Spotting the Weasel at Work: Mining Inappropriate Behavior Patterns in Event Logs*

Saimir Bala[1][0000−0001−7179−1901], Tim Jacobowitz[1], and Jan Mendling[1,2][0000−0002−7260−524X]

[1] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
firstname.lastname@hu-berlin.de
[2] Security and Transparency in Processes, Weizenbaum Institute, Berlin, Germany

Abstract. Diverging interests in the workplace may lead to undesirable employee behavior such as taking undue credit, underperforming, shirking responsibilities, and undermining colleagues. This kind of conduct, also referred to as *weasel behavior*, can have significant negative implications for both individuals and the organization as a whole. Therefore, its identification is of crucial importance. Recent work in process science has defined thirteen weasel behavior patterns and proposed the use of process mining related techniques to uncover them from the traces recorded by information systems. However, these definitions have not yet been tested on any event log. This paper aims at closing this gap by providing the design specifications and algorithms necessary to extract weasel behavior from event logs. We evaluate our implementation on the real-world logs provided by the IEEE Task Force on Process Mining and report the extent of weasel behavior present in each dataset. Our results have relevant implications on the application and development of resource-centered process analysis techniques and contribute to better understanding the information present in the widely-used BPI logs.

Keywords: event log · resource analysis · process mining · behavioral process mining

1 Introduction

When stress and conflicts of interest arise in workplace situations, individuals may exhibit inappropriate behavior. This conduct, often labeled as *weasel behavior*, includes activities that are unsanctioned and unrelated to work during work hours. Addressing weasel behavior is crucial for maintaining a productive and healthy workplace, ensuring legal compliance, and protecting employee well-being.

Traditionally, detecting this behavior has been challenging due to individuals' tendencies to conceal it. However, the advent of information systems that support

* The research of Jan Mendling was supported by the Einstein Foundation Berlin under grant EPP-2019-524, by the German Federal Ministry of Education and Research under grant 16DII133, and by Deutsche Forschungsgemeinschaft under grants 496119880 (VisualMine) and 531115272 (ProImpact).

work activities now allows for the recording of traces of these actions, providing an opportunity to identify undesired patterns. Recent studies [10] have theoretically defined various patterns of weasel behavior, using a Principal-Agent [12] framework to analyze resource interactions in event logs. Thirteen specific patterns have been identified, but these have not yet been implemented as algorithms or tested in real-world event logs.

This paper aims to bridge this gap by providing a specification for these behavior patterns, developing corresponding algorithms, and evaluating them against both synthetic and real-world event logs. The results reveal the distribution of these behaviors in well-known event logs, thereby enhancing our understanding of resource behavior in organizational settings and contributing to the development of resource-centered process analysis techniques.

This paper is structured as follows.Section 2 describes the setting of this work. Section 3 outlines the research methodology devised to translate the weasel-behavior patterns into design patterns for implementation purposes. Section 4 provides the specifications for each design pattern. Section 5 outlines the results of our algorithms on both synthetic and real-world event logs, including a discussion. Section 6 concludes the paper.

2 Literature Review

The scope of this paper is to explore the use of event logs for extracting and analyzing the behavior of resources. Therefore, we consider the literature that takes into account both the resource perspective and the analysis of event logs, at the same time. This section describes previous work, discusses works related to the analysis of behavioral issues and identifies the research gap to address.

Previous work. This paper follows the research stream started by Leyer et al. [10]. They use Principal-Agent Theory [12] to explain the relationship between resources in an organization. This theory states that there is an imbalance in power held by the principal and in information held by the agent, and their goals. Given that principals cannot fully control the actions of agents, the latter will show opportunistic behavior in to attain their individual goals. The propensity of employees to engage in unsanctioned, non-work related activities during work time has also been referred to as *weasel* behavior. Thanks to the adoption of information systems and the recording of work-related events into system logs, this behaviour can now be uncovered. Leyer et al. [10] have conceptualized this inappropriate behavior into the thirteen patterns provided in Table 1.

Related work. Apart the work of Leyer et al. [10], other related work exists that allows to evaluate resources' behavior. Specifically, in the process mining area, *social network analysis* has been used to examine the interactions and relationships between the resources of a process. Specifically, the work of [2] uses event logs to discover social networks within organizations. Song and van der Aalst [14] developed techniques to automatically extract social networks from event logs of workflow management systems. Moreover, to overcome the complexity of networks discovered in large event logs, Ferreira and Alves [7] focus

Table 1: The thirteen patterns of inappropriate behavior as defined by [10]

Rerouting-related	**Performance-related**	**Social-related**
1. Activity Deviation	5. Performance Masking	9. Idling
2. Originator Deviation	6. Performance Blow-out	10. Social Loafing
3. Re-Ordering	7. Overwork Hiding	11. Peer Mobbing
4. Preferential Work Selection	8. Gold Plating	12. Boss Mobbing
		13. Social Borrowing

on discovering communities at varying levels of abstraction. More recent work by Mustroph et al. [13] also considers additional information such as natural language description of the process to identify whether the mined work is deviating from the wanted behavior. Further approaches that inform on resources behavior are the use of standard process mining techniques [1], where the resource information is used as a case identifier. Techniques like the dotted chart [15] are able to pinpoint single events that may point to unwanted behavior.

Moreover, process performance indicators [6] and cycle time analysis [9] can help at pointing out anomalies in resources performance. To aid this kind of analysis, *simulation* [11,3] is a powerful technique that can be used for identifying improper behavior of resources.

Gap. Among the above mentioned approaches, the only work that adopts a theoretical lens to analyze the resource behavior is [10]. While conceptualized and exposed in a structured manner, the thirteen proposed patterns have not yet been implemented. Therefore, we pose the research question *how can we implement such patterns and to what extent can be mined from event logs?*. This paper aims at closing this gap by providing a specification that can be easily implemented as a prototype, an initial implementation and an application on real-world event logs that are commonly used in business process research.

3 Methodology

Next, we present our research approach to achieve the design patterns specifications. We describe the steps undertaken, the data and the framework used to specify the design patterns.

Research Approach. We summarize the steps of our approach in Figure 1. To start, we consider the behavior patterns conceptualized in [10] as listed in Table 1. Then, we proceed in five steps *i)* analyze pattern description; *ii)* derive requirements for identification; *iii)* derive event log requirements; *iv)* formulate design patterns specification; and *v)* implement and test a pattern detection algorithm.

Let us describe each step in more detail. In step *i)* we analayze the original description of each pattern guided by the question *what are the inputs required to apply this pattern?*. As a result, we can first classify the patterns whether they can directly be implemented with only the event log or whether they require more contextual information such as a process model. In step *ii)* we consider

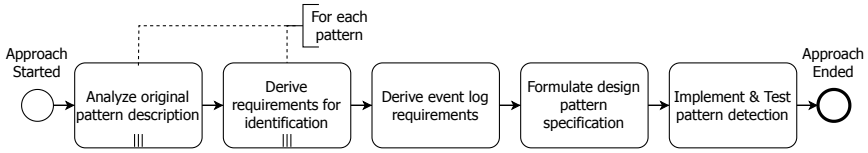

Fig. 1: Steps of the approach to specify and implement the patterns.

again each pattern and the information from step *i)* and we identify specific requirements of the input. For example, in order to detect *performance blow-out* the requirement is that the event log contains the attributes `Activity`, `Start` and `Complete Timestamps`, and `Resource`. In step *iii)* we consider the overall knowledge gathered by analyzing each pattern to derive general requirements about the input. This step is guided by the question *can we create one single input with all the required attributes?*. The result of this step is also related to the choice of what specification language used in the next step. In step *iv)* we use a patterns specification language to describe each pattern in a way that it can be easily implemented. In the final step *v)* we implement and test each pattern, according the specification. More specifically, before implementing and testing algorithms in the real data, this step creates synthetic event logs that present the required behavior (by construction). These synthetic event logs are then used to test the correctness of the algorithm in identifying the implemented behavior. The output of this step is further used to analyze the presence of the patterns.

Data, Design Patterns Selection and Specification Format. Next, we describe the kind of data used in this research approach, what design patterns we used for the specification of weasel behavior and their format.

Data. The first three steps of the reseach approach allow us to grasp an overall understanding of the data requirements for weasel behavior detection. In particular, our approach makes use of two kinds of data: synthetic event logs and real-life event log. Sythetic event logs are generated ad-hoc for each pattern. More specifically, we first create scenarios for each instance of inappropriate behavior. Then we manually create event log traces that directly reflect the scenario, for each pattern. In other words, in the end of our approach, we have 13 synthetic event logs (one per pattern), each affected by the respective inaproppriate behavior.

Design patterns selection. Before applying step *iv)*, there is a need for selecting how to transform the conceptualized patterns into a design that can be used for software specification. As the final goal is to implement and test the behavioral patterns in practice, we follow the principles of Design Patterns proposed by Gamma et al. [8]. The main advantange of specifying the thirteen inappropriate behavior types as design patterns is that they can be used to both generally describe the problem in context and as *requirements for algorithm development.*

Specification format. We keep the original classification of the patterns, but we use a design pattern specification format. Each pattern is described according to the following template, inspired by [8] and adapted to our case. `Pattern Name` conveys the essence of the design pattern, `Intent` gives information on the intent or purpose, `Problem` states the specific problem the pattern addresses, `Solution`

gives a details on the solution provided by the pattern, `Required Attributes` lists the attributes required in the event log, `Required Analysis` lists the minumum level of analysis an implementation must provide, `Implementation` explains what are the steps and considerations for implementing the pattern, `Code Examples` provide code snippets or pseudocode to illustrate the implementation, `Sample Applications` descibe real-world examples or scenarios where the pattern can be applied, and `Consequences` outlines benefits and potential drawbacks of using the pattern.

4 Design Patterns Specifications

In the following, we provide the specifications for each inappopriate behavior, using the design patterns specification format. For the sake of space, we do not report the implementation, code examples, sample applications and consequences. The implementation and code can be found in [4] wheras applications to real-world event logs and consequences follows in the paper.

Pattern 1. Activity Deviation (Rerouting-related)
`Intent:` This design pattern captures all those activities that are unexpected or undesired by the principal (e.g., the owner of the organization).
`Problem:` The specific problem addressed by this pattern is the conformance of activities to a desired process known by the principal. Specifically, the process model represents the desired work.
`Solution:` The solution provided by this pattern is to perform conformance checking of the activities present in the event log against the given process model. The priority is given to the process model. That is, if an activity is only present in the log, it is considered a deviation.
`Required Attributes:` Activity, Process Model, or a similar data structure containing all expected activities
`Required Analysis:` The activity has to be checked for each event in the log. The regarded activity then has to be compared with the expected activities specified in the model
`Implementation:` Loop through all activities in the event log. Each considered activity from the event log, should be the next expected activity in the model. If this is not the case, record the deviation. Collect all the deviations for all activities.

Pattern 2. Originator Deviation (Rerouting-related)
`Intent:` The purpose of this pattern is to detect those resources who worked on tasks they were not assigned to.
`Problem:` While it may be desirable that certain resources undertake tasks they were not assigned to, the focus of this pattern is on those cases when certain resources favour doing other tasks instead of their own, as this may give them more credit.
`Solution:` The solution provided by this pattern is to perform conformance checking. Especially, the check should be whether the resources who were assigned

certain tasks were also the ones who conducted them.
`Required Attributes:` Activity, Resource, Process Model (or a similar data structure containing the assigned resource for each activity)
`Required Analysis:` Check each activity in the event log to see if it was conducted by the assigned resource. Record any deviations where tasks were completed by non-assigned resources.
`Implementation:` To detect the pattern of Originator Deviation, the assigned resource has to be checked for each event in the log. The resource associated with the regarded event should then be compared to the resource who is expected for the corresponding activity according to the model. If the assignments from log and process model deviate, it is an indicator of the presence of Originator Deviation. Otherwise, it has to be proceeded to the next assignment.

Pattern 3. Re-ordering (Rerouting-related)
`Intent:` The purpose of this pattern is to detect those activties that were performed in an undesired order.
`Problem:` The responsible agent may think that the activities would be better performed in a different order than what is prescribed in the process model.
`Solution:` The solution provided by this pattern is to perform conformance checking. Especially, this pattern should provide discrepancies in terms of sequences of activities performed in the event log versus the model.
`Required Attributes:` Case, Activity, Timestamps, Process Model (Or a similar data structure containing the expected order of activities)
`Required Analysis:` The sequences of activities in the event log, sorted by timestamp must be compared to the expected sequences of activities prescribed in the process model.
`Implementation:` To detect the pattern of re-ordering, it is necessary to check the order in which activities occurred for each case by examining the timestamps. This order then has to be compared to the order prescribed in the process model. If the order of activities in the regarded case from the log is invalid according to the process model, this is an indicator of the presence of Re-Ordering. Otherwise, it has to be proceeded to the next case.

Pattern 4. Preferential Work Selection (Rerouting-related)
`Intent:` The purpose of this pattern is to detect those resources who choose working on certain tasks disproportionally often.
`Problem:` Agents may chose easier tasks that have a higher pay-off.
`Solution:` The solution must single out all those cases in which work was not performed as assigned (i.e., not in a first-come-first-served (FCFS) fashion).
`Required Attributes:` Case, Activity, (Start and Complete) Timestamps, Resource
`Required Analysis:` Output a list of activities that are chosen on average more often than expected by the resources
`Implementation:` To detect Preferential Work Selection, the frequency of certain activities being chosen by certain resources has to be observed. Average

frequencies have to be calculated for each activity on this basis. If the frequency of a resource choosing an activity is significantly greater than the average value for the regarded activity, this is an indicator of the presence of Preferential Work Selection. Otherwise, it has to be proceeded to the next frequency of a resource choosing an activity. Furthermore, it also has to be observed if a resource starts a new activity in a case while still not being finished with work in another case by checking timestamps. If a timestamp reveals that this did indeed happen, this is another indicator of the presence of Preferential Work Selection. Otherwise, it has to be proceeded to the next timestamps.

Pattern 5. Performance Masking (Performance-related)
`Intent:` The purpose of this pattern is to point out agents who try to prevent their performance from being measurable.
`Problem:` An agent could transfer the content of an online work item to finish it offline because the time needed is only measured when the work item is opened online. This way, the actual time needed for the completion of the task can not be measured.
`Solution:` The solution should provide information on how quickly the tasks are performed. Unusually short durations should be collected for further analysis
`Required Attributes:` Case, Activity, (Start and Complete) Timestamps, Resource
`Required Analysis:` Output a list of cases with many events registered in a short time
`Implementation:` To detect the pattern of Performance Masking, cases that have many events registered in a relatively short amount of time have to be observed by counting events in cases and checking timestamps. Activities that occur significantly often in such cases and also have a significantly short duration should then be searched. If the presence of such cases and activities is confirmed, it is an indicator of the presence of Performance Masking. Otherwise, it has to be proceeded to the next cases with many events in a short time.

Pattern 6. Performance Blow-Out (Performance-related)
`Intent:` The purpose of this pattern is detect agents stretching their work time by pretending they are still working on an activity when they are actually not.
`Problem:` Activities might have working times determined by service-level-agreements, which makes it possible to maximize the time spent on that activity even when the work is already completed. This suggests that the agent might want to gain additional free time.
`Solution:` The solution should provide information on agents who, compared to others, take longer time on performing similar tasks.
`Required Attributes:` Case, Activity, (Start and Complete) Timestamps, Resource
`Required Analysis:` Output all those resources that took different times in peforming similar tasks, along with the tasks performed.
`Implementation:` To detect the pattern of Performance Blow-Out, the time

different resources need for the same activities has to be inspected by checking timestamps. Those completion times of different resources must then be compared to determine whether they deviate significantly from one another or are approximately in the same range. If the standard deviation of the completion times for an activity is significantly great, this is an indicator of the presence of Performance Blow-Out. Otherwise, it has to be proceeded to the next completion times of an activity. Moreover, the completion times of the same resource for the same activity have to be analyzed. If this duration gets significantly longer over time, this is another indicator of the presence of Performance Blow-Out. Otherwise, it has to be proceeded to the next completion times.

Pattern 7. Overwork Hiding (Performance-related)
`Intent:` The purpose of this pattern is to signal agents performing work outside of their official working times.
`Problem:` Agents are not able to finish their allocated work in the expected time. Since the expected times would usually be set based on the stated skills and experience of an employee, Overwork Hiding might even infer that an agent is less skilled or experienced than stated expected by the principal.
`Solution:` The solution should provide information on agents that perform actions on work items outside of working hours
`Required Attributes:` Activity, Timestamps, Resource, Official working times
`Required Analysis:` Output all the agents that performed work on activities outside working hours
`Implementation:` To detect the pattern of Overwork Hiding, the timestamps have to be investigated for each resource and compared with the working times of the regarded resource. If a timestamp is found that is placed outside of the allocated working times, this is an indicator of the presence of Overwork Hiding. Otherwise, it has to be proceeded to the next timestamp.

Pattern 8. Gold Plating(Performance-related)
`Intent:` The purpose of this pattern is to show circumstances of overwork in which resources wether conduct more work than necessary or perfrom additional services and tasks
`Problem:` By conducting more work or providing additional services, agents may increase the favour of certain clients or aim for additional peformance bonus
`Solution:` Cases with additional duration or cost must be analyzed. Such cases would take longer in certain contexts (e.g., when associated to a specific client)
`Required Attributes:` Activity, Timestamps, Resource, Official working times
`Required Analysis:` Perform variant analysis and point out cases that last significantly longer from others in the same variant group.
`Implementation:` To detect the pattern of Gold Plating, cases categorized by their specific sequence of activities, also called process variants, have to be identified. Next, the average activity duration of each process variant has to be observed. If the average activity duration of a certain process variant is significantly longer than the overall average, this is an indicator of the presence of Gold Plating.

Otherwise, it has to be proceeded to the next process variant. Furthermore, it has to be searched for process variants containing significantly rare activities. If a process variant contains an activity that is significantly rare, this is another indicator of the presence of Gold Plating. Otherwise, it has to be proceeded to the next process variant.

Pattern 9. Idling (Social-related)
`Intent:` The purpose of this pattern is to point out agents engaging in non-work related activities during work time instead of working
`Problem:` Agents may perceive work as boring or unfair and use working time for other activities, such as socializing, procrastinating, smoking, etc. Yet, they do not want to be perceived as bad performers.
`Solution:` A solution to this problem would be to assign more stimulating work to resources. Resources that take substantially more time on completing tasks can be extracted from event logs.
`Required Attributes:` Activity, (Start and Complete) Timestamps, Resource
`Required Analysis:` Analyze the actions performed by resources on tasks. Check completion times.
`Implementation:` To detect the pattern of Idling, the timestamps have to be checked to calculate the completion times of each resource for each activity. Based on those completion times, the mean has to be calculated for each activity to be compared with the individual completion times. If there is a completion time value that is significantly greater than the values for other resources conducting the same activity or significantly greater than the values for the same resource conducting other activities, this is an indicator of the presence of Idling. Otherwise, it has to be proceeded to the next completion times. In addition, the timestamps have also be evaluated to detect resources taking possible breaks between two activities. If those breaks are significantly long, this is another indicator of the presence of Idling. Otherwise, it has to be proceeded to the next timestamp.

Pattern 10. Social Loafing (Social-related)
`Intent:` The purpose of this pattern is to detect when, in the context of group work, an agent avoids, neglects works or free-rides
`Problem:` As group members do not perform team work by shirking work, the whole team productivity is affected
`Solution:` Identify resources that exhibit this pattern and assign them to different groups.
`Required Attributes:` Activity, (Start and Complete) Timestamps, Resource, Classifications of events into group work and individual work
`Required Analysis:` Measure the average completion times of resources in the context of group work
`Implementation:` The completion times must be measure for each resource in the context of groupwork. This performance must then be compared to the completion times of the same resources working alone. If a resource shows a significantly better individual performance while working alone in comparison

to working in a group, this is an indicator of the presence of Social Loafing. Otherwise, it has to be proceeded to the next resource.

Pattern 11. Peer Mobbing (Social-related)
`Intent:` The purpose of this pattern is to detect agents or groups of agents who degrade their peers by taking over their work without consent.
`Problem:` Agents steal work items from the entitled resource in order to increase their pay-off, at the same time making the performance of their colleagues look poor.
`Solution:` Resource who exhibit this pattern must be detected and a limit on the items they can take over should be in place.
`Required Attributes:` Activity, Resource
`Required Analysis:` Collect work type that are performed unusually often by certain groups.
`Implementation:` To detect the pattern of Peer Mobbing, it is necessary to analyze the frequency of each activity being performed by each resource. If significantly many resources perform a certain activity significantly more often than a single resource, this is an indicator of the presence of Peer Mobbing. Otherwise, it has to be proceeded to the next resources.

Pattern 12. Boss Mobbing (Social-related)
`Intent:` The purpose of this pattern is to detect agents or groups of agents who repeatedly perform poorly as a team in order to make their boss look bad.
`Problem:` The boss may be too demanding or controlling and a group of agents decides to underpeform so that companies goals are not met.
`Solution:` Once these team member are detected, a solution is to assign people to different teams.
`Required Attributes:` Activity, (Start and Complete) Timestamps, Resource, Timestamp when a certain is assigned, Classifications of events into group work and individual work
`Required Analysis:` Collect resource peformance before and after the time a boss was assigned.
`Implementation:` To detect this pattern, the average completion times of resources performing in group work, before and after a predefined time, starting when a new boss took over, are analyzed separately. If we observe that resource groups take significantly longer on average after the boss takeover than before, we count this as Boss Mobbing.

Pattern 13. Social borrowing (Social-related)
`Intent:` The purpose of this pattern is to detect agents exploting other agents by letting them do their work without giving undue credit.
`Problem:` Agents may not be able to perform their work, they may be overlaoded or may simply want to increase their performance by offloading work to colleagues.
`Solution:` Agents that exploit other agents must be detected and it should be

made sure that they do not work on similar tasks or similar times
`Required Attributes:` Activity, (Start and Complete) Timestamps, Resource, Official working times
`Required Analysis:` Find correlations in peformance. Especially, when one resource is present, the performance of another resource is negatively affected.
`Implementation:` To detect this pattern, the average completion times of each resource are calculated, for when each other resource is present at work, and when the same resource is not present. These average times are investigated to check if there is a correlation between the presence at work of a resource and the average completion times of another resource. If a resource performs significantly worse when another resource is present versus when this is absent, that is a hit.

5 Evaluation and Analysis

This section evaluates the feasibility of the design patterns and applies the resulting algorithm to real-world logs, further analyzing and discussing the results.

Evaluation Criteria. We evaluate the applicability of the proposed design patterns by implementing them into a prototype. More specifically we implement each of the thirteen patterns in a dedicated Python script. The scripts are available in the GitHub repository linked in [4]. Our key criterion for evaluation is the effectiveness in translating the design patterns to outcomes. To evaluate this, we proceed in two steps. First, we construct synthetic input in which we represent each weasel behavior, for all the thirteen patterns. We then run our prototype on this synthetic data and improve the implementation until we are able to capture all the behavior. Second, we apply our prototypical implementation on real-world event data made available from the IEEE task force on process mining[3].

The most suitable logs for our analysis were the BPI Challenge 2012 log (BPIC12), BPI Challenge 2017 (BPIC17), both representing a loan application process of a Dutch financial institute, Conformance Checking Challenge 2019 log (CCC19) which represents a medical training process, specifically medical students learning how to install a specific catheter, and the Dutch academic hospital (Hospital) log (only used when a timestamp is not required).

Some patterns require not only an event log, but also a model as an input. In these cases, we create a surrogate by sampling the event log. We make various samples and use them as a reference model. This means that we also repeat the application of the algorithms for each sample and collect the average score.

Analysis of the Results. Next, we analyze the discovered weasel behavior in the real-world event logs. We report the results for each pattern on the event logs where the pattern could be applied. For the sake of space we will only use plots

[3] https://www.tf-pm.org/resources/xes-standard/about-xes/event-logs

where it is necessary to show comparison. We supply each result with a dedicated analysis, highlighting key information and limitations.

Pattern 1: Activity deviation. The script implementing the Activity deviation pattern (from exptected work) could be applied to BPIC12 (262200 events), BPIC17 (1202267 events), CCC19 (1394 events) and Hospital (150291 events). No activity deviation could be detected in the former three. In the Hospital log, 282 events were flagged as activity deviation, equals to 0.0019%. **Analysis:** A reason for the low occurence of the activity deviation (as implemented) can be associated to the non-availabilty of a process model. The sampling technique used 50% of the event log to build a model. Such sample may be too large, and include all the activities that are also present in the log. In this sense, every activity of the log is "expected" in the model.

Pattern 2: Originator deviation. Originator deviation (i.e., resources who worked on tasks not initially assigned to them) could be applied to BPIC12, BPIC17, CCC19 and Hospital. In this case, while the Hospital log, only showed 1 case of this pattern, CCC19, BPIC17, and BPIC12 presented the highest presence of the pattern, with 13.79, 12.85 and 6.56% of this behavior out of the total possible. The total possible behavior was calculated considering the product of all orginators multiplied by all the possible activities. **Analysis:** As in the previous case, given the non-availability of a process model, a sampling technique was used. Because the algorithm samples the log more than once to use the sample as a reference model, it is possible that more instances which present this pattern are detected. This may affect positively the number of hit couples, in case the sampled event log contains too many similar traces to the original.

Pattern 3: Reordering. Re-ordering (i.e., activities performed in an unexpected order) could be applied to BPIC12, BPIC17, CCC19 and Hospital. As the percentage of re-ordering cases is higher than the previous patterns, we display them in Figure 2. In BPIC12 log, 1213 out of 13087 cases in total are detected (9.27%). In BPIC17 12890 out of the total 31509 (40.91%) and Hospital log 703 out of 1143 total (61.5%) were detected. Finally, in the CCC19 log all the 20 cases got detected (100%). **Analysis:** As in the previous cases, the detection present potentially false positives. This can also be noted by the 100% value score on the CCC19 event log. However, given the non-availability of a reference process model, this is not possible to measure.

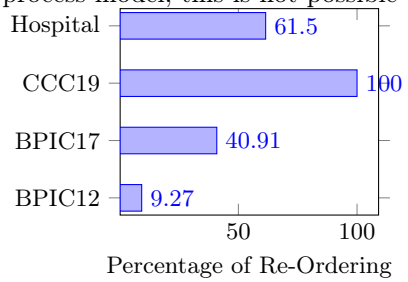

Fig. 2: Re-Ordering percentage in different logs

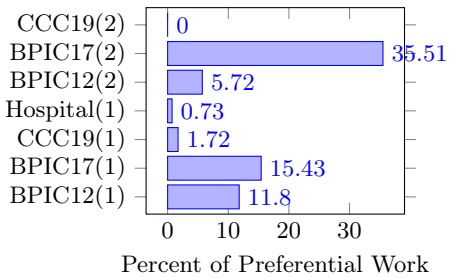

Fig. 3: Preferential Work Selection percentage in different logs

Pattern 4: Preferential work selection. This pattern occurs when i) certain activities are chosen more frequently than others by certain resources or ii) when resources start new activities while still working on other ones. We applied both conditions separately to the to BPIC12, BPIC17, CCC19 and Hospital. Condition ii) could not be applied to Hospital as the start timestamp is not available (thus, it is not possible to understand if resources are working in any other task). We report the results in Figure 3. The number in parethesis expresses which condition was measures. That is, BPIC12(1) means that condition i) was measured. We noticed here that the BPI challenges event logs present the higher occurrence of this pattern. **Analysis:** The values observed in the logs may present false positives or negatives, as the classification of normal frequency depends on a threshold value. Domain experts are required to set this value optimally.

Pattern 5: Performance masking. Given the requirement on start timestamp, this pattern could not be applied to the Hospital log. In other logs it was detected as follows. BPIC12 is affected as 16583 events over 262200 total (6.32%), BPIC17 as 47608 events out of 1202267 (3.96%), and CCC19 as 37 events over 1394 (2.65%). **Analysis:** The percentages are threshold based. A domain-expert can help setting them more optimal results.

Pattern 6: Performance Blow-Out. We report the results in Figure 4. The numbers in parentheses denote which condition of Performance Blow-Out the corresponding bar represents: 1 for the first condition (different resources, same activities) and 2 for the second condition (same resources, same activities). **Analysis:** No instances of Performance Blow-Out are detected in the CCC19 log. In contrast, the remaining two logs exhibit significant proportions of Performance Blow-Out. The proportion is particularly pronounced in BPIC17, with approximately twice the percentage of signelled instances versus BPIC12, for both conditions of the pattern. These findings reveal that resources perform certain activities slower over time in the two logs.

Pattern 7: Overwork hiding. This algorithm was applied to BPIC12, BPIC17, CCC19, and Hospital. The detection was as follows 80250 detected events over 262200 for BPIC12 (30.61%), 298728 events over 1202267 total for BPIC17 (24.85%), 1394 over 1394 for CCC19 (100%), and 150291 over 150291 for Hospital (100%). **Analysis:** There are two primary factors that could contribute to false positives and negatives. Firstly, the code assigns default working times of 09:00 to 17:00 to all resources in the four logs due to the absence of predefined working times. This can lead to events registered within the true working hours being falsely detected as Overwork Hiding. However, this issue could also result in false negatives, where events occurring within the default working hours are not flagged despite being outside of the actual working hours for the respective resource. Timezones recorded in the timestamps are a further source of threat.

Pattern 8: Goldplating. We report the decetion results of this pattern in Figure 5. The numbers in parentheses denote which condition of Gold Plating the corresponding bar represents. Conditions are variants with a significantly i) longer duration and ii) rare activities. The highest percentage of hits for both conditions was observed in the BPIC12 log. Specifically, out of 4371 process variants in the

BPIC12 log, 927 were found to take significantly longer to complete than others, and 2742 were found to contain a significantly rare activity. In the BPIC17 log, although fewer, still significant proportions of positives were identified, with 2699 and 6447 out of 15930 total process variants. Similarly, in the CCC19 log 1 and 10 out of 20 process variants were detected. **Analysis:** These patterns we always possible to detect in the real-world event logs. The significance threshold was set to 1%. Yet, it was still possible to detect this pattern.

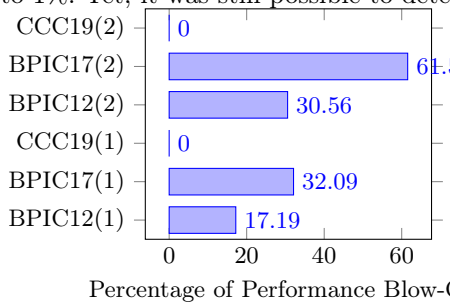

Fig. 4: Performance Blow-Out

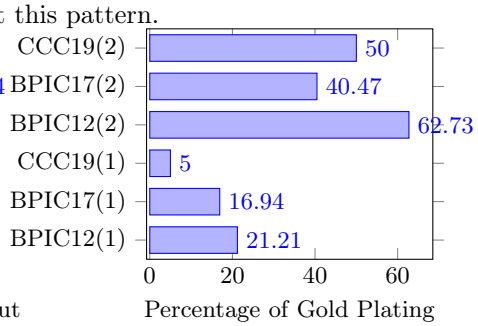

Fig. 5: Gold Plating percentage

Pattern 9: Idling. Durations are analyzed for this pattern, so the Hospital log is not considered. Three conditions are calculated: i) count of instances where a resource requires significantly more time for an activity compared to the average resource, ii) resources that require significantly more time for a particular activity compared to their average time for other activities and iii) instances where resources take significantly long breaks without registering any activity. The results are as follows. BPIC12 (i, ii, iii) = (186, 224, 783), BPIC17 (i, ii, iii) = (455, 579, 397), and CCC19 (i, ii, iii) = (0, 0, 0). **Analysis:** No instances of Idling were detected in the CCC19 log for any of the three conditions. The most significant tendency was found in the BPIC12 log, where approximately 31.52% of possible resource/activity combinations were flagged as resources taking long breaks. The proportions of the remaining two conditions in the same log, as well as all three conditions in the BPIC17 log, are between 7% and 15%.

Pattern 10: Social loafing. This pattern was applied to CCC19, BPIC12, and BPIC2017. No social loafing pattern was detected in the CCC19 log. On the contrary, 42 out of 69 resources in the BPIC12 log and 105 out of 149 resources in the BPIC17 log exhibited such behavior. **Analysis:** Considerable difference was found between BPIC12 and BPIC17 event logs. Respectively, they are affected as much as 68.87% and 70.47%. Possible false positives may be attributed in part to the threshold, as its value determines the extent to which the average completion time of a resource must deviate between group and individual work. It is set to 10 minutes. Furthermore, the group work detection function also presents potential for false positives and negatives. This function checks for four conditions, which may not 100% accurately identify group work flags for all events, leading to potentially skewed results.

Pattern 11: Peer mobbing. This pattern was applied to CCC19, BPIC12, BPIC2017, and Hospital. Peer mobbing was only detected in BPIC12, with a 110 instances. **Analysis:** Two threshold values are used in the code for this pattern,

which bear potential for causing false positives and negatives again, as the ideal value to use for threshold values always depends on the context and is subjective.
Pattern 12: Boss mobbing. Among the 13 patterns, Boss Mobbing could not be applied to the real-world data, as there is no information which resources play this role and since when. No realistic assumption could be made about it.
Pattern 13: Social borrowing. This pattern was applied to CCC19 and BPIC2017. No social borrowing was detected in CCC19. BPIC17 presented 1192 cases of this pattern, equivalent to 5.36%. **Analysis:** A resource may receive multiple flags for a time interval in which it worked slower, especially if there are other resources whose working times align with that interval. This situation can yield false positives. Additionally, the setting of a threshold value to define when the average time difference of the borrowing resource is considered significant, may lead to false positives and negatives.

Table 2: Summary of the percentage of each pattern (P1–13) in event logs.

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
BPIC12	0	6.56	61.5	5.72	6.32	30.56	30.61	62.73	31.52	60.87	4	-	-
BPIC17	0	12.85	100	35.51	3.96	61.54	24.85	40.47	10.24	70.47	0	-	5.36
CCC19	0	13.79	40.91	1.72	2.65	0	1	50	0	0	0	-	0
Hospital	0.0019	0.004	9.27	0.73	-	-	1	-	-		0	-	-

Discussion. Table 2 summarizes main results of the detection of the patterns in the real-world event logs. While subject to limitations, this paper presents a first successful implementaiton of the thirteen behavioral patterns to detect inappropriate behavior at work using event logs. The results indicate varying levels of occurrence and distribution of these patterns across different datasets. While limitations may exist due to assumptions made when applying our algorithms to real-world data, it is still possible to observe a certain degree of presence of inappropriate behavior. This is useful to raise flags for further investigation.

Our results also showcase the challenges associated with detecting these patterns, such as the potential for false positives and negatives due to the subjective nature of threshold values. For instance, the *Performance Blow-Out* pattern's detection relies heavily on comparing completion times of similar tasks across different resources. Variations in these times may not always indicate inappropriate behavior but could be influenced by external factors such as task complexity or resource skill levels. Furthemore, the implementation of these patterns revealed the necessity for contextual information. To avoid misclassification, domain knowledge must be taken into account.

6 Conclusion

In this study, we have developed and implemented a set of thirteen behavioral patterns to detect inappropriate behaviors within organizational event logs, using design patterns to guide our specification. Our results on real-world datasets

reveal various significant occurrences of specific patterns that may have potential impacts in the organization.

The implications of our study point towards proactive management and improvement of work conditions within an organization. Future work should focus on refining the behavioral patterns and validate their applicability across various organizational settings.

References

1. van der Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer (2016)
2. van der Aalst, W.M.P., Reijers, H.A., Song, M.: Discovering social networks from event logs. Computer Supported Cooperative Work (CSCW) **14**, 549–593 (2005)
3. van der Aalst, W.M.: Process mining and simulation: A match made in heaven! In: SummerSim. pp. 4–1 (2018)
4. Bala, S.: Towards mining inappropriate behaviour at work (Jul 2024). https://doi.org/10.5281/zenodo.12656557
5. Brock, M.E., Martin, L.E., Buckley, M.R.: Time theft in organizations: The development of the time banditry questionnaire. International Journal of Selection and Assessment **21**, 309–321 (9 2013)
6. del-Río-Ortega, A., Resinas, M., Cortés, A.R.: Defining process performance indicators: An ontological approach. In: OTM Conferences (1). pp. 555–572 (2010)
7. Ferreira, D.R., Alves, C.: Discovering user communities in large event logs. In: Business Process Management Workshops (1). pp. 123–134 (2011)
8. Gamma, E., Helm, R., Johnson, R.E., Vlissides, J.M.: Design patterns: Abstraction and reuse of object-oriented design (reprint). In: Software Pioneers, pp. 701–717. Springer Berlin Heidelberg (2002)
9. Lashkevich, K., Milani, F., Chapela-Campa, D., Suvorau, I., Dumas, M.: Why am I waiting? data-driven analysis of waiting times in business processes. In: CAiSE. pp. 174–190 (2023)
10. Leyer, M., ter Hofstede, A.H.M., Syed, R.: Detecting weasels at work: A theory-driven behavioural process mining approach. In: BPM (Forum). Lecture Notes in Business Information Processing, vol. 490, pp. 337–354. Springer (2023)
11. Martin, N., Depaire, B., Caris, A.: The use of process mining in business process simulation model construction - structuring the field. Bus. Inf. Syst. Eng. **58**(1), 73–87 (2016)
12. Meckling, W.H., Jensen, M.C.: Theory of the firm. Managerial Behavior, Agency Costs and Ownership Structure (1976)
13. Mustroph, H., Winter, K., Rinderle-Ma, S.: Social network mining from natural language text and event logs for compliance deviation detection. In: CoopIS. pp. 347–365 (2023)
14. Song, M., van der Aalst, W.M.P.: Towards comprehensive support for organizational mining. Decis. Support Syst. **46**(1), 300–317 (2008)
15. Song, M., van der Aalst, W.M.: Supporting process mining by showing events at a glance. In: Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS). pp. 139–145 (2007)