Pattern mining in system logs: opportunities for process improvement

Dolev Mezebovsky, Pnina Soffer, and Ilan Shimshoni

University of Haifa, Carmel Mountain 31905, Haifa, Israel
dkmezebov@gmail.com, spnina@is.haifa.ac.il, ishimshoni@mis.haifa.ac.il

Abstract. Enterprise systems implementations are often accompanied by changes in the business processes of the organizations in which they take place. However, not all the changes are desirable. In “vanilla” implementations it is possible that the newly operational business process requires many additional steps as “workarounds” of the system limitations, and is hence performed in an inefficient manner. Such inefficiencies are reflected in the event log of the system as recurring patterns of log entries. Once identified, they can be resolved over time by modifications to the enterprise system. Addressing this situation, the paper proposes an approach for identifying inefficient workarounds by mining the related patterns in an event log. The paper characterizes such patterns, proposes a mining algorithm, and rules for prioritizing the required process improvements.

Keywords: Process mining, Enterprise systems, Event log

1 Introduction

Enterprise systems implementations are often accompanied by changes in the business processes of the organizations in which they take place. In fact, the desired change in the business processes is in many cases one of the reasons that motivate the enterprise system implementation. Changes in the business processes can also stem from the need to adapt the enterprise to the enterprise system rather than the other way around [10]. In such cases, some process changes can be considered improvements relatively to the original processes prior to the implementation, but not necessarily all of them.
This is especially true in implementations that take a “vanilla” strategy [15], in which the system is implemented as it is with minimal customizations and adaptations. In such situations, a typical scenario would be that the newly operating business process is still capable of achieving its operational goal, but requires many additional steps as workarounds of the system’s limitations. Thus, the achievement of operational goals is at the cost of more effort, resources, and time.
To illustrate the situation, we will consider the following case taken from a university and use it as a running example throughout this paper. In the university, a student registers for a program, and may decide to switch to another program while he studies. Prior to the implementation of an enterprise system, changing the program to which a student was registered was done through a legacy system. When the secretary

was reporting a change in a student's program, all the courses the student had already taken were "converted" to the new program. Then the secretary could specifically remove the credits of the courses which were not relevant for the new program. Such activity is not supported by the enterprise system implemented in the university. Hence, when a student wishes to change the program he is registered to, the secretary has to separately detach all the course credits the student already has, and attach them again under the new program. This task is both time consuming and error-prone.

Typically, such situations arise shortly after the system becomes live, and are intended to be addressed later on, as incremental improvements of the already running system. For example, such an improvement could be achieved by adding a function to the university enterprise system. This function would automatically detach all the credits of a student and attach them again under a new program, while all the secretary has to do is to indicate the program change. However, since this may be the case with a large number of processes, they cannot all be immediately addressed. Furthermore, as time passes by, the people who operate the process may get used to the inefficient way of performing their task, and thus they will not require its improvement. As a result, the process will remain in its inefficient form. The problem which is then faced by the organization is first to identify the inefficient processes, and second, to prioritize them so they can gradually be improved. To the best of our knowledge, this problem has not been addressed so far.

This paper proposes an approach for identifying and prioritizing requirements for process improvement. Specifically, we address inefficient processes whose inefficiency stems from workarounds forced by a newly introduced enterprise system. The identification is based on mining event logs of the system, and prioritization is based on the frequency of these workarounds and on their magnitude.

The situation addressed here is when technology (new enterprise system) drives changes in the business processes, albeit in an undesirable way. The approach uses a technological solution (mining event logs) to drive desirable changes in the processes.

The remainder of the paper is organized as follows. Section two demonstrates and characterizes the reflection of workarounds in the event log of a system; Section three provides a basic formalization of a pattern in a log file and an algorithm for pattern mining; Section four addresses the prioritization and utilization of the patterns for process improvement; Section five discusses the proposed approach as compared to related work; conclusions are given in Section six.

2 The reflection of workarounds in an event log

Our premise is that a series of steps that logically reflect an activity from the business process point of view is reflected in the event log of an enterprise system as a recurring pattern performed by the same user. In this section we illustrate this by an example related to the above mentioned university process.

Although a log file includes actions performed by all the system users, we show in our example (Table 1) only the log entries that relate to one user (YPRESS). Table 1 includes log entries, specifying the process code and name, where "process" is actually a transaction, the timestamp (date and time), the user name, and the parameters to which the transaction applies (in this case course name, program name, and student name). All entries include two types of processes (transactions): attach course and detach course. They all apply to the same student (Fredrick), three programs (MIS Major, CS Minor, and MIS Minor), and different course names. Finally, all the entries relate to the same date and were performed within about 15 minutes.

Table 1. Event Log Example

Row Num	Process	Process Name	Date	Time	User Name	Student Name	Course Name	Program Name
1	PR12	Attach Course	15.06.08	13:45:52	YPRESS	Fredrick	Linear Algebra	MIS Major
2	PR12	Attach Course	15.06.08	13:46:26	YPRESS	Fredrick	Algorithms	MIS Major
3	PR12	Attach Course	15.06.08	13:47:44	YPRESS	Fredrick	Data Structures	MIS Major
4	PR11	Detach Course	15.06.08	13:49:18	YPRESS	Fredrick	Linear Algebra	CS Minor
5	PR11	Detach Course	15.06.08	13:49:24	YPRESS	Fredrick	Algorithms	CS Minor
6	PR11	Detach Course	15.06.08	13:49:31	YPRESS	Fredrick	Data Structures	CS Minor
7	PR12	Attach Course	15.06.08	13:54:19	YPRESS	Fredrick	Information Technology	MIS Major
8	PR11	Detach Course	15.06.08	13:55:28	YPRESS	Fredrick	Information Technology	MIS Minor
9	PR12	Attach Course	15.06.08	13:56:40	YPRESS	Fredrick	Business Intelligence	MIS Major
10	PR11	Detach Course	15.06.08	13:58:20	YPRESS	Fredrick	Business Intelligence	MIS Minor
11	PR12	Attach Course	15.06.08	13:59:35	YPRESS	Fredrick	Programming Design	MIS Major
12	PR11	Detach Course	15.06.08	14:01:29	YPRESS	Fredrick	Programming Design	MIS Minor

The short time frame, within which a series of operations concerning a recurrent set of parameters was performed, may indicate a pattern that stands for one “logical” activity. Our goal is to be able to automatically identify such patterns in an event file, and successfully indicate a larger activity that has been done by the user. Note that the patterns we address do not bear a meaning which is similar in any sense to the workflow patterns [4]. They are not generic. Rather, they capture a recurrent set of related log entries. To get a better understanding about patterns and their structure, we represent the log entries of Table 1 graphically in Fig. 1 and Fig. 2.

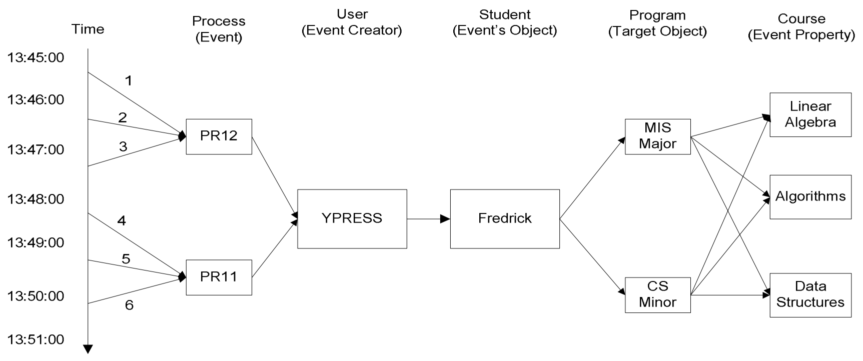

Fig. 1. A graphical representation of rows 1 to 6 of Table 1**.**

Fig. 1 shows two distinct sets of entries along time. The first three entries perform the operation PR12 (attach course) to MIS Major program with three different courses, and the last three entries perform the operation PR11 (detach course) to CS Minor program with the same three courses. All the operations are performed by the same user to the same student. In Fig. 2 no such distinct sets of operations exist over the time axis. Rather, the operations PR12 and PR11 alternate. Still, they are performed to two programs of the same student and by the same user.

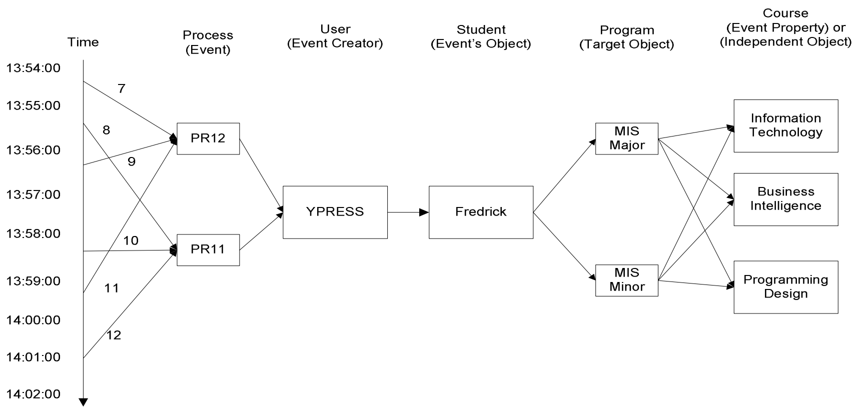

Fig. 2. A graphical representation of rows 7 to 12 of Table 1

We classify all these entries as belonging to the same pattern, and draw the following general indications for the existence of a pattern. (a) All the entries are performed by the same user and within a limited time frame. The maximal time frame for pattern identification can be given as a parameter to an automated application which will identify patterns in a system log. (b) The entries have at least one parameter whose value is fixed. We term the fixed parameter(s) the *invariant* set of the pattern. (c) The entries have at least one parameter whose value is different for different entries. We term the parameter(s) whose value changes throughout the entries the *variant* set of the pattern. As to the order of performing the operations in the pattern, we do not consider it mandatory for a fixed order (e.g., all PR12 and then all PR11, or alternating operations). Since we assume that for the person who performs these operations they all belong to one logical activity, the specific execution order is not necessarily of importance.

In the following section we formalize the pattern definition and propose an algorithm for pattern detection.

3 Pattern Mining

3.1 Basic concepts

To formalize the pattern concept, we need to start by providing formal definitions of an entry in a log file and its components.

User – A field in the log entry that indicates who made the commit of the event.
$User \in \{System\ Users\}$
Timestamp – The time the log entry was committed.
$Timestamp > 0$
Operation – The type of activity (transaction) that was performed.
$Operation \in \{System\ Operations\ \}$
Operand – Parameter of a mathematical function. In a log file entry an operand is a pointer to an object or a pointer to a parameter value of the function (transaction).
Operand $\in \{Operands\ :\ Operands\ \ \notin\ \emptyset\}$
ORSO – An ordered set of operands, with at least one operand in the set.
ORSO = (< Operands > **:** $|Operands|\ \geq 1$)
Entry – An event in the log file, which is represented by a tuple. The entry includes user, timestamp, operation and an ordered list of operands.
Entry = $< User, Timestamp, Operation,$ ORSO $>$.
TimeFrame – Delta of timestamps that are used to set pattern start and end entry.
TimeFrame $\geq$ [(end entry).timestamp - (start entry).timestamp]

For defining a pattern, we rely on the following two assumptions.

1. For every two entries in a log file, if they employ the same operation, then their number of operands, order of operands, and type of operands are the same.
2. Each log file entry has all the needed operands to perform the event transaction.

Both these assumptions are logical when considering an event log. First, the operands characterize an operation, hence it makes sense to assume that entries with the same operation have the same set of operands. As well, there is no reason to believe that the order in which the operands are given in the log file varies in different entries. Second, we consider a complete log file without missing information.
Based on the above definitions and assumptions, we may now define a pattern. We consider a pattern as a combination of entries that satisfy certain conditions. For the entries to relate to a single "logical" activity, they need to (a) relate to the same set of operands, and (b) include repetition in the values of some fields and some fields whose values differ. Fields whose value does not change in the pattern are termed *invariant* while the others are termed *variant*.

For a legal pattern the user must be invariant and the timestamp variant with a timeframe smaller than a user defined constant. In addition, the union of operation and operands must have at least one invariant field and one variant field. We represent a pattern as an entry whose components are sets that can be variant or invariant. Note that:
If *S* is a set such that $S \in \{invariant\}$ then $|S| = 1$
If *S* is a set such that $S \in \{variant\}$ then $|S| \geq 1$
$ORSO$ includes the same operand types for all entries. Then a pattern is formally defined as:
Pattern = $< User, Timestamps, Operations, ORSOs >$:

$$User \in \{invariant\}, \quad Timestamps \in \{variant\}, \quad ORSO \neq \emptyset,$$
$$(\text{Operations} \cup \text{ORSOs}) \cap \{invariant\} \neq \emptyset$$
$$(\text{Operations} \cup \text{ORSOs}) \cap \{variant\} \neq \emptyset$$

The order of components in the pattern is the same as in the entry. A log file entry is by definition a trivial pattern.

3.2 Pattern Finder Algorithm

For two given patterns (entryA and entryB), we will determine if their composition yields a pattern using the algorithm DIPFinder depicted in Fig. 3.
The algorithm verifies that the entries have the same user and fall within the predetermined timeframe. Then it goes through their operations and list of operands, compares their values, and classifies them as variant or invariant. If there is at least one variant and at least one invariant, the algorithm returns the pattern (specified as a combined entry).

Entries that were recognized as patterns will be considered as a single entry for the next iteration of recurrence. The algorithm uses a variable patternEntry that contains the specific values of the pattern invariants and sets of values for the pattern variants. This variable will be returned by the function in order to be used by the algorithm in the next iteration.

3.3 DIPFinder Example

We demonstrate the algorithm by applying it to data from Table 1 as inputs. Fig. 4 shows the entries that relate to rows 1-6 in Table 1. The rest of the entries can be similarly analyzed.

```
DIPFinder  entry  (entryA, entryB, TimeFrame)
   If (entryA ≠ Empty and entryB ≠ Empty)
         Tf = max(entryA.timestamp ∪ entryB.timestamp)- min(entryA.timestamp ∪
entryB.timestamp);
      If  Tf ≠ 0 and entryA.user = entryB.user and  Tf  <  TimeFrame then
         invariantCounter ← 0;
         variantCounter ← 0;
         patternEntry ← Empty;
         patternEntry.user = entryA.user;
         patternEntry.timestamp = entryA.timestamp ∪ entryB.timestamp;
         opSetLength ← entryA.ORSO.length;
         If entryA.oprtSet = entryB.oprtSet or
             entryA.oprtSet ⊂ entryB.oprtSet or
             entryB.oprtSet ⊂ entryA.oprtSet  then
            invariantCounter++;
                patternEntry.operation = entryA.operation ∪ entryB.operation;
         Else
               variantCounter++;
               patternEntry.oprtSet = entryA.oprtSet ∪ entryB.oprtSet;
         End
         For i = 1 → opSetLength do
                      If entryA.ORSO[i].ordSetValues =
entryB.ORSO[i].ordSetValues or
                         entryA.ORSO[i].ordSetValues ⊂
entryB.ORSO[i].ordSetValues or
                         entryB.ORSO[i].ordSetValues ⊂
entryA.ORSO[i].ordSetValues then
                         patternEntry.ORSO[i].ordSetValues ←
                               entryA.ORSO[i].ordSetValues ∪
entryB.ORSO[i].ordSetValues;
                         invariantCounter++;
                       Else                           // we met some new value/s
                         patternEntry.ORSO[i].ordSetValues ←
                               entryB.ORSO[i].ordSetValues ∪
entryA.ORSO[i].ordSetValues
                         variantCounter++;
         End  // end for loop
                If variantCounter = 0 or   invariantCounter = 1 /* no pattern */
                    patternEntry ← Empty;
                End
                return patternEntry
           End
   End
   return Empty
```

Fig. 3. DIPFinder algorithm

```
(1) : < YPRESS, '13.45.52', PR12, Fredrick, 'Linear Algebra', 'MIS Major'>
(2) : < YPRESS, '13.46.26', PR12, Fredrick, 'Algorithms', 'MIS Major'>
(3) : < YPRESS, '13.47.44', PR12, Fredrick, 'Data Structures', 'MIS Major'>
(4) : < YPRESS, '13.49.18', PR11, Fredrick, 'Linear Algebra', 'CS Minor'>
(5) : < YPRESS, '13.49.24', PR11, Fredrick, 'Algorithms', 'CS Minor'>
(6) : < YPRESS, '13.49.31', PR11, Fredrick, 'Data Structures', 'CS Minor'>
```

Fig. 4. Log entries for rows 1-6 in Table 1

We select entries (1) and (2) as first inputs to our algorithm. Timeframe for the process is set to 20 minutes. The output is the combined entry (1, 2).

DIPFinder [(1): < YPRESS, '13.45.52', PR12, Fredrick, 'Linear Algebra', 'MIS Major'>,
(2): < YPRESS, '13.46.26', PR12, Fredrick, 'Algorithms', 'MIS Major'>] →
(1, 2) : < YPRESS, ('13.45.52', '13.46.26'), PR12, Fredrick, (L.A., 'Alg.'), 'MIS Major'>

We will now apply the algorithm again to the combined entry (1, 2) and to entry (3).

DIPFinder [(1, 2), (3)] =

[(1, 2) : < YPRESS, ('13.45.52', '13.46.26'), PR12, Fredrick, (L.A., 'Alg.'), 'MIS Major'>,
(3): < YPRESS, '13.47.44', PR12, Fredrick, 'Data Structures', 'MIS Major'>] →
(1, 2, 3): < YPRESS, ('13.45.52', '13.47.44'), PR12, Fredrick, ('L.A.', 'Alg.', 'DS'), 'MIS Major'>

A similar processing of the entries 4, 5, and 6 yields the following output:

(4, 5, 6): < YPRESS, ('13.49.18', '13.49.31'), PR11, Fredrick, ('L.A.', 'Alg.', 'DS'), 'CS Minor'>

Next, we try to process together the pattern entries (1,2,3) and (4,5,6).

DIPFinder [(1, 2, 3), (4, 5, 6)] =

[(1, 2, 3) : < YPRESS, ('13.45.52', '13.47.44'), PR12, Fredrick, ('L.A.', 'Alg.', 'DS'), 'MIS Major'>,
(4, 5, 6) : < YPRESS, ('13.49.18', '13.49.31'), PR11, Fredrick, ('L.A.', 'Alg.', 'DS'), 'CS Minor'>] →
(1, 2, 3, 4, 5, 6): < YPRESS, ('13.45.52', '13.49.31'), (PR12, PR11), Fredrick, ('L.A.', 'Alg.', 'DS'), ('MIS Major', 'CS Minor') >

We have a pattern in which the User is invariant, the start and end times meet the limits of TimeFrame, the operation is variant (PR12, PR11), and there is at least one invariant operand – the student 'Fredrick'. With this recognition of pattern we can draw a conclusion that this is a set of related activities, which may stand for one “logical” activity which is inefficiently performed by the users. To make further conclusions we have to determine what the purpose of this set of activities is, or basically what it does. Section 4 deals with this question.

While the DIPFinder algorithm is capable of incrementally aggregating log entries into a pattern, some higher-level algorithm is still needed for managing the entire log file, and particularly for reducing the complexity of the search. This algorithm, which is currently under development, will be a version of a divide and conquer algorithm. It will recurrently employ DIPFinder for combinations of entries whose size increases gradually until all patterns are identified.

4 Utilizing the identified patterns for process improvement

Having identified patterns in the log file, it is still not certain that they really stand for a “workaround” of the limitations imposed by the enterprise system. It may be possible that they reflect the normal and expected way the business process should be performed. For example, when a student registers to a number of courses at the beginning of a semester, this will be manifested as a pattern in the log file. Nevertheless, this is a series of operations which should be performed sequentially and do not require process improvement. Hence, patterns that are identified serve as a basis for interviews with the system users, to verify that they stand for inefficiencies in the business processes.

Once patterns that stand for inefficient process execution are identified, the process can be improved by introducing changes to the enterprise system. Such changes can be, considering our example, a designated user interface in which the user indicates the student whose program should be changed as well as the source and target programs. The attaching and detaching of courses is then automatically performed by the system. However, since many such patterns may be identified, some prioritization should be made for performing the required changes. For this purpose, we propose the following prioritization rule.

Assuming the log file relates to a given period of time (e.g., a month), it is possible to calculate the following metrics:

The *count* of a pattern: given a pattern P, its count C_P is the number of times the pattern appears in the log file.

The *average size* of a pattern: given a pattern P, its average size AS_P is the average number of entries it includes. Let P occur C_P times in a log file, so occurrence i includes n_i entries. Then $AS_P = \frac{1}{C_P}\sum_{i=1}^{C_P} n_i$.

The *weighted count* of a pattern (weighted by size): $SC_P = AS_P*C_P$.

Priority for process improvement can be given to patterns whose occurrence is frequent and which entail a relatively large number of entries, namely, patterns whose weighted count is high. Alternatively, it is possible to consider the actual time span of a pattern (average or median) instead of the count. Such a measure does not assume that the entries of different patterns are equally time-consuming.

Note that the patterns and the proposed priority rules are merely an indication of potential improvement. Usually, when metrics are not applied, prioritization can only rely on human considerations. These are influenced by the interaction with the system users who raise their complaints. The proposed rules provide an objective measure which can be used, possibly in addition to other prioritization considerations. Additional considerations are mainly related to specific business and organizational priorities which can only be assigned by humans in the organization.

5 Related work

The approach presented in this paper relates to the area of process mining, since it analyzes data in a system log in order to get some understanding about a business process. In this section we review process mining literature to establish the unique features of our approach.

Process mining primarily aims at discovering a process model based on the process reflection in an event log of a system. Processes that are actually performed by users have in most cases a flow which is different than the flow that the process designing team has thought of. Process mining is capable of discovering these actual flows and composing an actual process model. The motivation for developing this approach was to find an alternative way of analyzing processes in less time than the traditional way of interviews and observations. Creating a workflow design is a complicated time-consuming process and typically there are discrepancies between the actual workflow processes and the processes as perceived by the management [18]. In addition the analysis made by people is error prone, may lead to inconsistencies between individual views of the same process, and is subject to possible incompleteness of information collected from employees about the process [8].

An early work that relied on event logs for discovering behavioral patterns was reported in [9]. The technique is based on a probability analysis of the event traces. Metrics such as frequency and regularity of the event occurrences behavior were saved by the system. This technique is useful in many tasks of software engineering, including architecture discovery, reengineering, user interaction modeling, and software process improvement.

Relating specifically to business processes, the main challenges involved in extracting a process models include definitions of edge conditions, identifying concurrency of events, and overcoming diversity which leads to complex models that are difficult to interpret. The presence of duplicate activities, hidden activities, and non-free-choice constructs are also challenges when a process mining technique is applied.

Besides the construction of an actual process model, process mining has served for other purposes as well. Delta analysis and conformance testing compares the actual process with some predefined process, and detects differences between the model constructed in the design phase and the actual use that was registered in the log files [1]. Another use of mining techniques was presented in [6]. It focuses on the performer of the event and derives social networks using this information. Another investigated aspect, which is quite close to our focus, is efficiency analysis based on timestamps [3]. Timestamps indicate activities which cause delays in the process. In contrast, we use the timestamps as indication of actions that were performed sequentially and within a short period of time, as representing an inefficient way of performing one “logical” activity.

Pattern discovery is mentioned in several works. Dealing with flexible processes [17], the mining approach is to divide the log file to homogeneous subsets by using a clustering technique, and then to build a process model for each subset. Our pattern discovery approach differs from that since we look for a pattern (subset) performed by a single user, while [17] does not. Pattern discovery is also possible in [7], where the event log is clustered iteratively so each of the resulting clusters relates to a set of cases that can be represented by a process model. This work relies on the frequency of an event for pattern discovery regardless of its type. In contrast, our work identifies a pattern based on event types regardless of their frequency.

Process mining has been used for various domains. In particular, healthcare [13], as an environment of very dynamic behavior, was indicated as a challenging domain, where process mining can significantly contribute. Examples include [12] where process mining techniques discover paths followed by particular groups of patients. Three different perspectives were analyzed using the ProM framework [11]: control flow, organizational, and performance. Another domain where process mining was applied is the public sector [5], where it was used for office work analysis. In the domain of industry and supply chain [14] the discovered process enabled analysis across the supply chain, and could be used as a tool to improve business processes in networked organizations. The application in the software development domain raised several challenges [16]. Since process models and software process models cover different aspects, the work considered the main aspects that can connect between the models such as the control flow aspect, the information aspect which records the data produced by the event, and the organization aspect. This approach is somehow close to our approach, but our goal is different. The use of process mining in the security domain was presented in [2], using process mining techniques to analyze audit trails for security violations. The purpose was to support security levels ranging from low-level intrusion detection to high-level fraud prevention.

Our approach differs from the above reviewed process mining works in two main issues. First, as opposed to the process mining aim of creating a process model, we use the system event log with the aim of discovering a pattern which may reflect a single activity from the user's point of view. Hence, the focus of our approach is narrower than the entire process model aimed at by process mining approaches. Second, the specific use for which these patterns are intended is the identification of process inefficiencies resulting from a lack of system support. This specific use has not been proposed yet.

6 Conclusions

The paper deals with two ways in which technology can drive business processes. First, the introduction of an enterprise system results in changes in the business processes. However, these are not necessarily desirable changes. Second, mining technology can be utilized in such situations as a driver for process improvement.

The problem of inefficient processes as a result of enterprise system adoption is very common in practice (e.g., 10]), and, to the best of our knowledge, has not received a technology-based solution so far. One contribution of the paper is, therefore, making this problem explicit and discussing it. Besides that, the main contribution of the paper is the approach proposed for addressing such situation. This includes (a) a clear definition of the reflection of inefficient workarounds as patterns in an event log of the system, (b) an algorithm for pattern identification, and (c) rules for prioritizing improvement requirements.

The algorithm presented here is still an initial step towards a complete and efficient algorithm, needed for addressing the high volume of data in a real system log file. In future, we intend to complete the development and implementation of the algorithm and to apply it to real data of the university case study, as well as in other domains.

References

1. van der Aalst W.M.P.: Business alignment: using process mining as a tool for Delta analysis and conformance testing. Requirements Engineering Journal 10(3), pp.198--211. (2005)
2. van der Aalst W.M.P. and de Medeiros A.K.A.: Process Mining and Security: Detecting Anomalous Process Executions and Checking Process Conformance. Second International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004), N. Busi and R. Gorrieri and F. Martinelli, STAR, Servizio Tipografico Area della Ricerca, CNR Pisa, Italy, pp. 69--84. (2004)
3. van der Aalst W.M.P and van Dongen B.F.: Discovering Workflow Performance Models from Timed Logs. In: Y. Han, S. Tai, and D. Wikarski, (eds.), International Conference on Engineering and Deployment of

Cooperative Information Systems (EDCIS 2002), volume 2480 of Lecture Notes in Computer Science, pp. 45--63. Springer-Verlag, Berlin. (2002)

4. van der Aalst W. M. P., Hofstede, A. H. M. ter, Kiepuszewski, B., and Barros, A. P..: Workflow Patterns, Distributed and Parallel Databases. 14(1), p. 5--51. (2003)
5. van der Aalst W.M.P, Reijers H.A., Weijters A.J.M.M., van Dongen B.F., Alves de Medeiros A.K., Song M., and Verbeek H.M.W.: Business Process Mining: An Industrial Application. Information Systems, 32(5) pp. 713--732. (2007)
6. van der Aalst W.M.P, Reijers H.A. and Song M.: Discovering Social Networks from Event Logs. Computer Supported Cooperative Work, 14(6) pp. 549--593. (2005)
7. Alves de Medeiros A.K., Guzzo A., Greco G., van der Aalst W.M.P., Weijters A.J.M.M., van Dongen B., and Saccà D.: Process Mining Based on Clustering: A Quest for Precision. In: A. ter Hofstede, B. Benatallah, and H.-Y. Paik, (eds.), BPM 2007 Workshops, LNCS 4928. pp. 17--29. (2008)
8. Bandinelli, S., Fuggetta, A., Lavazza, L., Loi, M., and Picco, G.: Modeling and improving an industrial software process. In: IEEE Trans. Softw. Eng. 21, 5, pp 440--454. (1995)
9. Cook J.E. and Wolf A.L.: Discovering Models of Software Processes from Event-Based Data. ACM Transactions on Software Engineering and Methodology 7(3) pp. 215--249. (1998)
10. Davenport, T.: Putting the Enterprise into the Enterprise System. Harvard Business Review 76(4) pp. 121--131. (1998)
11. van Dongen B.F., de Medeiros A.K.A., Verbeek H.M.W., Weijters A.J.M.M. and van der Aalst W.M.P.: The ProM framework: A new era in process mining tool support. In: 26th International Conference on Applications and Theory of Petri Nets (ICATPN 2005), G. Ciardo and P. Darondeau, LNCS 3536, pp. 444--454. (2005)
12. Mans R.S., Schonenberg M.H., Song M., van der Aalst W.M.P., and Bakker P.J.M..: Process Mining in Health Care. In: L. Azevedo and A.R. Londral, (eds.), International Conference on Health Informatics (HEALTHINF’08), pp. 118--125, Funchal, Maldeira, Portugal, January 28-31. (2008)
13. Maruster L., van der Aalst W.M.P., Weijters A.J.M.M., van den Bosch A. and Daelemans W.: Automated Discovery of Workflow Models from Hospital Data. In: B. Kröse, M. de Rijke, G. Schreiber, and M. van Someren, (eds.), Proceedings of the 13th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC 2001), pp. 183--190. (2001)
14. Maruster L., Wortmann J.C., Weijters A.J.M.M., and van der Aalst W.M.P.: Discovering Distributed Processes in Supply Chains. Proceedings of the

International Conference on Advanced Production Management Systems (APMS 2002), pp. 119--128. (2002)

15. Parr, A.N. and Shanks, G.: A taxonomy of ERP implementation approaches. Proceedings of the 33rd Annual Hawaii International Conference on System Sciences,Volume 1, IEEE Press, pp. 1--10. (2000)
16. Rubin V., Günther C.W., van der Aalst W.M.P., Kindler E., van Dongen B.F., and Schäfer W.: Process Mining Framework for Software Processes. In: International Conference on Software Process, Software Process Dynamics and Agility (ICSP 2007), volume 4470 of Lecture Notes in Computer Science, pp. 169--181. Springer-Verlag, Berlin. (2007)
17. Song M., Günther C.W. and van der Aalst W.M.P.: Trace Clustering in Process Mining. 4th Workshop on Business Process Intelligence (BPI 08). (2008)
18. Weijters A.J.M.M., van der Aalst W.M.P.: Process mining: discovering workflow models from event-based data. In: B. Kröse, M. de Rijke, G. Schreiber, M. van Someren (eds.), Proceedings of the 13th Belgium–Netherlands Conference on Artificial Intelligence (BNAIC 2001), pp. 283--290. (2001)