Conformance Checking of Software Development Processes Through Process Mining

Artini M. Lemos, Caio C. Sabino, Ricardo M. F. Lima, César A. L. Oliveira
Center for Informatics, Federal University of Pernambuco
Recife, Pernambuco, Brazil
{aml2, ccss2, rmfl, calo}@cin.ufpe.br

Abstract

The design and management of software development processes is essential to reduce costs and improve the quality of software products. The execution of such processes is usually monitored to register important information about the dynamic behavior of the software development process. As a result, a huge amount of information is stored in the database and the software managers are deluged by data. Then, it is important to find ways to automatically analyze the process execution logs to discover the actual process model, characterize trends in it, and flag anomalies. Process mining is a well established technique designed to attack this challenge. This paper demonstrates the application of process mining technique and tools to perform a software development process conformance analysis to detect inconsistencies between a process model and its corresponding execution log. The work conducts a case study using an event log database with more than 2,000 process instances gently provided by a Brazilian software house. The paper aims at closing the gap between process mining and software engineering areas. We hope to motivate and guide software managers to adopt process mining technique as a practical tool for the analysis and improvement of software development processes.

1. Introduction

Software development usually follows a sequence of activities designed to capture the best practices from software development projects. Such sequence of activities defines the software development process, which should be constantly revised and improved to embody strategies for accomplishing software evolution. Even when the process is automated, due to personal preferences, previous experiences, or even time and cost pressures, it is common to see project managers adopting their own software development processes. Moreover, usually some process activities are ignored to save time and meet the project deadline. Therefore, the first step of any software process improvement project should be the conformance analysis to determine if the defined process is being actually followed.

In this context, it is important to discover the real software development process to compare it with the company’s formal process. Software development process discovery has been an intensive task, either done through exhaustive empirical studies or in an automated fashion.

Software houses usually automate their development processes through a monitoring system. This system is capable of registering valuable information about process execution, such as timestamps, task identification, stakeholders involved in the execution of an activity etc., in the form of execution logs. Such execution logs (also called event logs or audit trails) are the starting point for the process mining technique, which was developed to semi-automatically extract useful information from them [4].

Processing mining has been successfully applied in many different areas, including software engineering. In [1] authors explore process mining techniques for discovering development processes from publicly available open source software development repositories. In [6] authors define a process mining framework to discover software development processes from event logs generated by software configuration management (SCM) systems. Unfortunately, these works are more oriented to the computational intelligence area. They focus on describing the mining techniques, demonstrating their efficiency in discovering software process models. Less attention is devoted to the application and use of such techniques to evaluate the software development process in a real environment.

This work aims at bridging this gap between process mining and software engineering areas by demonstrating how to explore process mining techniques to perform a conformance analysis and evaluate whether the established software development process is being followed. The conformance analysis main goal is detecting inconsistencies.
between a process model and its corresponding execution log. Despite some works have explored process mining techniques for the conformance analysis of general business processes [5], it is not of our knowledge any study that applies process mining to conformance checking of software development processes.

In this work we use the process mining ProM tool [9] to mine the event logs of a real company and discover the process model that is actually employed by them. We are particularly interested in the process control-flow perspective. In the control-flow conformance analysis, we want to evaluate if process activities are being executed in the correct order and discover which activities are actually being executed in the process instances. For this objective, we employed a feature of the ProM framework that generates a Markov chain from the process logs.

The paper explores a real database with event logs generated in the past five years from the execution of a software development process. The database was gently provided by a Brazilian software house with annual revenue of more than US$ 500 million. The database includes more than 2,000 cases (process instances).

The paper is organized as follows. Section II describes the process mining technique and presents the ProM tool. Section III discusses the conformance checking study conducted in this work. It begins by presenting the formal software development process defined by the company; then describes the event log characteristics and the mining strategy; in the next step it demonstrates how to use ProM to identify problems in the even logs and how these problems were corrected; finally, it discusses the conformance analysis study herein conducted. Conclusions and future works are discussed in Section IV.

2. Process Mining

Business process mining, or just process mining, is a “nascent research field at the intersection of data mining and business process modeling” [3] that targets the extraction of non-trivial and useful information based on execution logs [4]. Since many systems record their transactions in log files, process mining techniques can be used for a large class of business processes. By using process mining, many kinds of information can be collected about the process, such as control-flow, performance, organizational information and decision patterns.

It is important to note that most process mining techniques do not generate a model for only reproducing all log entries, but collects general information about the process real execution, considering that exceptional behavior or some minor errors may occur in the log file that should not be part of the process.

Once the process model is obtained and possibly validated, it can be analyzed in order to better understand how it actually works. In the analysis of the process model, some changes can be proposed and their effects can be estimated and measured. That means that the impact of some changes can be calculated without having to change and verify the real process. Therefore, process mining techniques can be used to gain insight on how to improve the process quality.

Many process mining algorithms have been proposed. The ProM Framework [9] is a tool that features many mining and analysis plug-ins implementing several of these algorithms.

ProM is an extensible open source framework implemented in Java with a user-friendly graphical interface that provides a wide variety of plug-ins that support, implement, and use process mining techniques. There are currently more than 230 plug-ins, distributed on functionalities like filtering, mining, analysis, exporting and conversion.

The input log files for ProM are in a common log format: Mining XML (MXML). ProM can also take as input process model files, such as PNML and many others. When the log file is opened, ProM collects some statistical data about it and present them in the main window. Also, many plug-ins become available to operate on the log.

The mining plug-ins aim at discovering multiple perspectives of the process, such as the control-flow, the organizational structure, the data perspectives and some other information. Details about these perspectives are given in [9]. The analysis plug-ins deal with verification of process models, conformance checking between the log and the model, performance analysis, among other things.

ProM also provides filters for cleaning the log from undesired elements. Filtering “is usually a projection of the log to consider only the data you are interested in” [9]. Also, some mining plug-ins have configurable settings in order to obtain better results depending on the user’s purposes.

ProM allows multiple combinations of plug-ins, making it possible to analyze the results of a mining algorithm or merge the contents of multiple perspectives to obtain a more complete model for the process.

With an extensible framework, new algorithms can be easily added as new plug-ins and this makes ProM a very powerful tool. ProM works with several modeling languages, such as Petri Nets [2], EPCs [7], YAWL [8], and supports the conversion between most of them, being a very versatile tool.

3. Discovering a Software Development Process

In this section, we explore the case of a Brazilian software house with annual revenue of more than US$ 500 million. Five years ago, the company defined a software devel-
Development process and deployed a monitoring system to register relevant information about its execution. Since then, a huge volume of event logs has been generated, in a way that became impossible to the software quality team to manually extract useful and trustful information from the database.

In this context, we applied process mining techniques to extract important information on the real software development process from the event logs. The main objective is to discover the actual process being executed by the development team and compare it against the documented process. The database under consideration includes more than 2,000 instances of the process.

### 3.1. The Formal Software Development Process

In its documented specification, the software development process begins with the activity Proposal and Production Budget (PPB) as it can be seen in Figure 1. The PPB includes two tasks: 1) creating a functional specification document, which aims at detailing the recommended solution; 2) and filling in a worksheet to estimate the software development effort. A manager analyzes the functional specification document in the activity Functional Specification Document Analysis (FSDA) before submitting the proposal to the client. In the next step the production planning and budget proposal is sent to the client - activity Send Production Planning and Budget (SPPB). Then, the client analyses the production planing and budget proposal and returns an approval/reject response - activity Client Proposal Analysis (CPA).

When the customer approves the proposal and production budget (CPA activity), the team leader analyses the technical specification to evaluate its complexity. Only complex specifications are validated - activity Validate Technical Specification (VTE). The Development (DEV) activity starts after validation. During this activity, programmers implement the software, create the processing scripts and database objects script (triggers, functions, packages, and view). When the development phase is completed, the Code Verification (CODV) activity is performed to check whether code complies with the specification and follows the development patterns. After the code verification phase, the program is tested in the Testing (TEST) activity. Eventually, with the software validated and tested, the process reaches the Documentation activity (DOC) and the process terminates.

This is an iterative process, allowing specification refinements/corrections and negotiation of the production planning and budget proposal. As one can see in Figure 1, activities FSDA, CPA, VTE, CODV, and TEST represent points where the process may move back to previous activities.

### 3.2. Mining the Software Process Logs

The company’s quality team is interested in assessing the model (or models) of the real process and compare it against the process that was supposed to be executed. Such conformance analysis will be the starting point for a process improvement project in the company.

The Mining XML (MXML) is an extensible, XML-based format for storing process event logs which can be analysed by the ProM tool [9]. Since the company’s database adopts a private file format, a compiler was developed to translate the event logs into the MXML file format.

ProM allows one to discovery different perspectives of the process being analyzed, namely control-flow, performance, and organizational perspectives. In the conformance analysis conducted in this paper, we are interested in the control-flow perspective. To discover which tasks are actually performed and their order of execution is of particular importance for the software quality team. The idea is to compare the control-flow model of the formal process against the control-flow model (or models) discovered from the real event logs.

In the ProM tool, information about each event is recorded in the form of an audit trail entry. Our event logs store the following information on each audit trail: 1) **taskID**: name of the task involved in a process event; 2) **event type**: a mark to indicate whether the event is a start or complete point in the process instance; 3) **timestamp**: moment at which the event took place; 4) **originator**: worker who perform the task (originate the event).

Since the audit trail entry identifies the tasks that are executed in a process (**taskID**) and allows for inferring their order of execution (**event type** and **time stamp**), then the control-flow perspective can be discovered.

Existing techniques for process mining perform very well for well-structured and organized event logs. However, the situation is not that simple upon dealing with real event logs. In this case the logs are usually very obscure and have noises, making it difficult to extract useful information. Therefore, in these situations a preprocessing phase to remove noises and organize the event logs is an obligation.

Our event log database was generated in the past five years, in a very active company, with hundreds of process instances executed each year. Thus, the preprocessing phase is essential to produce a well structured event log database. We used the reports produced by ProM tool to carefully analyze the database and a number of problems were found and fixed, as follows.

**Incomplete timestamps** - a) The monitoring system only registers the moment a given activity initiates, but it does not monitor the activity duration. However, this would be a concern only if we were interested in analyzing the per-
formance perspective of the process, which is not the case.
b) The original log recorded timestamps using information about days, months, and years, but missing hour, minute, and second details. Thus, activities occurring within a day were assumed to be parallel activities, rather than sequential ones. To solve this problem we redefined timestamps such that all activities in a process instance received different timestamps. We used the formal company’s process to determine the order of activities in time. One might argue that the solution is biased by the control-flow of the formal process. However, observing the original database, in more than 90% of the cases, if two activities in the same process instance have different timestamps, they follow the control-flow order defined for the formal process, which is a good result for the conformance analysis being conducted. Therefore, our assumption for the timestamps modification is acceptable. Nevertheless, despite the good conformance results for the control-flow order, in most cases, the set of activities being executed in a process instance are only a subset of the activities defined in the formal process. So, a deeper conformance analysis is still required.

Database corruption - Data stored in the audit trail entry of a small number of process instances appeared to be corrupted. For instance, we found timestamps with invalid date values. Moreover, some process instances had no start or complete events. Fortunately, the number of process instances with this kind of problem represented less than 2% of the whole database. Thus, they were discarded without significative loss for the conformance analysis study.

After the database preprocessing, we proceeded with the conformance analysis. We want to answer two main questions: 1) does the real process respect the ordering of execution defined in the formal process?; 2) which subset of activities in the formal process are actually executed in the real process? As we already observed, the answer for the first question is yes for more than 90% of the cases. But we need to investigate more to answer the second question.

3.3. Conformance Analysis

In this section, we describe the approach employed to analyze the software process based on the information provided by process mining. ProM features a plugin capable of generating a Markov chain from the discovered model, where each node is a process activity and transitions are labeled with probabilities as in ordinary Markov chains. Through the Markov chain it is possible to discover if sequences of activities defined for the process actually occur, and with which frequency. It is also possible to measure the probability of skipping a given activity that is defined in the flow and the probability of each decision that must be taken during the process execution.

We used Prom’s Markov chain model discovery feature in our study. Table 1 presents the Markov chain generated using ProM’s sequence clustering analysis plug-in.

With the Markov chain in hands, we can start the conformance analysis of the software development process. The purpose is to detect points in the actual process that are not in conformance with the company’s expectations and provide useful information for the company to initiate the process improvement project.

Figure 2 presents the formal process enriched with transition probabilities assessed from the Markov chain (Table 1). For clarity and readability, Fig. 2 includes only transitions relevant for the control-flow conformance analysis. Besides the transitions already present in the formal process, we included some transitions (dashed arrows) to indicate undesirable situations observed in the discovered process model.

Now, let us analyze the results presented in Fig. 2 to point out and discuss some conformance problems.

The probability of transitioning from the initial state (in) to the development state (DEV) is of 25.2%. This means that about one in each four projects started by the development activity, skipping the whole planning stage. The activity send production planing and budget (SPPB) is almost never reached by any other activity in the process model and could be removed from the formal process. Although validate technical specification (VTE) is performed for complex software projects, the execution of such activity in the actual process is fairly rare, with probability of 0.2%. This points out that this activity is executed in very exceptional cases and, therefore, could be removed from the company’s standard development process.

A more critical problem can be seen in the end of the process. Software verification, testing, and documentation are important activities of the best practices for software development projects. Nevertheless, such activities are ignored by 43.6% of the software development processes analyzed (see transition from the development phase (DEV) to the
Table 1. Markov chain model for the process

<table>
<thead>
<tr>
<th></th>
<th>in</th>
<th>PPB</th>
<th>FSDA</th>
<th>SPPB</th>
<th>CPA</th>
<th>DEV</th>
<th>VTE</th>
<th>CODV</th>
<th>TEST</th>
<th>DOC</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>in</td>
<td>-</td>
<td>0.56</td>
<td>0.018</td>
<td>0.003</td>
<td>0.128</td>
<td>0.252</td>
<td>-</td>
<td>0.018</td>
<td>0.007</td>
<td>0.014</td>
<td>-</td>
</tr>
<tr>
<td>PPB</td>
<td>-</td>
<td>-</td>
<td>0.285</td>
<td>0.021</td>
<td>0.276</td>
<td>0.136</td>
<td>-</td>
<td>0.005</td>
<td>0.008</td>
<td>0.003</td>
<td>0.266</td>
</tr>
<tr>
<td>FSDA</td>
<td>-</td>
<td>0.068</td>
<td>-</td>
<td>-</td>
<td>0.305</td>
<td>0.19</td>
<td>-</td>
<td>0.032</td>
<td>0.004</td>
<td>0.016</td>
<td>0.385</td>
</tr>
<tr>
<td>SPPB</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.658</td>
<td>-</td>
<td>0.026</td>
<td>-</td>
<td>-</td>
<td>0.316</td>
</tr>
<tr>
<td>CPA</td>
<td>-</td>
<td>0.01</td>
<td>0.006</td>
<td>0.002</td>
<td>-</td>
<td>0.767</td>
<td>0.002</td>
<td>0.065</td>
<td>0.015</td>
<td>0.002</td>
<td>0.131</td>
</tr>
<tr>
<td>DEV</td>
<td>-</td>
<td>0.016</td>
<td>0.006</td>
<td>-</td>
<td>0.013</td>
<td>-</td>
<td>0.002</td>
<td>0.462</td>
<td>0.061</td>
<td>0.004</td>
<td>0.436</td>
</tr>
<tr>
<td>VTE</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0.75</td>
<td>-</td>
<td>0.25</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CODV</td>
<td>-</td>
<td>0.009</td>
<td>-</td>
<td>-</td>
<td>0.003</td>
<td>0.035</td>
<td>-</td>
<td>-</td>
<td>0.234</td>
<td>0.013</td>
<td>0.706</td>
</tr>
<tr>
<td>TEST</td>
<td>-</td>
<td>0.006</td>
<td>0.006</td>
<td>-</td>
<td>0.029</td>
<td>0.02</td>
<td>-</td>
<td>0.018</td>
<td>-</td>
<td>0.553</td>
<td>0.368</td>
</tr>
<tr>
<td>DOC</td>
<td>-</td>
<td>0.032</td>
<td>-</td>
<td>0.001</td>
<td>0.028</td>
<td>0.016</td>
<td>-</td>
<td>0.036</td>
<td>0.015</td>
<td>-</td>
<td>0.872</td>
</tr>
</tbody>
</table>

Figure 2. The actual development process with the undesired transitions shown as dashed arrows

Table 2. Activity execution frequency

<table>
<thead>
<tr>
<th>Activity</th>
<th>Absolute occurrence</th>
<th>Relative occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEV</td>
<td>1556</td>
<td>27.385%</td>
</tr>
<tr>
<td>PPB</td>
<td>1356</td>
<td>23.865%</td>
</tr>
<tr>
<td>CODV</td>
<td>853</td>
<td>15.012%</td>
</tr>
<tr>
<td>CPA</td>
<td>841</td>
<td>14.801%</td>
</tr>
<tr>
<td>FSDA</td>
<td>442</td>
<td>7.779%</td>
</tr>
<tr>
<td>TEST</td>
<td>342</td>
<td>6.019%</td>
</tr>
<tr>
<td>DOC</td>
<td>250</td>
<td>4.4%</td>
</tr>
<tr>
<td>SPPB</td>
<td>38</td>
<td>0.669%</td>
</tr>
<tr>
<td>VTE</td>
<td>4</td>
<td>0.07%</td>
</tr>
</tbody>
</table>

terminal state (out)). This shows that the software project terminates in the development stage with a very high probability, without even being verified. Even worst, looking at the transition from the code verification activity (CODV) to testing (TEST), one can notice that, if the software happen to be verified, it is usually neither tested (TEST) nor documented (DOC).

Eventually, Figure 2 demonstrates that the actual process is not iterative. Thus, it is very unlikely to observe a return to previous activity during the process execution. Apparently, when a problem occurs during the process execution, it goes straight to the terminate state.

In ProM, a cluster of process instances with same sequence of activities is called a process type. Using ProM’s sequence clustering plugin we can discover the process types present in the log and the frequency of occurrence of each one. The goal is to evaluate the conformance of these process types against the formal process.

ProM discovered 190 different process types in our event log. Figure 3 presents the two most frequent. The most frequent process type only executes the development (DES) activity, which is a clear indication of nonconformance against the formal process. On the other hand, the second most frequent process shows that many projects are cancelled by the company after the proposal and production budget (PPB), which is a fairly acceptable behavior.

We then analyzed the list of the ten most frequent process types to select those with at least three activities. Out of the ten most frequent process types, only the 4th, 5th, and 10th had three or more activities. Figure 4 depicts these process types. The testing (TEST) and documentation (DOC) activities are not executed in the fourth and fifth most frequent process types. A good conformance result is observed in the tenth most frequent process type. Only validation (VTE) and send production planing and budget (SPPB) activities are missing. Nevertheless, VTE activity is optional in the formal process. Additionally, through the analysis of the Markov chain, we had already identified that the SPPB activity is almost never executed and recommended its elimination from the formal software development process.
4 Conclusions

This paper demonstrated the application of process mining techniques to examine if the actual software development process employed by a company conforms to its formal specification. The work is conducted within the context of a Brazilian software house. A total of 2,000 process instances, recorded in five years of operation, were evaluated.

The conformance checking demonstrated that, although some projects follow the formal process, most of them do not occur as expected. Some serious problems were identified. The initial phase of the formal process, responsible for budget/functionalities proposal/approval, is usually skipped. This may have important impact on the budget planning and increase the costs with software maintenance. Another conformance problem was observed in the final phase of the process. The software implementation is validated against the specification only in 46.2% of the cases. Even worst, if the software happens to be validated, in 70.6% of the cases it is neither tested nor documented.

It is not the purpose of this work to evaluate the impact of the conformance problems in the company’s productivity. Indeed, the conformance analysis produced a set of reports indicating inconsistencies detected between a process model and its corresponding execution log. For instance, one of the reports enumerate the most common process types\(^1\) in the event logs. In this report, we observed that the tenth most common process type conforms to the formal software development process. However, only the software quality team is able to attest if this is a good scenario and take actions to solve the detected issues.

This work aimed at bridging the gap between process mining and software engineering areas. From a software engineering perspective, it describes how to apply the process mining tool in a practical fashion. We believe that software project managers can benefit from this paper by gaining insights on how to incorporate process mining tools in their professional routines.

Although in the conformance analysis we have explored the control-flow perspective, the procedures may be easily adapted to evaluate other software development process perspectives, such as performance, cost, resource allocation, and many others. As a future work, we intend to demonstrate how process mining can be applied to analyze other software development process perspectives and develop a guide to help software engineering people in the use of process mining techniques and tools.

References


