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
{jcpcp, fba, jvprs}@iscte-iul.pt

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

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 IEventBroker3 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 plugin4 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.

Listing 1: Sample Eclipse Event Instance

```
{  
  "team" : "T-01",  
  "session" : "a5d633-9d13-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",  
  ...
}
```

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](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. 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 and imported into ProM process mining tool.

   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.

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 α-algorithm, the heuristics, genetic and fuzzy miner. However, our need to discover and visualize the processes in multiple ways lead us to choose the ProM’s StateChart Workbench plugin [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
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 I

<table>
<thead>
<tr>
<th>Team</th>
<th>TM</th>
<th>UCC</th>
<th>UCA</th>
<th>UEA</th>
<th>PE (#/%)</th>
<th>GE (#/%)</th>
<th>TE (#)</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-01</td>
<td>1</td>
<td>4</td>
<td>14</td>
<td>7</td>
<td>928</td>
<td>126</td>
<td>355</td>
</tr>
<tr>
<td>T-02</td>
<td>2</td>
<td>3</td>
<td>22</td>
<td>16</td>
<td>371</td>
<td>192</td>
<td>738</td>
</tr>
<tr>
<td>T-04</td>
<td>3</td>
<td>2</td>
<td>37</td>
<td>32</td>
<td>791</td>
<td>283</td>
<td>738</td>
</tr>
<tr>
<td>T-21</td>
<td>1</td>
<td>9</td>
<td>27</td>
<td>2</td>
<td>296</td>
<td>296</td>
<td>296</td>
</tr>
</tbody>
</table>

**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**

### Table II

<table>
<thead>
<tr>
<th>Team</th>
<th>F (%)</th>
<th>P (%)</th>
<th>A</th>
<th>HD</th>
<th>SS</th>
<th>CS</th>
<th>T</th>
<th>CC</th>
</tr>
</thead>
<tbody>
<tr>
<td>T-01</td>
<td>85.8%</td>
<td>29.9%</td>
<td>27</td>
<td>2</td>
<td>93</td>
<td>12</td>
<td>130</td>
<td>35</td>
</tr>
<tr>
<td>T-24</td>
<td>94.4%</td>
<td>45.0%</td>
<td>25</td>
<td>2</td>
<td>60</td>
<td>6</td>
<td>85</td>
<td>27</td>
</tr>
<tr>
<td>T-26</td>
<td>94.2%</td>
<td>46.5%</td>
<td>36</td>
<td>2</td>
<td>93</td>
<td>12</td>
<td>131</td>
<td>36</td>
</tr>
</tbody>
</table>

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

Some users have stopped the collection mechanism which makes it impossible to understand what they were doing during that period. We also found that a few teams have made a pause in the task, causing it to express more execution time than what was really needed. The mined processes reflect 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 cannot 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.

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 (Software Process On-the-run Tracking System). 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

