# Assessing Software Development Teams' Efficiency using Process Mining

João Caldeira
Fernando Brito e Abreu
José Reis
ISTAR-IUL
Instituto Universitário de Lisboa (ISCTE-IUL)
Lisboa, Portugal
{jcppc, fba, jvprs}@iscte-iul.pt

Jorge Cardoso
CISUC, Dept. of Informatics Engineering
University of Coimbra, Portugal
Huawei Munich Research Center, Germany
jcardoso@dei.uc.pt

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

***Objective.*** **The overall goal of this study is to improve the knowledge on how to measure efficiency in development teams where a great deal of variability may exist due to the human-factor. The main focus is on the discovery of the underlying processes and compare them in terms of efficiency and effectiveness. By doing so, we expect to reveal potentially hidden costs and risks, so that corrective actions may take place on a timely manner during the software project life cycle.**

***Method.*** **Several independent teams of Java programmers, using the Eclipse IDE, were assigned the same software quality task, related to code smells detection for identifying refactoring opportunities and the quality of the outcomes were assessed by independent experts. The events corresponding to the activity of each team upon the IDE, while performing the given task, were captured. Then, we used process mining techniques to discover development process models, evaluate their quality and compare variants against a reference model used as "best practice".**

***Results.*** **Teams whose process model was less complex, had the best outcomes and vice-versa. Comparing less complex process variants with the "best practice" process, showed that they were also the ones with less differences in the control-flow perspective, based on activities frequencies. We have also determined which teams were most efficient through process analysis.**

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

## I. Introduction

Inaccurate planning and/or project plan deviations cause substantial financial losses on software development projects [1]. Further, constant inaccuracies and losses may degrade the reputation of development teams as they become perceived as non-compliant to organizational plans and budget forecasts.

Critical success factors have always been at the forefront of the research related with software development projects [2]–[5]. The existence of a vast literature about this topic, either about successes [1] or failures [6], [7], reveals the concerns and doubts that still haunt software development practitioners regarding the efficiency and effectiveness of their own projects.

It is frequently suggested that software projects can be assessed across four perspectives: quality, scope, time and cost [2], which are related with the planning and execution of the project's main activities. Each perspective has its own critical success and failure factors, that can be grouped into five different dimensions: organizational, people, process, technical, and project [3]. In this paper, we will be mainly concerned with the effect of the human factor in process variability.

To start a software development project from scratch is a complex activity on its own [8], specially in the absence of a formalized process or methodology [9] that acts as a referential. Evidences found suggest that in addition to the initial project planning, the way people are organized, the tools they use and the processes they follow are key features for the success or failure of any software project [9]. As for software development, although prescribed process models may exist, projects often do not comply with them, both because each developer or team usually has some freedom to interpret the process and because its compliance is not verified on the run, since it is mainly intangible. As a result, it has been noted that process executions (i.e. projects) often deviate from what was planned [10]. In this paper we bring further evidence that the human factor is a very important source of process variability and the latter will have an impact on process efficiency and effectiveness.

To understand how the process was actually performed by its practitioners, we used process mining techniques. Our

approach, initially proposed in [11], captures events due to practitioners activities executed in the IDE, as well as records which artifacts were used and when, plus additional details on the ecosystem of components supporting the process. This new perspective on software development analytics, that uses process mining, allows the discovery of the actual processes practitioners are following, as well as deviations from those they were supposed to comply to, without the complexity and workload of collecting and merging information from different information systems, such as, source code systems, configuration management repositories or bug tracking tools. As we will show later in this article, we were able to identify the most and less efficient teams, and the ones that drifted less from the same process when executed by an expert.

This paper is organized as follows: on section II we introduce software development analytics challenges and introduce process mining as a natural option to mine software process events' logs; on section III we present the research questions, describe the experiment setup and the methods used for data analysis; next, on section IV, we present the results, elaborate on the main findings and identify threats to validity; finally, in section V, we draw the main conclusions and outline the future work.

## II. Context

### A. Software Development and the IDE

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

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

### B. Process Mining Within the IDE

Process Mining is now a mature discipline with validated techniques producing accurate outcomes on several business domains [12], [13]. A process mining project, if best practices are followed [14], should use goals and event logs as inputs, and produces actions to implement as outputs. The goals may consist of improving some performance indicators, such as time, risks and costs associated to a specific process, or simply to maximize a service level. Actions may be the redesign of a specific project, adjust a current process or, if there is a fluctuation in case volume, one may want to include more resources.

Our short-term goal, whose fulfillment we will describe in this paper, was to assess teams' efficiency by mining the software development process flow and variability that occurs due to the human factor. Our medium-term goal is to provide operational support to software developers, systematically and continuously using current event data to recommend the best activity, adequate resource or action to execute now or in the future. In both cases we will take as input the events emerging from using the IDE. Those events convey a spaghetti-like process [15] mainly because there is a very large number of possible commands/tasks to execute within any IDE that will grow exponentially with the number of installed plugins and, as a consequence, so grows the potential complexity of any mined process.

### C. Related Work

This work is in the crossroads of software development practices and process mining techniques. Much have been said in literature about software development processes [16], [17] and process mining separately [18]. However, elaborating about works combining these two disciplines requires a careful approach, mainly because their intersection is vague in some cases and not fully explained in others. Going back almost a decade, [12] have mined software repositories to extract knowledge about the underlying software processes, and [19], [20] have learned about user behavior from software at runtime. Recently, [21] was able to extract events from `Eclipse` and have discovered, using a process mining tool, basic developers' workflows. Some statistics were computed based on the activities executed and artifacts edited. In [22], the authors have extracted development activities from non-instrumented applications and used machine learning algorithms to infer a set of basic development tasks, but no process mining techniques were used to discover any pattern of application usage. [23] used a semi-automatic approach for analyzing a large dataset of IDE interactions by using cluster analysis [23] to extract usage smells. More recently, [24] used process mining to gain knowledge on software in operation by analyzing the hierarchical events produced by application calls(eg: execution of methods within classes) at runtime. The studies mentioned above, extracted data from several different sources and have used a multitude of statistics methods, machine learning and process mining techniques. However, to the best of our knowledge, none of these works combine data from the IDE utilization with process mining methods with the aim of measuring individuals or teams efficiency. Even in the case of [21], where the approach is similar to ours, nothing was done related to conformance checking on the processes followed by developers, as there was no existing reference model to compare with. Our work introduces a valid

approach for this purpose, and bring a new perspective to software development analytics by filling this gap.

## III. Experiment

We analyzed several teams performing independently the same well-defined task on software quality assurance. To block additional confounding factors in our analysis, all teams had similar backgrounds and performed the same task upon the same software system. To provide authenticity, the task targeted a real-world (large) open-source Java system, the `Jasml (Java Assembling Language)`[1].

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

### A. Research Questions

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

RQ1) To what extent can process mining discover accurate models representing developers' behavior?

RQ2) Can we assess the efficiency of software development teams by using process mining techniques ?

RQ3) The assessment of teams' proficiency, performed by a process expert, is reflected in the quality of the produced models?

### B. Experimental Setup

*1) Subjects:* Subjects were finalists (3rd year) of a BSc degree on computer science at the ISCTE-IUL university, attending a compulsory software engineering course. By this time they had been trained across the same set of almost 30 courses and therefore had similar backgrounds. They worked in teams up to 4 members each and were requested to complete a code-smells detection assignment, aiming at identifying refactoring opportunities, using the `JDeodorant` tool[2]. This tool allowed the detection of four different types of code smells: `Long Method`, `God Class`, `Feature Envy` and `Type Checking` [25]. Once they have detected the occurrences of those code smells, they were required to apply `JDeodorant`'s automatic refactoring features to the critical ones.

*2) Data Collection Instrument:* The `Eclipse` IDE has an internal event bus accessed by the interface `IEventBroker`[3] which is instantiated once the application starts. It contains a publishing service to put data in the bus, whilst the subscriber service reads what's in that bus. This allows a subscriber to read all or part of the events being managed within the IDE. Using this feature we developed an `Eclipse` plugin[4] capable of listening to the actions developers were executing. Before the experiment, the plugin was installed on each subject work environment, and later, all received a unique *username/key* pair as credentials. This method was useful to unlock all the plugin features and allowed us to identify each subject and the corresponding team.

*3) Collected Data:* A sample event instance collected with our plugin is represented in listing 1 in `JSON` format. The field tags are self explanatory.

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

Listing 1: Sample Eclipse Event Instance

*4) Data Storage:* Collected data was stored locally in a CSV file. Whenever Internet connection was available, the same data was stored in the cloud[5]. This storage replication allowed offline and online collection. The final dataset, combining the two different sources, was then loaded into a MySQL database table where the username and event timestamps that formed the table's unique key were used for merging duplicated data. Figure 1 presents a schema of the data collection workflow.

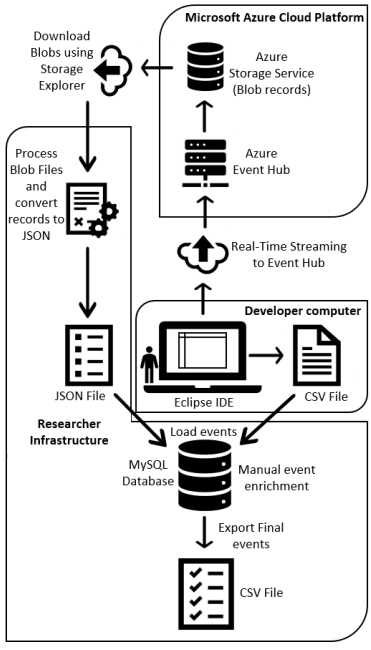


Fig. 1. Experiment Data Collection Workflow

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

[2]https://marketplace.eclipse.org/content/jdeodorant

[3]https://wiki.eclipse.org/Eclipse4/RCP/Event_Model

[4]https://github.com/jcaldeir/iscte-analytics-plugins-repository

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

*5) Data Preparation:* When the software quality task ended, all events stored in the database were converted to the IEEE eXtensible Event Stream (XES) standard format [26] and imported into ProM process mining tool[6]. The following event properties were mapped when converting to XES format:

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

*6) Data Demographics:* As previously mentioned, we only analyzed data collected on the same software system, to block confounding factors. The chosen system was `Jasml (Java Assembling Language)`[7].

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

We present their statistics in Table I. Project events should be seen as fundamental events for the task programmers were requested to execute, and, in a certain way represent the focus their are putting into that work. Generic events are seen as collateral actions not mandatory for the task in hand, but that programmers may need or want to execute to prepare their environment. These generic events somehow convey a lack of focus on the task developers were supposed to execute.

The `REFERENCE`(also identified as `REF.`) team, corresponds to the professor that proposed the task itself. Being the main expert, he executed it in one of the most efficient ways. The full dataset, that includes data on all teams with fine grained data that is not addressed in this paper, is publicly available.[8]

### C. Data Analysis

*1) Context:* Several approaches have been proposed to evaluate the quality of discovered process models. Software quality metrics were mapped to process metrics in [27]. Groups of metrics were also used in [28], [29] to evaluate several dimensions in a process model and, more recently, artifacts were created to support process quality evaluation and perform process variants comparisons [24], [30]. All of these fit within the well defined [15] and generally accepted four dimensions to assess the quality of a model: *fitness, precision, simplicity* and *generalization*.

*2) Process Discovery:* Several well known algorithms exist to discover process models, such as, the $\alpha$-algorithm, the heuristics, genetic and fuzzy miner. However, our need to discover and visualize the processes in multiple ways lead us to choose the `ProM's StateChart Workbench` plugin [24]. This plugin, besides supporting process model discovery using multiple hierarchies and classifiers, also allows to visualize the model as a Sequence Diagram and use notations such as Petri Nets and Process Trees. This plugin is particularly suitable for mining software logs, where an event structure is supposed to exist, but it also supports mining of other so-called generic logs.

Events collected from software in operation (e.g. Java programs) reveals the presence of a hierarchical structure, where methods reside within classes, and classes within packages [31]. The same applies to IDE usage actions, since identified menu options and executed commands belong to a specific category of command options built-in the `Eclipse` framework. Supported by this evidence, we used the Software log Hierarchical discovery method with a Structured Names heuristic, to discover the models based on the fact that the events were using a *category*|*command* structure (e.g. `Eclipse Editor|File Open`). Several perspectives can be used to discover and analyze a business process and the most commonly used are: `Control-Flow`, `Organizational`, `Social` and `Performance`. For the sake of space, we have just focused on the `Control-Flow` perspective in this paper. It defines an approach that consists in analyzing how each task/activity follows each other in an event log, and infer a possible model for the behavior captured in the observed process.

*3) Process Variant Comparison:* Our goal was also to compare the behaviour among the teams involved in the experiment against the "best practice" process, as performed by the expert, and identify the ones with less differences. For this purpose, we used the `Process Comparator` plugin [30], which is a tool that compares a collection of event logs, using a directed flow graph. It uses transition systems to model behavior and to highlight differences. Transition systems are annotated with measurements, and used to compare the behavior in the different variants. The annotations of each variant are compared using statistical significance tests, in order to detect relevant differences.

## IV. Results

Figure 2 presents team T-26 process variant, showing the code smells detection activities, and the correspondent statistics about the process followed to execute the requested task. It is clear, based on the different levels of blue in the activities performed, that they executed more often the activities related with the code smells detection and correction. We confirm this

[6]version 6.8, available at http://www.promtools.org
[7]http://jasml.sourceforge.net/
[8]doi:10.17632/8dmdwpgdy4.1

TABLE I
COLLECTED EVENTS STATISTICS

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

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

by observing the `Eclipse Editor | File Editing` activity which was executed more than any other activity.

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

Software artifacts with higher cyclomatic complexity tend to be harder to maintain. It has been claimed that the same rationale is applicable to process models [32]. Based on this, we were looking for the teams with less complexity in their processes. As shown, teams T-26, T-24 and T-41 are the ones with less `Cyclomatic Complexity` (as represented by different levels of green), therefore closer to the complexity of the REFERENCE model. That is also reflected by the number of `Simple` and `Composite States`, and `Activities` discovered in each of those models. Team T-26 modelled behavior was also the one discovered with best precision (45%) among these 3 teams.

On the opposite pole (as represented by different levels of red) with an unique characterization, we have team T-01, with four members, which did not delivered the results of the requested task. Its proficiency was insufficient and careful review of the process revealed this team produced more generic events than project related events, as shown in Table I. From Figure 3 we can also learn this team used more unique command actions and respective categories than any other team, and that did not increase the number of edited files, as one would have expected. This leads us to think its members did not understand or follow the process at all, since many of their actions in the IDE apparently were not aligned with the required task. The high values of complexity, activities, number of transitions and composite states metrics observed in Table II complements this assumption.

We can, therefore, state the following: T-01 was an "expensive" team and the one that presented more risks from a project management perspective. When compared with other teams, this team had a similar process duration (see table IV) in executing the task, but did not deliver the expected outcomes at all. This team was not only non effective, but also showed major inefficiencies in whatever they tried to produce.

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

We also compared the behaviour between the 3 teams with less complex models against the reference model. The level of Control-Flow differences based on activity frequencies, as calculated with the Process Comparator plugin, is plotted in table III. Team T-24 was the one with less differences when compared with the reference model, followed very closely by T-26. Based on the complexity measurements, control-flow differences and team size, we advocate that T-26 had accomplished the task with the best overall efficiency and effectiveness. In fact, that is also reflected in the proficiency mark given by the professor (that acted as the task expert), as shown in table IV. This raises a set of other research questions, such as: can process mining be used to assess the proficiency of developers in general, or just for specific kinds of tasks?

TABLE II
MODELS DISCOVERED - METRICS SUMMARIZATION

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

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

### A. Validity Threats

*1) Internal validity:* Since some teams worked in shared laboratories at the university campus, different team members may have used, in the same computer, the same user/key pair

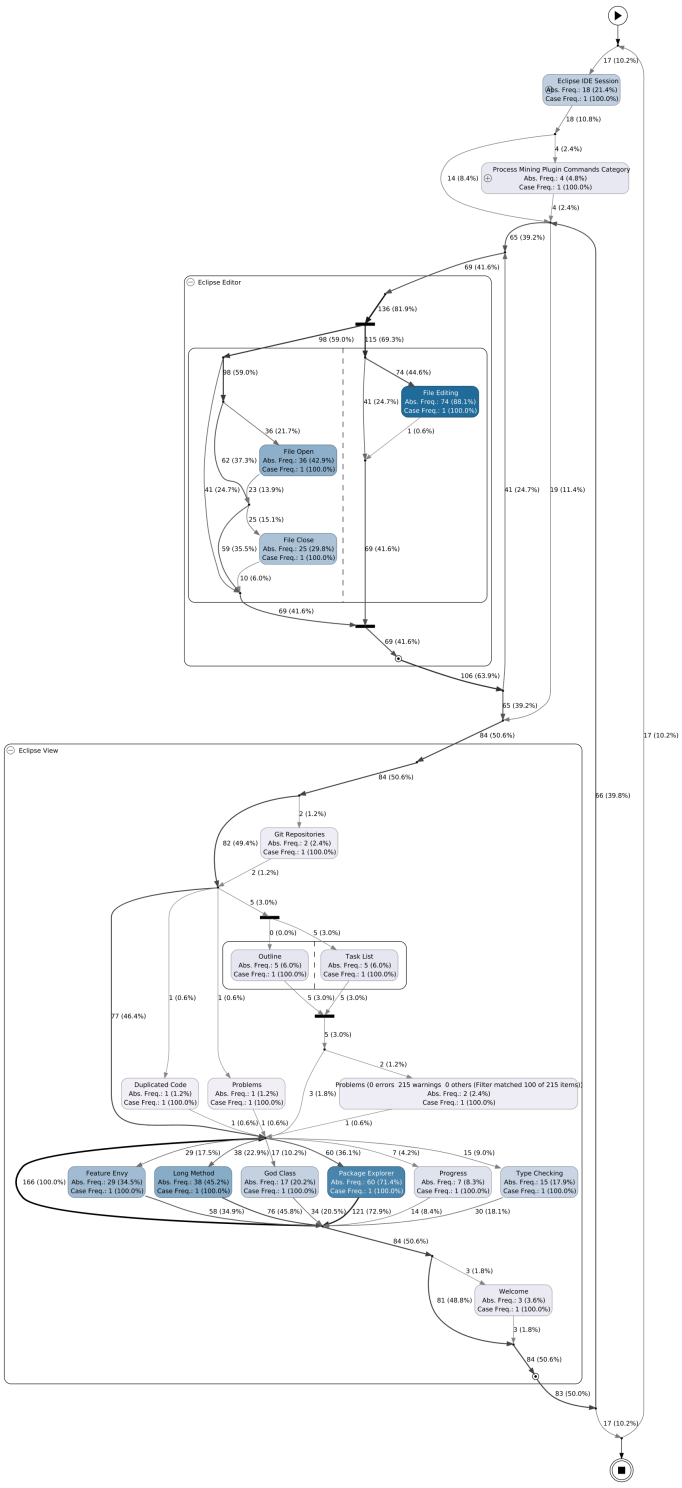


Fig. 2. Team T-26 Process Variant

to activate the collection plugin. This may be a source of non accuracy in collected data.

Some users have stopped the collection mechanism which makes it impossible to understand what they were doing during that period. We also found that a few teams have made a pause in the task, causing it to express more execution time than what

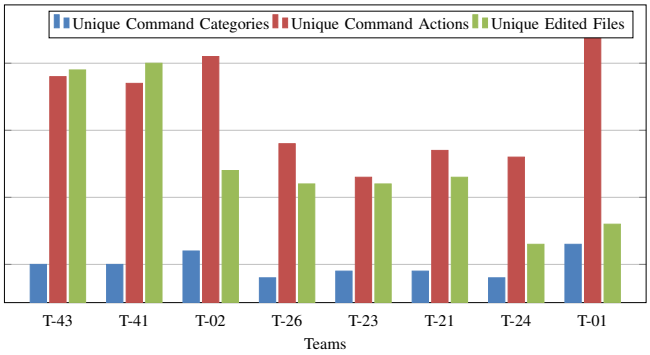


Fig. 3. Unique Categories,Commands and Edited Files Statistics

TABLE III
BEHAVIOR DIFFERENCES COMPARISON

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

TABLE IV
ASSIGNMENT DURATION

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

was really needed. The mined processes reflects these times, but indeed, that was idle time. Nevertheless, other reasons may exist for these delays, and therefore, their model is in fact accurate, because it plots what really happened.

*2) External validity:* Since we wanted to block some factors such as the degree of previous experience (background) in the proposed process, and repeat the data collection process in a between groups design, to avoid the learning effects of paired designs, the only feasible solution was to use students as subjects, as referred in subsection III-B1. We cannot claim that these students are adequate surrogates for professional software developers.

## V. CONCLUSIONS AND FUTURE WORK

### A. Conclusions

*1) RQ1:* We can not underestimate the fact that software development IDEs provide the users with a vast number of commands and menus to execute from, as seen in II-C. Trying to model these, is indeed a challenge, and, most times, a spaghetti-like process is the result of a successful process discovery. However, from an event log containing user actions,

we were able to model teams' behavior with moderate-to-strong Fitness and Precision values and yet achieve readable models. We are however well aware that these values should be validated with more experiments and different data.

The need to answer RQ1 was vital to understand if we could indeed discover processes followed by different people, that may be using different tools, in different locations but contributing to the same final outcome or product. The importance of understanding and measure teams' dynamics has grown with the current business trends that lead to Global Software Engineering (GSE) and Global Software Development (GSD). This is one of the main challenges faced by GSE and GSD, as in those kinds of projects the usual monitoring techniques are obsolete [33].

*2) RQ2:* We were able to discover and reconstruct process models representing the efficiency of software development teams, where, in some cases, members were working individually, each with their own IDE setup configurations. We confirmed that process mining may play a fundamental role in assessing the efficiency of software development teams and in potentially contributing to keep them focused on their tasks by checking and enforcing compliance to the prescribed processes.

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

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

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

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

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

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

### B. Future Work

The current work can be expanded in breadth and in depth. In this paper we mainly explored the control-flow perspective, but others are worth exploring, such as the organizational and performance perspectives. Devising team dynamics based upon the identification of the artifacts impacted/touched by the developers can be one of the following paths to research further. This study also opens the opportunity for new research related with forensic analysis on software development processes, exploring a combined perspective of the quality of the artifacts produced and the underlying processes.

While unveiling the details of past process instances is important to understand what went wrong or unplanned, we should be able to react as soon as possible, that is, while the process is being executed, to enable just-in-time corrective actions. The IDE-based process mining architecture presented in this paper is forming the base of our **SPOTS** (**S**oftware **P**rocess **O**n-the-run **T**racking **S**ystem). This tool will provide near real-time software development process insights, at the individual or team level, such as in the Personal Software Process [34], or Team Software Process approaches [35], but in an automated fashion. According to [15], this kind of operational support is the most advanced form of process mining action.

We also plan to investigate how development process smells [36] may be used to assess software process drift management. Machine learning techniques are plausible candidates to automatically classify mined models (as good or bad process smells).

## ACKNOWLEDGMENT

The authors would like to thank to all ISCTE-IUL students involved in this research, as well as Prof. Vitor Basto Fernandes, who was in charge of the Software Engineering course where the experiment took place.

## REFERENCES

[1] P. Mohagheghi and M. Jorgensen, "What Contributes to the Success of IT Projects? Success Factors, Challenges and Lessons Learned from an Empirical Study of Software Projects in the Norwegian Public Sector," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 5 2017, pp. 371–373. [Online]. Available: http://ieeexplore.ieee.org/document/7965362/

[2] T. Chow and D.-B. Cao, "A survey study of critical success factors in agile software projects," *Journal of Systems and Software*, vol. 81, no. 6, pp. 961–971, 6 2008.

[3] M. Tanner and U. Von Willingh, "Factors leading to the success and failure of agile projects implemented in traditionally waterfall environments," in *Management, Knowledge and Learning*. Cape Town: University of Cape Town, South Africa, 2014, pp. 693–700. [Online]. Available: http://www.toknowpress.net/ISBN/978-961-6914-09-3/papers/ML14-618.pdf

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

[5] M. Borges Ribeiro, V. Diniz Duarte, E. Gomes Salgado, and C. Vieira Castro, "Prioritization of Critical Success Factors In The Process of Software Development," *IEEE Latin America Transactions*, vol. 15, no. 1, pp. 137–144, 1 2017. [Online]. Available: http://ieeexplore.ieee.org/document/7827917/

[6] K. E. Emam and A. G. Koru, "A Replicated Survey of IT Software Project Failures," *IEEE Software*, vol. 25, no. 5, pp. 84–90, 9 2008. [Online]. Available: http://ieeexplore.ieee.org/document/4602680/

[7] P. A. McQuaid, "Software disasters—understanding the past, to improve the future," *Journal of Software: Evolution and Process*, vol. 24, no. 5, pp. 459–470, 2012.

[8] IEEE Computer Society, *SWEBOK V3.0*. IEEE Computer Society, 2014, no. V3.0. [Online]. Available: http://www4.ncsu.edu/ tjmenzie/cs510/pdf/SWEBOKv3.pdf www.swebok.org

[9] M. Niazi, S. Mahmood, M. Alshayeb, M. R. Riaz, K. Faisal, N. Cerpa, S. U. Khan, and I. Richardson, "Challenges of project management in global software development: A client-vendor analysis," *Information and Software Technology*, vol. 80, no. C, pp. 1–19, 12 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584916301227

[10] A. M. Lemos, C. C. Sabino, R. M. F. Lima, and a. L. Oliveira, "Conformance Checking of Software Development Processes Through Process Mining," *Seke*, 2011.

[11] J. Caldeira and F. Brito E Abreu, "Software development process mining: Discovery, conformance checking and enhancement," in *Proceedings - 2016 10th International Conference on the Quality of Information and Communications Technology, QUATIC 2016*. IEEE, 9 2017, pp. 254–259. [Online]. Available: http://ieeexplore.ieee.org/document/7814558/

[12] W. Poncin, A. Serebrenik, and M. V. D. Brand, "Process Mining Software Repositories," *2011 15th European Conference on Software Maintenance and Reengineering*, pp. 5–14, 2011.

[13] W. Van Der Aalst, A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. De Leoni, P. Delias, B. F. Van Dongen, M. Dumas, S. Dustdar, D. Fahland, D. R. Ferreira, W. Gaaloul, F. Van Geffen, S. Goel, C. Günther, A. Guzzo, P. Harmon, A. Ter Hofstede, J. Hoogland, J. E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F. Maggi, D. Malerba, R. S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H. R. Motahari-Nezhad, M. Zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H. Seguel Pérez, R. Seguel Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, E. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M. Wynn, "Process mining manifesto," *Lecture Notes in Business Information Processing*, vol. 99 LNBIP, pp. 169–194, 2012.

[14] M. L. Van Eck, X. Lu, S. J. J. Leemans, and W. M. P. Van Der Aalst, "PM 2 : a Process Mining Project Methodology," in *International Conference on Advanced Information Systems Engineering CAiSE 2015: Advanced Information Systems Engineering*. Springer, Cham, 2015, pp. 297–313.

[15] W. van der Aalst, *Process Mining : Data Science in Action*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016.

[16] A. Fuggetta, E. D. Nitto, and P. Milano, "Software Process," *Proceedings of the on Future of Software Engineering*, pp. 1–12, 2014.

[17] A. Meidan, J. A. García-García, I. Ramos, and M. J. Escalona, "Measuring Software Process," *ACM Computing Surveys*, vol. 51, no. 3, pp. 1–32, 6 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3212709.3186888

[18] A. R. C. Maita, L. C. Martins, C. R. López Paz, L. Rafferty, P. C. K. Hung, S. M. Peres, and M. Fantinato, "A systematic mapping study of process mining," *Enterprise Information Systems*, vol. 12, no. 5, pp. 505–549, 5 2018. [Online]. Available: https://www.tandfonline.com/doi/full/10.1080/17517575.2017.1402371

[19] V. A. Rubin, I. Lomazova, and W. M. P. v. d. Aalst, "Agile development with software process mining," *Proceedings of the 2014 International Conference on Software and System Process - ICSSP 2014*, pp. 70–74, 2014.

[20] V. A. Rubin, A. A. Mitsyuk, I. A. Lomazova, and W. M. P. 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 - ESEM '14*, pp. 1–8, 2014.

[21] C. . Ioannou, A. . Burattin, and B. Weber, "Mining Developers' Workflows from IDE Usage," *Lecture Notes in Business Information Processing*, vol. 316, pp. 167–179, 2018.

[22] L. Bao, Z. Xing, X. Xia, D. Lo, and A. E. Hassan, "Inference of development activities from interaction with uninstrumented applications," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1313–1351, 6 2018. [Online]. Available: http://link.springer.com/10.1007/s10664-017-9547-8

[23] K. Damevski, D. C. Shepherd, J. Schneider, and L. Pollock, "Mining Sequences of Developer Interactions in Visual Studio for Usage Smells," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 359–371, 4 2017.

[24] M. Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand, "The Statechart Workbench: Enabling scalable software event log analysis using process mining," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 3 2018, pp. 502–506. [Online]. Available: http://ieeexplore.ieee.org/document/8330248/

[25] K. Beck, M. Fowler, J. Brant, W. Opdyke, and D. Roberts, "Bad Smells in Code," in *... Improving the design of existing code*, 1999.

[26] C. Günther and E. Verbeek, "XES standard definition," *Fluxicon Process Laboratories*, 2014.

[27] I. Vanderfeesten, J. Cardoso, J. Mendling, H. A. Reijers, and W. Van Der Aalst, "Quality Metrics for Business Process Models," Technische Universiteit Eindhoven, Tech. Rep., 2007. [Online]. Available: http://wwwis.win.tue.nl/ wvdaalst/publications/p364.pdf

[28] A. Rozinat, A. de Medeiros, C. Günther, A. Weijters, and W. van der Aalst, "Towards an evaluation framework for process mining algorithms," *Beta, Research School for Operations Management and Logistics.*, pp. 1–20, 2007. [Online]. Available: http://www.processmining.org.

[29] A. Rozinat, M. Veloso, and W. M. P. van der Aalst, "Evaluating the Quality of Discovered Process Models," *Information Systems Journal*, vol. 16, no. Section 2, pp. 1–8, 2008.

[30] A. Bolt, M. de Leoni, and W. M. van der Aalst, "Process variant comparison: Using event logs to detect differences in behavior and business rules," *Information Systems*, vol. 74, pp. 53–66, 5 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437916305257

[31] M. Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand, "Recursion aware modeling and discovery for hierarchical software event log analysis," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 185–196.

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

[33] F. Jurado and P. Rodriguez, "Sentiment Analysis in monitoring software development processes: An exploratory case study on GitHub's project issues," *Journal of Systems and Software*, vol. 104, pp. 82–89, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215000485

[34] W. S. Humphrey, "Personal Software Process (PSP)," in *Encyclopedia of Software Engineering*, 2nd ed., J. J. Marciniak, Ed. New York, NY, USA: John Wiley & Sons, 2002, p. 1584.

[35] ——, "Team Software Process (TSP)," *Encyclopedia of Software Engineering*, 2002.

[36] B. Weber, M. Reichert, J. Mendling, and H. A. Reijers, "Refactoring large process model repositories," *Computers in Industry*, vol. 62, no. 5, pp. 467–486, 2011.