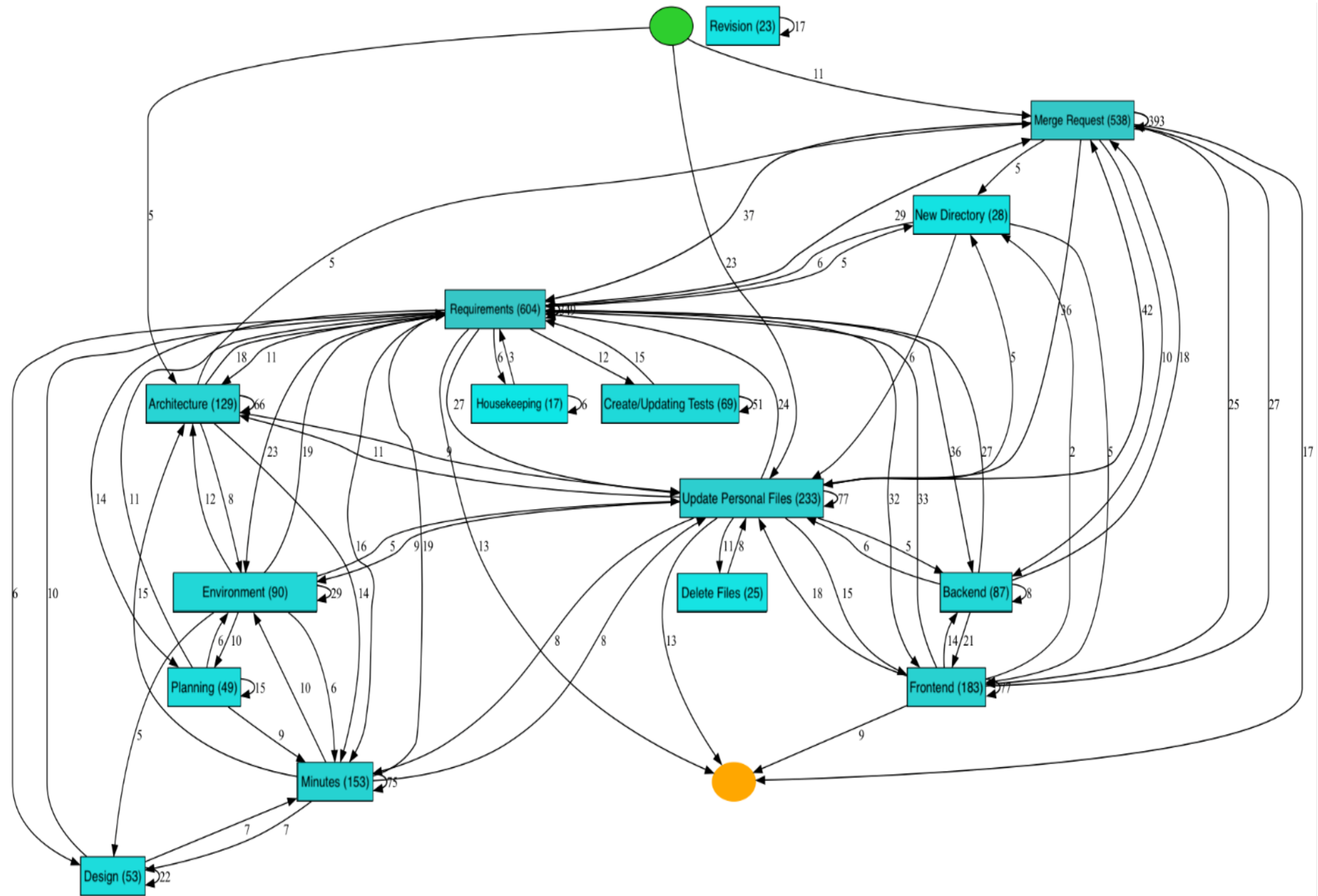


# Appendix F

## Modelled Processes

In this appendix, we present the real modelled processes for each one of the final testing groups, respectively, Quirkep Up Software, KahUC and byDeiCrowd.

Just like the simplified versions of the models, the start of the process is represented by a green ellipse and the end by an orange one. The activities may then be found in the blue boxes, the higher the frequency, the darker the shade of blue.



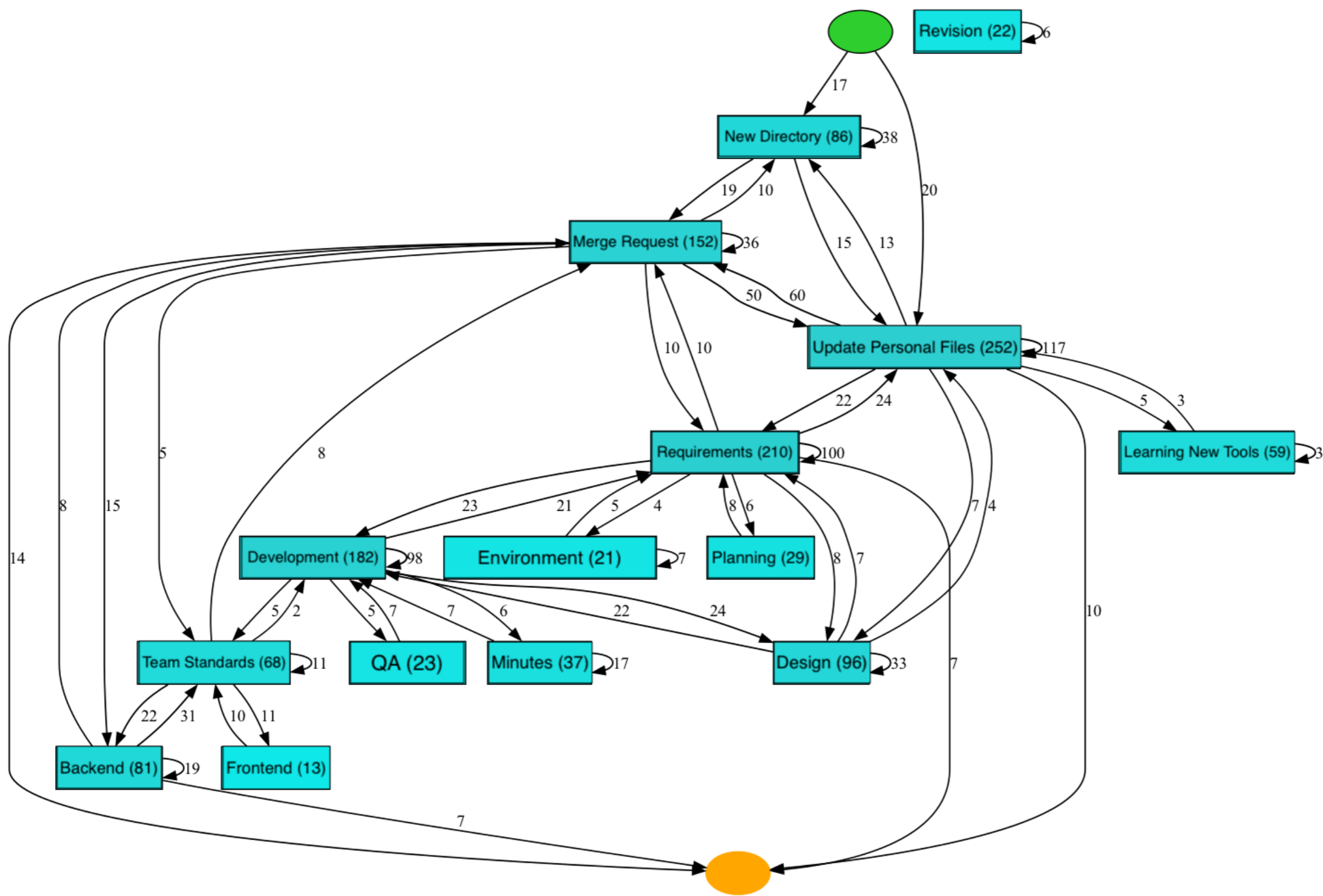


Figure F.2: Real process modeled for KahUC

