Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

A Scoring-Based Approach for Structuredness-Aware Business Process Classification

Kerstin Andree

kerstin.andree@tum.de

Technical University of Munich

Ivan Kuzmin

Technical University of Munich

Luise Pufahl

Technical University of Munich

Research Article

Keywords: Activity Relationships, Process Behavior, Structuredness, Discovery, Process Classification

Posted Date: July 24th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10398302/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

A Scoring-Based Approach for Structuredness-Aware Business Process Classification

Kerstin Andree[1*], Ivan Kuzmin[1] and Luise Pufahl[1]

[1*]School of Computation, Information and Technology, Technical University of Munich, Bildungscampus 2, Heilbronn, 74076, Germany.

*Corresponding author(s). E-mail(s): kerstin.andree@tum.de;
Contributing authors: ivan.kuzmin@tum.de; luise.pufahl@tum.de;

Abstract

Business processes vary in their degree of predictability and flexibility, ranging from highly standardized workflows to loosely structured, knowledge-intensive processes. Identifying the level of process structuredness is important for organizations, as it influences the selection of process modeling paradigms, and governance or process improvement initiatives. However, process mining does not offer automated support for assessing process structuredness from event logs. To address this gap, we propose a scoring-based approach for automatically classifying event logs into the structuredness classes structured, semi-structured, loosely structured, and unstructured. The approach uses a fine-grained activity relationships notation that differentiates between temporal and existential dependencies among activities. Based on the discovered relationship patterns from an event log, class-specific weighting is applied to identify the structuredness class that best represents the observed process behavior. We evaluate the approach using synthetic and real-world event logs representing different levels of process structuredness showing that the proposed method can effectively characterize process structuredness.

Keywords: Activity Relationships, Process Behavior, Structuredness, Discovery, Process Classification

1 Introduction

Business processes constitute the backbone of organizations, as they describe the coordinated set of activities through which companies create value, deliver services, and achieve strategic objectives [1]. By defining how work is performed, by whom, and in what sequence, *Business Process Management* (BPM) enables organizations to ensure efficiency, quality, compliance, and continuous improvement of their processes through concepts, methods, and techniques. Business processes can be characterized by their type of behavior. Following the process behavior spectrum [2], which primarily considers the structure and level of predictability of the underlying process, we distinguish between structured, semi-structured, loosely structured, and unstructured processes. Structured processes are highly predictable and follow standardized procedures with limited flexibility in execution, such as invoice processing. Semi-structured processes contain well-defined segments that can be flexibly combined depending on contextual factors, for example, insurance claim handling. Loosely structured processes are characterized by a high degree of flexibility and lower predictability, as often observed in patient treatment. Processes that do not follow a recurring behavioral pattern and are primarily knowledge-driven are considered unstructured, such as exploratory research activities.

For organizations, such a structuredness-aware characterization of their processes is important for two core reasons: (1) process representation and (2) process maturity assessment.

The *business process model* serves as a core representation artifact in BPM to abstract from the original process for a visual, human-oriented representation of its behavior. Process models are also used for further analysis in, for example, conformance checking [3], process redesign [4], and compliance checking [5], or for process execution [6, 7]. To create process models, different modeling approaches and languages exist that can be mapped to the respective levels of structuredness. In general, we differentiate between three model paradigms [2, 8, 9]: *declarative* languages are used for loosely structured processes, *imperative* languages for structured processes, and *hybrid* languages for semi-structured processes.

The level of process maturity reveals potential for standardization. Standardized processes reduce cost, ensure quality, and improve execution time, while flexibility and unpredictability are essential for innovation, knowledge work, and customer management [10, 11]. Balancing these process performance dimensions is essential for organizations, making process mining an essential capability to remain competitive [12]. Structured processes are considered standardized and compliance-driven [13], whereas highly flexible processes are considered innovative and knowledge-intensive. By understanding the level of structuredness of their processes, organizations can appropriately design, govern, and improve measures.

To analyze process behavior, process mining offers methods and techniques for investigating *event log* data, i.e., records of process execution data [3]. Process discovery, for example, creates a process model for a given event log by investigating behavioral patterns, visualized in one pre-defined modeling language [8]. However, existing process mining approaches do not provide automated methods to assess the

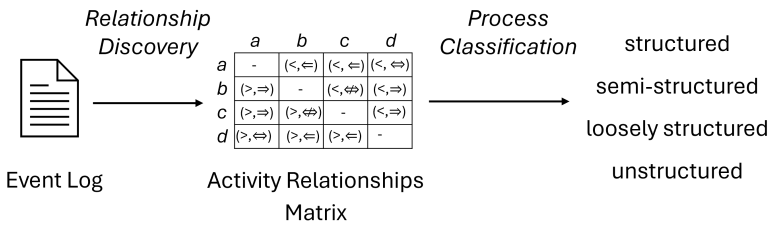


Fig. 1 Approach for automated process classification

structuredness of a business process based on event log data. A mapping from behavioral patterns to the level of structuredness of the underlying process is missing. Thus, selecting an appropriate process modeling paradigm for process model creation remains challenging, as process analysts can only manually draw conclusions about the process structuredness by reviewing the number of trace variants in the log. Consequently, we address the following research question in this paper:

RQ: How can we automatically classify the level of structuredness for a given process by observing behavioral patterns in an event log?

To address this research question, we propose a scoring-based approach to classify event logs according to the structuredness of the underlying process. We follow the spectrum of structuredness introduced in [2] and adopt the established classes, namely structured, semi-structured, loosely-structured, and unstructured processes. The proposed classification algorithm builds on the activity relationships notation introduced in [14] as an intermediate step to extract behavioral patterns from the event log. This notation provides a more fine-grained characterization of activity relationships by distinguishing between temporal and existential dependencies, thereby separating temporal ordering from occurrence dependencies [15]. Based on this distinction, we determine the level of structuredness by applying class-specific weights to relationship ratios discovered from an event log to check for the most suitable structuredness class. To extract these relationships from process execution data, we develop a relationship discovery algorithm that takes an event log as input and constructs an activity relationships matrix, providing a comprehensive overview of all activity relationships. An overview of the overall approach is shown in Fig. 1.

In summary, we contribute an approach to classify event logs by their level of structuredness. As part of this contribution, we also present an automated discovery algorithm to automatically extract activity relationships from event logs.

The remainder of this paper is structured as follows. Section 2 introduces background knowledge on structuredness and activity relationships. Related work is discussed in section 3. We present our research method in section 4 and explain the proposed approach across two sections: Section 5 presents the relationships discovery algorithm whereas section 6 introduces the classification heuristic. We evaluate our work in section 7 and section 8 discusses the results and concludes.

2 Background

2.1 Process Structuredness

Business processes can be classified according to their structure and predictability [2, 16, 17]. Both characteristics influence the level of automation and control as well as the process flexibility. Fig. 2 presents the adapted spectrum of process classification introduced in [2].

Highly structured processes are low in flexibility, predictable, and repeatable [18]. Inputs and outputs are clear, and the majority of process execution paths are similar. Exceptions are predictable and included in the model at design time [19]. A common modeling language that supports such processes is *Business Process Model and Notation* (BPMN). The industry standard is known for its support to represent only allowed behavior via imperative process models [9].

Semi-structured processes are classified as being unstructured but having pre-defined, structured segments [2]. The overall execution order is not defined, however, there exist previously known constraints for sets of activities. *Fragment-based Case Management* (fCM) is an approach that supports such processes. So-called fragments, i.e., structured segments, can be flexibly combined while preserving the pre-defined order of a few activities.

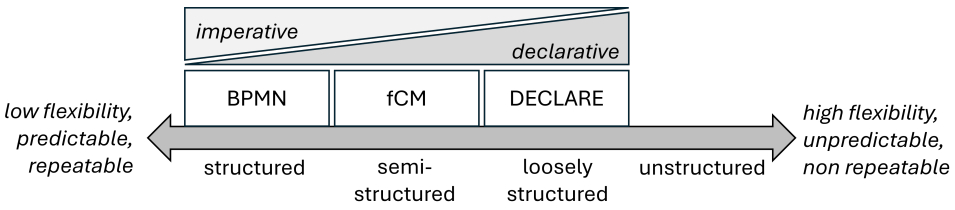


Fig. 2 Spectrum of process classification, adapted from [2]

Loosely structured processes are highly flexible and depend on individual decisions of knowledge workers during runtime. Respective models describe "the normal way of doing things" [17], but the real process executions can deviate. Process models using the *ConDec* language [20], for example, specify a set of constraints that must be followed [18]. These can include allowed, optional, and forbidden behavior [9]. Anything not specified is up to the knowledge worker executing the process.

Unstructured processes cannot be represented or defined in advance due to their non-repeatable characteristics [2]. Every process execution is individual and not predictable. Thus, modeling languages do not support such processes because there are no activity dependencies that can be defined.

Based on the spectrum of process classification and empirical results, we define four main classes and two sub-classes covering the intersections, respectively (cf. Fig. 3). Thus, we are able to provide a more fine granular classification of processes without introducing harsh thresholds. For the remainder of this paper, we are using the following notation to indicate the respective structuredness class:

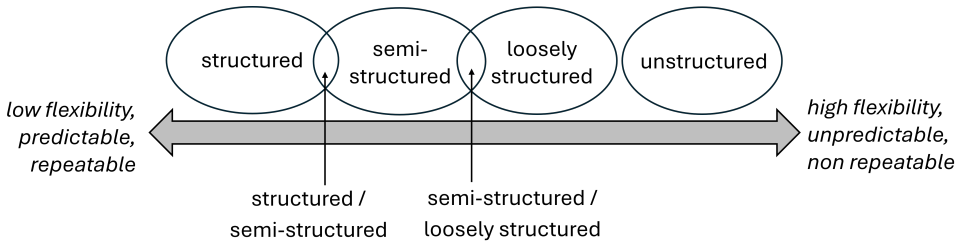


Fig. 3 Classes of structuredness used for automated process classification

- S: Structured process
- S/SS: Borderline case, the process contains structured and semi-structured characteristics
- SS: Semi-structured process
- SS/LS: Borderline case, the process contains semi-structured and loosely structured characteristics
- LS: Loosely structured process
- US: Unstructured process

2.2 Activity Relationships

Business processes consist of a set of activities that are executed in a coordinated manner [1] which is why *activity relationships* are essential for accurate process discovery. While DFGs and LTL formulae focus only on temporal dependencies, Andree et al. [14] introduce a notation for binary activity relationships defined as a tuple (d_{temp}, d_{exist}) consisting of two types of activity dependencies: *temporal* (d_{temp}) and *existential* dependency (d_{exist}). A temporal dependency specifies the ordering between two activities if both activities exist in a given trace. It is distinguished between *eventual* (activities can occur in between, $A \prec B$) and *direct* (directly-follows relationship, $A \prec_d B$) temporal dependency. In contrast, an existential dependency specifies occurrence dependencies. Given the occurrence, i.e., existence, of an activity A in a trace, the existential dependency reveals what we can infer for the occurrence of a second activity B. For example, an existential implication $A \Rightarrow B$ indicates that an occurrence of A implies the occurrence of B but not vice versa. Table 1 provides an overview of the types of existential dependency. For further information, please refer to [14].

If the process contains loops, certain activities are repeated multiple times. This behavior is captured by $(A, A) = (A_{first}, A_{second})$ for two instances of activity A. For multiple iterations of the loop, the relationship remains consistent with $(A_{second}, A_{third}), (A_{third}, A_{fourth})$, etc. The approach covers structured, non-nested loops, i.e., single-entry-single-exit loops.

The differentiation between temporal and existential dependencies enables a more fine-grained definition of activity relationships than is possible with traditional process models, for several reasons. Process modeling languages typically adhere to

Table 1 Types of existential dependencies for two activities A,B

Type	Explanation	Notation
Implication	The occurrence of A implies the occurrence of B	$A \Rightarrow B$
Co-Occurrence	Either both activities A,B occur or none of them	$A \Leftrightarrow B$
Non-Co-Occurrence	Exactly one out of the two activities A,B occurs	$A \not\Leftrightarrow B$
Inclusive OR	At least one of the activities A,B occurs	$A \vee B$
NAND	A and B cannot occur together	$A \bar{\wedge} B$
Independence	The occurrence of A does not depend on the occurrence of B and vice versa	$A - B$

either a declarative or an imperative paradigm, which introduces a bias when analyzing process structuredness. For example, Business Process Model and Notation (BPMN) emphasizes the control-flow perspective, implicitly assuming structured and predictable behavior, as all possible execution paths must be explicitly modeled. In contrast, the ConDec language, commonly used for Declare models [20], specifies constraints between activities without explicitly representing execution sequences. As a result, classifying process structuredness based on process models is challenging, not only because of paradigm-induced bias, but also due to the expertise required to interpret different modeling notations.

Although the activity relationships introduced in [14] can be expressed in both BPMN and Declare, they are not bound to a specific modeling paradigm. Furthermore, separating temporal from existential dependencies allows us to analyze these dimensions independently, which is particularly important for structuredness analysis. For instance, semi-structured processes differ from fully structured processes primarily in their temporal constraints: while predefined segments remain structurally well-defined, flexibility arises from when—and whether—these segments are executed. Such flexibility is captured by eventual temporal dependencies and, in the case of optional segments, by existential independence.

3 Related Work

This section introduces related work in the domain of process classification according to its level of structuredness.

When analyzing process characteristics, process analysts often use automated process discovery [3] to compare different process representations or manually check the number of process variants in an event log. Common tools, such as ProM [21], allow analysts to run different discovery algorithms to compare a declarative representation with an imperative one for the same event log. Log analysis tools, e.g.,PM4Py [22], support process analysts in investigating process variants and their frequencies. The literature review of Imran et al. [23] shows that flexible processes are often associated with increased process model complexity. This complexity can be measured along two

dimensions: behavioral (e.g., the number of subprocesses) and structural (e.g., control flow complexity) complexity. However, such analyses still require a manual mapping from gained insights to the structuredness classes due to a missing benchmarks and automated support.

Structuredness-aware classification based on event log data is also present in discovery algorithms for hybrid models [9]. The basic idea of these approaches is to identify structured and unstructured parts without classifying the event log as a whole. The Hybrid Miner [24] finds structured (modeled as a Petri net) and unstructured (modeled as a DECLARE model) parts of data by counting the predecessors and successors for each event contained in the event log. If an event exceeds a certain cutoff value, the event is considered to be part of the unstructured past. Otherwise, it is assigned to the structured part. A similar approach is followed by the Fusion Miner [25]. Based on the resulting model after applying the Heuristics Miner, the approach counts the connections that each activity has to others. An activity that has several connections is then considered to be part of the declarative part of the process model and modeled using declarative constraints rather than control-flow relationships. Although both approaches count the participation of one event in different relationships, they do not consider its type. Our approach is based on different types of activity dependencies extracted from an event log, so we can provide a more detailed analysis for classifying processes. The Precision Optimization Hybrid Miner [26] uses the Inductive Miner to identify unstructured process parts. The ability of the Inductive Miner to represent unstructured parts as so-called flower models for redo operators is leveraged. These unstructured parts are then replaced by DECLARE constraints.

In contrast to these approaches, our algorithm automatically identifies the level of structuredness according to its characteristics. Based on this classification, organizations can then identify appropriate process representations.

To assess process maturity with respect to standardization, reference models are commonly used [27]. Large vendors, such as SAP or Celonis, have access to process data from multiple organizations, which enables industry-wide benchmarking of individual processes. However, such benchmarking approaches largely depend on the availability of suitable reference models. Wurm et al. [13] propose a measurement instrument for process standardization spanning multiple dimensions, from control-flow characteristics to organizational culture and strategic alignment. Nevertheless, their approach relies on manual assessment and is therefore not suited for automated evaluation.

We provide an automated approach for assessing process maturity based on structuredness, without requiring reference models, thus offering a fast solution to get a first understanding of process characteristics.

4 Research Method

The idea of mapping a given activity relationship matrix to a specific level of structuredness is motivated by the characteristics of the underlying relationships notation. By decomposing each relationship into a temporal and an existential dependency component, we find that certain dependency types occur more frequently in some classes of process structuredness than in others.

Highly structured processes are typically linear and predictable, consisting mainly of sequential activity flows, some parallelism, and a few decision points. As a result, such processes usually exhibit direct temporal dependencies and co-occurrences, while implications and independencies, especially existential independence, occur rarely. Semi-structured processes, in contrast, comprise predefined structured segments that can be flexibly combined. This leads to an increased presence of temporal and existential independencies, while still maintaining a relatively high number of direct-follow relationships. Loosely structured processes further amplify this trend, showing a higher proportion of temporal and existential independencies as well as more frequent implications, reflecting increased flexibility in execution.

Based on our observations, activity relationships provide a strong indicator of the level of structuredness of a process. This insight motivates the derivation of process structuredness directly from the activity relationship matrix. So far, such an activity relationships matrix can only be constructed manually from BPMN diagrams, limiting applicability and scalability [14]. To enable a fully automated and generally applicable solution, it is necessary to first develop a discovery algorithm that automatically extracts activity relationships from event logs, and show its validity. In a second step, we can then use the discovered activity relationships to derive a heuristic that classifies event logs based on their structuredness.

As no existing event logs are labeled with levels of structuredness, a synthetic dataset was generated for development and testing purposes. Although limited in size, the dataset was designed to include representatives from all defined levels of structuredness. Consequently, we formulate the classification problem as a score-based approach that exploits the differing distributions of activity dependency types across structuredness classes. Based on the synthetic dataset, we derive class-specific weights for each relationship type and compute a weighted score over the observed relationship distribution of a process. An overview of the proposed method is shown in Fig. 4.

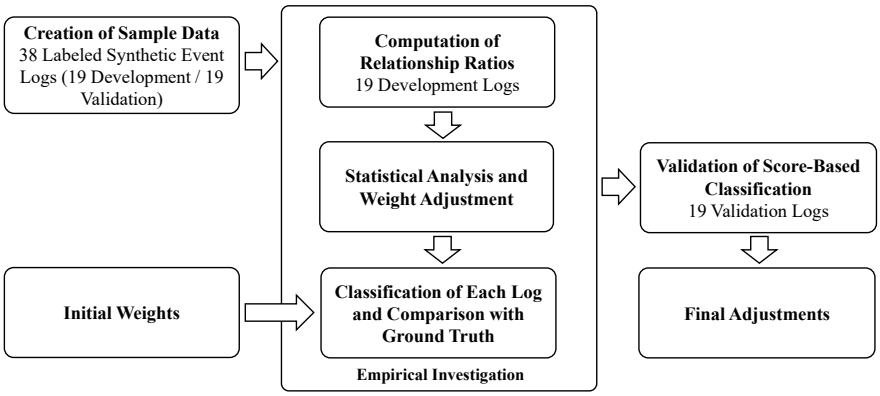


Fig. 4 Score-based classification approach development

We generated a total of 38 synthetic event logs from manually created process models, categorized as follows: 12 structured, 12 semi-structured, 12 loosely structured, and 2 unstructured. The two unstructured logs were deemed sufficient, given the extreme case of this category. Loosely structured logs were generated by manually creating Declare models, following the respective structuredness criteria, which were simulated using MINERful [28] simulation. Semi-structured logs were created by simulating manually created BPMN diagrams and Declare models using the PLG2 Log Generator [29] and MINERful, respectively. Structured logs were created by generating sets of variants following the control-flow patterns [30] that were converted into an XES format[1]. Similarly, unstructured logs were created by generating all permutations for a given set of activities. The dataset was divided into 19 event logs for development and 19 for validation and final adjustments. For each event log, we derived the activity relationships matrix using the discovery algorithm presented in section 5. An empirical investigation of these matrices revealed which ratios of activity relationships indicate a given level of structuredness. Based on these findings and initial class-specific weights derived from the general definitions (see section 2, adjustment was done by conducting a statistical analysis so that the classification output matches the ground truth. The scoring algorithm was then validated with the 19 remaining logs. Minor adjustments were made to refine the classification criteria.

5 Discovering Activity Relationships

For the purpose of structuredness classification, we analyze temporal and existential dependencies between activities in order to capture process behavior more precisely. The activity relationships notation introduced in [14] already distinguishes between three types of temporal and six types of existential dependencies. However, this notation exhibits the following limitations: (1) the category of eventual temporal dependency also subsumes direct succession, making it impossible to distinguish between true eventual dependencies and eventual dependencies that may also allow direct succession, and (2) activity relationships can currently only be derived from BPMN models, but not directly from event logs. Both limitations are critical for an effective and fine-grained classification of process structuredness, particularly for enabling a more accurate and fully automated assessment. This section, therefore, introduces extensions to the existing notation that address these limitations and serve as the foundation for the proposed structuredness classification approach. Subsection 5.1 presents the extensions to the activity relationships notation before subsection 5.2 details the corresponding discovery algorithm. Finally, we validate the algorithm in subsection 5.3.

5.1 Extending the Activity Relationships Notation

The extension of the activity relationships notation with a *true eventual temporal dependency* is motivated by the following observation. Consider Fig. 5, which depicts a BPMN model consisting of five activities, where exactly one activity from the set

[1]We used the following converter publicly available on GitHub: https://insm-tum.github.io/activity-relationship-matrix-discovery/

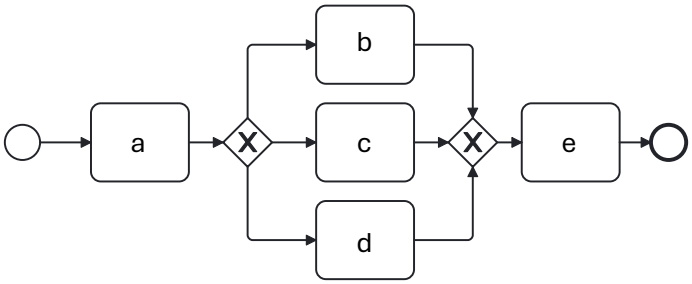


Fig. 5 BPMN diagram illustrating the need for a true eventual temporal dependency

$\{b, c, d\}$ must be selected in each process execution. When deriving activity relationships for this process using the existing notation [14], the following relations are obtained: $(a, b) = (a, c) = (a, d) = (\prec_d, \Leftarrow)$ and $(a, e) = (\prec, \Leftrightarrow)$. However, the relationships between the activity pairs (b, c), (b, d), and (c, d) cannot be represented unambiguously using the current set of relationship types.

Modeling these relations as non-co-occurrences is not consistent, as it would imply that either b or c must occur, while simultaneously enforcing mutual exclusivity between c and d as well as between b and d. Alternatively, a NAND relation could be used to express a choice among three activities; however, NAND relations also allow the case in which none of the three activities is executed, which contradicts the semantics of the model. In combination with the eventual temporal dependency between a and e, a direct succession of a and e is possible, although such behavior is not intended in the modeled process.

As a result, the derived set of activity relationships is not fully consistent with the BPMN model and the underlying process behavior. From a structuredness perspective, this distinction is also relevant, as the prevalence of NAND versus non-co-occurrence relations has different implications: non-co-occurrences indicate more structured behavior, as they enforce the execution of exactly one activity from a set. Similarly, a NAND relationship allows for too many process variants that indicate more flexible behavior, which should not be the case.

To accurately capture process behavior in cases such as the example shown in Fig. 5, we therefore introduce the *true eventual temporal dependency* ($\prec_t$). This dependency explicitly excludes direct succession and requires that at least one intermediate activity be executed between the two related activities. Moreover, we differentiate between *no activity ordering observed* (x), e.g., in case of exclusive behavior, and *both activity ordering directions observed* (-) as it is the case for activities that are executed in parallel. This extension results in the following relationships for the example given in Fig. 5: $(a, b) = (a, c) = (a, d) = (\prec_d, \Leftarrow)$, $(a, e) = (\prec_t, \Leftrightarrow)$ and $(b, c) = (b, d) = (c, d) = (\mathrm{x}, \overline{\wedge})$.

For relationships between multiple instances of the same activity, (a, a) denotes the relationship from the first instance of a to the second instance of a, which is also true for the relationship from the second to third instance of a, the third to fourth instance of a, and so on [14]. We further extend this notation to relationships between

multiple instances of different activities. For example, the relationship $(a, d) = (\prec, \Leftarrow)$ indicates that activity a precedes d in any process execution. If multiple instances of a occur, all of them occur before d. In contrast to temporal dependencies, an existential dependency is not defined for multiple instances of activities. $a \Leftarrow b$ means that if there exists at least one activity b, there also exists at least one activity a in the same case. We deliberately refrain from defining existential dependencies for multiple instances because it would lower the precision of the proposed discovery algorithm.

5.2 Relationships Discovery Algorithm

To determine the type of relationship for each pair of activities, we discover the temporal and existential dependency separately. Fig. 6 shows the general approach. For each pair of activities, retrieved from the event log in a preprocessing step, and the set of traces, we determine both dependencies (see Algorithms 1 and 2) before merging them into one relationship. Prior to dependency discovery, infrequent behavior is removed at the trace level using automatic variant-frequency filtering inspired by Conforti et al. [31]. Consequently, the dependency discovery stage itself no longer relies on tolerance thresholds when deciding temporal or existential relationships. Note that the algorithms present simplified pseudocode for clarity. For more details on the additional logic for handling edge cases, bidirectional relationships, and complex loop structures, we refer to the GitHub repository linked in this paper.

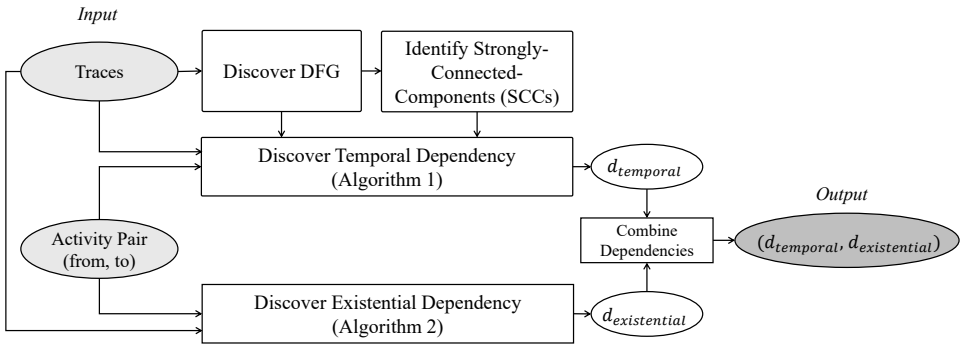


Fig. 6 General approach to discover activity dependencies

Algorithm 1 determines the temporal relationship between two activities (A, B) within a set of traces. It analyzes in which order they occur across all traces in the event log. The goal is to identify whether one activity consistently occurs before another, whether both orders are possible, whether there is no temporal ordering at all, or whether the activities are temporally independent.

The algorithm expects several input parameters that are generated in a preprocessing step. We generate a DFG based on the set of traces to differentiate between direct,

eventual, and true eventual orderings. Additionally, we run a strongly-connected-components (SCC) identification algorithm on the DFG to discover loop structures that support the discovery of temporal dependencies within loops. This information is contained in the input parameter *LoopContext*.

Based on its input, the algorithm extracts ordering information. For each pair of activities, it records whether the two occur together in the same trace and how often one precedes the other, and vice versa (line 1). Multiple instances within a single trace are taken into account. Then, the algorithm iterates over all activity pairs (line 2). If both activities are identical and the activity has a self-loop, the algorithm distinguishes two cases: if the activity has other outgoing DFG edges besides the self-loop, it can repeat after intervening activities and is classified as EVENTUAL; otherwise, the activity immediately repeats and is classified as DIRECT. If the activities never co-occur in the same trace, the algorithm returns NO_ORDERING (line 4), indicating that no temporal dependency can be observed between them.

For co-occurring activities, the algorithm checks whether each direction has a viable directly-follows edge in the DFG and whether one activity is observed before the other (lines 5–8). Viable direct edges are obtained after the preceding noise-filtering step and may additionally be constrained by edge-confidence gating. If both directions have viable DFG edges and both orderings are observed across traces, the algorithm checks whether each direction occurs as the *exclusive* ordering in at least one trace (line 10). If so, there is no fixed temporal ordering and the pair is classified as INDEPENDENCE (line 11). This captures behavior where both orderings are genuinely possible across different process executions, as opposed to loop-style repetition within a single trace (where neither direction would have exclusive traces).

If exactly one viable direct edge exists (lines 13–18), the algorithm distinguishes between DIRECT and EVENTUAL. A dependency is classified as DIRECT when the corresponding succession is deterministic and no alternative path exists. Otherwise, the dependency is classified as EVENTUAL. The same logic is applied symmetrically for the backward direction (lines 16–18).

If no viable direct edge exists, the algorithm relies solely on observed orderings in the traces. If both orderings are observed (line 19), the pair is classified as INDEPENDENCE. If only one direction is observed (lines 20–23), the algorithm distinguishes between TRUE_EVENTUAL and EVENTUAL using SCC information: activities inside the same SCC are assigned the weaker EVENTUAL relation, whereas activities in different SCCs are assigned TRUE_EVENTUAL. The same logic is applied symmetrically for backward dependencies.

Algorithm 2 determines the type of existential dependency between two activities (A, B) within a set of traces. First, it handles the special case where both activities are identical (line 1). In this case, the algorithm returns an IMPLICATION dependency if activity A occurs multiple times within at least one trace, indicating that the occurrence of one instance of A implies the occurrence of another instance of A. For distinct activities, the algorithm iterates over all traces and classifies each trace into one of four categories: traces containing both A and B, only A, only B, or neither activity. As described in Section 5.1, we omit multiple instances here. The existential

dependency analysis is done based on a set notion. For each trace, we generate the set of contained activities and perform the checks in lines 6–10.

These derived dependency counts are then evaluated directly, without any tolerance parameter. Infrequent behavior has already been removed before dependency discovery by the automatic trace-level filter. Based on the resulting co-occurrence patterns, the algorithm identifies the existential dependency type between A and B. If none of the specific patterns applies, the algorithm returns the default type INDEPENDENCE. The final **else** branch (line 28) is a safety net and is not reachable for well-formed input, as the preceding conditions collectively cover all possible count combinations.

Noise Handling

Noise filtering is performed by automatically deriving a minimum variant frequency threshold from the frequency distribution of variants observed in the event log, combining two separately computed estimators.

The first, a gap-based estimator, sorts all variant counts in ascending order and searches the lower half of its distribution (concentration of infrequent, noise-prone variants) for the largest adjacent ratio between consecutive counts. If a ratio of at least $3\times$ is found between two neighbouring counts, the threshold candidate is set to the geometric mean of the two values, normalized by the total number of traces in the log. This defines a natural "cliff" separating low-frequency noise from higher-frequency signaling variants. If no such gap exists, the estimator returns zero.

The second, a floor-based estimator, is computed as a size-aware absolute lower bound, normalized by the number of total traces:

$$\text{floor} = \frac{\max(2,\ \min(12,\ 0.5 \times \text{varcount}))}{|\text{traces}|},$$

with varcount indicating the second largest variant count. This bounds the minimum retained frequency to at least two traces and at most 12, while ensuring that no more than half of the second-most-frequent variant's occurrences can be discarded.

If a qualifying gap is detected in the lower half of the variant frequency distribution, the resulting cliff-based threshold is used, as it reflects distributional evidence specific to the given event log. Only when no such gap exists does the algorithm fall back on the size-aware absolute floor. Since structuredness classification relies on the observed distribution of activity relationships to distinguish, for example, legitimate optional or exceptional behavior from noise, a distribution-derived threshold better preserves rare-but-legitimate variants than a fixed, data-agnostic floor would. The floor is retained only as a fallback for cases where no clear separation between noise and signal can be identified from the data itself, ensuring that filtering remains grounded in the log's actual behavior whenever such evidence is available.

Algorithm 1 DiscoverTemporalDependency

Input: Traces, DFG, LoopContext, confidence_weak
Output: $(A, B) \to$ TemporalDependency
Functions:

EdgeConfident(X, Y) confidence$(X \to Y) \geq$ confidence_weak.
HasSelfLoop(X) X appears more than once in some trace.
HasOtherEdges(X) X has outgoing DFG edges besides its self-loop.
ObservedBefore(X, Y) True if X occurs before Y in any trace.
ExclusiveBefore(X, Y) Traces where X occurs before Y *and* Y does *not* occur before X.
HasMultiplePaths(X, Y) Y reachable from X via a path that does not use the direct edge $X \to Y$.
IsDeterministic(X, Y) X has exactly one outgoing DFG successor in traces where both X and Y occur.
SCC(X) SCC index of X from LoopContext; same SCC $\implies$ possible loop.

Build *cooccurrence*, *orderings*, *exclusive_counts* from Traces
for $(A, B) \in$ AllPairs **do**
  if $A = B \wedge$ HasSelfLoop(A) **then**
    if HasOtherEdges(A) **then return** EVENTUAL
    elsereturn DIRECT
    end if
  else if $\neg$CoOccur(A, B) **then return** NO_ORDERING
  end if
  $e_{AB} \leftarrow$ EdgeConfident(A, B)
  $e_{BA} \leftarrow$ EdgeConfident(B, A)
  $o_{AB} \leftarrow$ ObservedBefore(A, B)
  $o_{BA} \leftarrow$ ObservedBefore(B, A)
  $x_{AB} \leftarrow$ ExclusiveBefore$(A, B) > 0$
  $x_{BA} \leftarrow$ ExclusiveBefore$(B, A) > 0$
  if $e_{AB} \wedge e_{BA} \wedge o_{AB} \wedge o_{BA} \wedge x_{AB} \wedge x_{BA}$ **then return** INDEPENDENCE ▷ Each direction exclusive in some trace
  end if
  if e_{AB} **then**
    if IsDeterministic$(A, B) \wedge \neg$HasMultiplePaths(A, B) **then return** DIRECT
    elsereturn EVENTUAL
    end if
  end if
  if e_{BA} **then**
    if IsDeterministic$(B, A) \wedge \neg$HasMultiplePaths(B, A) **then return** DIRECT_BACKWARD
    elsereturn EVENTUAL_BACKWARD
    end if
  end if
  if $o_{AB} \wedge o_{BA}$ **then return** INDEPENDENCE
  else if o_{AB} **then return** EVENTUAL if SCC(A) = SCC(B), else TRUE_EVENTUAL
  else if o_{BA} **then return** EVENTUAL_BACKWARD if SCC(A) = SCC(B), else TRUE_EVENTUAL_BACKWARD
  elsereturn NO_ORDERING
  end if
end for

Algorithm 2 DiscoverExistentialDependency

Input: Traces, ActivityPair (A, B)
Output: Existential dependency type

if $A = B$ **then**
    return IMPLICATION if A repeats in any trace, else skip
end if
$c_{both} \leftarrow 0,\ c_{only_A} \leftarrow 0,\ c_{only_B} \leftarrow 0,\ c_{neither} \leftarrow 0$
for $trace \in Traces$ **do**
    $has_A \leftarrow (A \in trace),\ has_B \leftarrow (B \in trace)$
    if $has_A \wedge has_B$ **then** $c_{both} \leftarrow c_{both} + 1$
    else if $has_A \wedge \neg has_B$ **then** $c_{only_A} \leftarrow c_{only_A} + 1$
    else if $\neg has_A \wedge has_B$ **then** $c_{only_B} \leftarrow c_{only_B} + 1$
    else$c_{neither} \leftarrow c_{neither} + 1$
    end if
end for
if $c_{only_A} = 0 \wedge c_{only_B} = 0 \wedge c_{both} > 0$ **then return** EQUIVALENCE
else if $c_{only_A} = 0 \wedge c_{both} > 0$ **then return** IMPLICATION
else if $c_{only_B} = 0 \wedge c_{both} > 0$ **then return** IMPLICATION_BACKWARD
else if $c_{both} = 0 \wedge c_{neither} = 0 \wedge c_{only_A} > 0 \wedge c_{only_B} > 0$ **then return** NEGATED_EQUIVALENCE
else if $c_{both} = 0$ **then return** NAND
else if $c_{both} > 0 \wedge c_{only_A} > 0 \wedge c_{only_B} > 0 \wedge c_{neither} > 0$ **then return** INDEPENDENCE
else if $c_{both} > 0 \wedge (c_{only_A} > 0 \vee c_{only_B} > 0)$ **then return** OR
elsereturn INDEPENDENCE
end if

5.3 Validation

We use synthetic data to validate the proposed algorithm's ability to discover activity relationships from a given event log. For event log generation, we use the PLG2 Log Generator [29] to create XES event logs from 11 manually designed BPMN diagrams. Each BPMN diagram captures distinct process characteristics, including nested exclusive and parallel gateways, loops, and inclusive behavior, thereby covering a broad range of process behaviors. The BPMN models are publicly available on GitHub[2]. Our validation focuses on two aspects: (1) whether common behavioral patterns can be discovered correctly, and (2) the robustness of the approach in the presence of noise. To this end, we generated two event logs for each BPMN model: one noise-free log and one noisy log. Each log contains 1,000 traces. For noise insertion, we used the tool's default control-flow-only noise configuration. This configuration introduces several types of noise, including missing heads and tails, missing episodes (i.e., omitted activities), and attribute noise that results in ordering-related deviations. Each noise parameter was set to 5, indicating a probability of 5% of, e.g., a missing tail. Tail, head, and episode size were set to 2 meaning that maximum 2 events are missing in noisy traces. The complete dataset is available in our GitHub repository.

The results are shown in Table 2. For each log, we specify the types of process behavior, including nested structures, block structures, and skipping activities (indicated by checkmarks). Results are provided as a tuple showing percentages for each dependency type. (1.0, 1.0) means that 100% of the temporal dependencies and 100% of the existential dependencies were correctly discovered compared to the ground truth.

Table 2 Results of relationship discovery based on synthetic logs (✓: behavior covered by log). Values show the percentage of correctly identified temporal and existential dependencies compared to ground truth.

Log	Sequence	Parallelism	XOR	NAND	Multi-Choice	Looping	Nesting	Block-Structure	Skipping	Excl. Noise	Incl. Noise
L01	✓		✓					✓		(1.00, 1.00)	(1.00, 1.00)
L02		✓	✓							(1.00, 1.00)	(1.00, 1.00)
L03	✓		✓					✓		(1.00, 1.00)	(1.00, 1.00)
L04		✓	✓				✓	✓		(1.00, 1.00)	(1.00, 1.00)
L05		✓	✓				✓	✓		(1.00, 1.00)	(1.00, 1.00)
L06		✓			✓					(1.00, 1.00)	(1.00, 1.00)
L07										(1.00, 1.00)	(1.00, 1.00)
L08		✓				✓			✓	(1.00, 1.00)	(1.00, 1.00)
L09		✓	✓							(1.00, 1.00)	(1.00, 1.00)
L10	✓		✓				✓	✓		(1.00, 1.00)	(1.00, 1.00)
L11				✓						(1.00, 1.00)	(1.00, 1.00)

[2]Note that the BPMN diagrams were created based on the behavior supported by the PLG2 log generator. The same behavior can, in fact, also be modeled less complex.

The results for event logs without noise demonstrate that all tested process behavior patterns are correctly detected with 100% accuracy for both temporal and existential dependencies (Table 2). This comprehensive coverage includes sequential patterns (L01, L03, L10), parallelism (L02, L04, L05, L06, L08, L09), exclusive choice/XOR (L01-L05, L09-L10), NAND patterns (L11), single-entry-single-exit (SESE) looping behavior (L08), multi-choice/inclusive OR (L06), and nested structures (L04, L05, L10). Both block-structured processes and non-block-structured processes (L07) are correctly discovered, demonstrating the algorithm's versatility across different process modeling paradigms.

For the event logs containing noise, the automatic trace-level filtering step restores the clean-log discovery result for all eleven cases. The injected noise primarily creates low-frequency trace variants by inserting, deleting, or shifting events. Once these low-frequency variants are removed automatically before dependency extraction, the remaining traces recover the original process behavior closely enough for both temporal and existential dependencies to match the noise-free ground truth exactly. Consequently, the noisy logs achieve 100% accuracy for both dependency dimensions in Table 2, demonstrating that the combined approach of trace-level filtering followed by deterministic dependency classification is robust under the considered noise model.

6 Structuredness-Aware Process Classification

Having introduced the algorithm for discovering activity relationship matrices from event logs, we now turn to their use for structuredness-aware process classification. The central idea is that the relationship patterns captured in these matrices provide evidence about the degree to which a process follows predefined, repeatable behavior or allows for more flexible and variable execution. In this section, we therefore first analyze the empirical connection between activity relationships and process structuredness before explaining the scoring-based approach. Finally, a sensitivity analysis discusses how the classification output responds to perturbation of class-specific weights.

6.1 Activity Relationships and Structuredness

Table 3 presents the percentages of activity relationship types relative to all extracted relationships from 19 synthetic event logs used for development. The logs are ordered by their level of structuredness.

Processes characterized by pure independence, i.e., activities neither follow a defined sequence nor depend on each other's existence $(-, -)$, are considered unstructured. This aligns with the definition of unstructured processes and is reflected in the results shown in Table 3.

Relationships of type $(\prec, \Leftrightarrow)$ occur more often in structured and semi-structured processes. An existential co-occurrence indicates that either both activities occur in an instance or neither of them. This relatively strong existential relationship occurs in structured processes since a general execution path containing the core activities is usually pre-defined. Similarly, if pre-defined fragments in semi-structured processes are executed, all contained activities must occur. In contrast, loosely structured processes contain less restrictions, and thus, less existential co-occurences.

Table 3 Percentage of activity relationship types relative to all extracted relationships (relationships marked with * represent the set of relationships containing the same type but different directions)

	Structured						**Semi-Structured**						**Loosely Structured**						**Unstructured**
Relationship	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**SS1**	**SS2**	**SS3**	**SS4**	**SS5**	**SS6**	**LS1**	**LS2**	**LS3**	**LS4**	**LS5**	**LS6**	**U1**
$(-, -)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	56.06%	20.00%	8.33%	15.56%	0.00%	8.33%	80.00%	46.67%	0.00%	0.00%	23.81%	13.33%	**100.00%**
$(-, \Rightarrow)^*$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	16.67%	0.00%	8.89%	2.78%	**10.00%**	**26.67%**	**33.33%**	**40.00%**	**19.05%**	**26.67%**	0.00%
$(-, \Leftrightarrow)^*$	0.00%	10.00%	0.00%	0.00%	0.00%	0.00%	1.52%	0.00%	2.78%	2.22%	0.00%	0.00%	0.00%	**13.33%**	0.00%	**40.00%**	0.00%	**6.67%**	0.00%
$(-, \nLeftrightarrow)$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(-, \vee)$	0.00%	0.00%	0.00%	0.00%	0.00%	**6.67%**	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(-, \bar{\wedge})$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(\prec_d, -)^*$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(\prec_d, \Rightarrow)^*$	0.00%	0.00%	**40.00%**	**21.43%**	**16.67%**	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	2.38%	0.00%	0.00%
$(\prec_d, \Leftrightarrow)^*$	**40.00%**	**10.00%**	**10.00%**	**10.71%**	**33.33%**	**13.33%**	0.76%	4.55%	4.17%	0.00%	6.67%	6.94%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
$(\prec, -)^*$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	**22.73%**	**41.82%**	**11.11%**	**53.33%**	0.00%	0.00%	10.00%	0.00%	0.00%	0.00%	14.29%	6.67%	0.00%
$(\prec, \Rightarrow)^*$	0.00%	0.00%	20.00%	21.43%	33.33%	53.33%	**15.15%**	**32.73%**	**44.44%**	**4.44%**	**44.44%**	**61.11%**	0.00%	13.33%	33.33%	0.00%	30.95%	40.00%	0.00%
$(\prec, \Leftrightarrow)^*$	**60.00%**	**80.00%**	**20.00%**	**14.29%**	**16.67%**	**26.67%**	2.27%	0.91%	12.50%	20.00%	31.11%	20.83%	0.00%	0.00%	16.67%	20.00%	0.00%	6.67%	0.00%
$(\prec, \vee)^*$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%

Furthermore, the presence of relationships characterized by temporal independence and existential implication, i.e., $(-, \Rightarrow)$ or $(-, \Leftarrow)$, typically indicates a less structured process. These relationships often reflect decision points where an activity may be skipped. In particular, when looking at the percentage of this relationship type relative to the amount of observed relationships of type $(-, \Rightarrow)$ or $(-, \Leftarrow)$, i.e., all relationships indicating temporal independence and any existential relationship, we can see that a higher percentage correlates with loosely structured processes. Conversely, a lower ratio of these relationships may point to either structured or semi- structured processes, depending on the overall distribution of relationship types.

For semi-structured and loosely structured processes, we observe less directly-follows relationships. This indicates more flexible behavior in terms of temporal orderings.

6.2 Classification Algorithm

The classification algorithm gets an activity relationships matrix as an input and assigns it to one of the four structuredness classes defined in Section 2.1. For each class, the algorithm computes a weighted score based on the observed distribution of relationship types across all activity pairs. The unstructured class is treated separately because it constitutes a theoretical edge case: by definition, it contains only independence relations and no other constraints. Consequently, classification as unstructured is determined through a pre-filter check rather than the score-based classification procedure. Before describing the overall scoring procedure, we first introduce the class-specific weights and their interpretation, as they form the basis of the subsequent score computation.

Ratio Vector

Let $\mathcal{P}$ be the set of all ordered activity pairs (a_i, a_j) with $i \neq j$. Let $\mathcal{T} = \{t_1, \ldots, t_8\}$ and $\mathcal{E} = \{e_1, \ldots, e_7\}$ denote the sets of temporal and existential dependency types, respectively, so that together they comprise the 15 individual dependency types. For each individual type $k \in \mathcal{T} \cup \mathcal{E}$, we define its ratio as the fraction of pairs assigned that type:

$$r_k = \frac{|\{(a_i, a_j) \in \mathcal{P} : \text{type}(a_i, a_j) = k\}|}{|\mathcal{P}|} \tag{1}$$

We write r_t and r_e for r_k specialized to a temporal type $k = t \in \mathcal{T}$ or an existential type $k = e \in \mathcal{E}$, respectively. The ratio vector $\mathbf{r}^{\text{marg}} = (r_1, \ldots, r_{15})^\top$ henceforth referred to as *marginal* ratios, covers eight temporal and seven existential dependency types.

In addition, for each *joint* combination (t, e) of a temporal dependency t and an existential dependency e, we define the joint ratio as the fraction of pairs exhibiting that specific co-occurrence:

$$r_{t|e} = \frac{|\{(a_i, a_j) \in \mathcal{P} : \text{temp}(a_i, a_j) = t \ \wedge \ \text{exist}(a_i, a_j) = e\}|}{|\mathcal{P}|} \tag{2}$$

Joint ratios capture the correlation between temporal and existential dimensions, information that marginal ratios discard. Based on the empirical relationship distributions, the following joint combinations are used for classification (cf. Table 3):

- Temporal independence with each existential type: $(-, -)$, $(-, \Rightarrow)^*$, $(-, \Leftrightarrow)$, $(-, \not\Leftrightarrow)$, $(-, \vee)$, $(-, \overline{\wedge})$.
- Direct follows with implication and equivalence: $(\prec_d, \Rightarrow)^*$, $(\prec_d, \Leftrightarrow)^*$.
- Eventual and true eventual with independence, implication, equivalence, and or: $(\prec, -)^*$, $(\prec, \Rightarrow)^*$, $(\prec, \Leftrightarrow)^*$, $(\prec, \vee)^*$.

Combinations marked with $*$ include both directional variants (e.g., forward and backward).

Scoring with Dominance-Based Selection

For each structuredness class $c \in \{S, SS, LS\}$, the score aggregates contributions from all 15 marginal slots, where each slot's contribution is computed either from its marginal weight or from a dedicated joint weight, depending on whether a single joint combination *dominates* that slot.

For a given temporal type $t \in \mathcal{T}$ with marginal ratio r_t, let $\mathcal{E}_t \subseteq \mathcal{E}$ be the set of existential types that form relevant joint relationships with t (enumerated above). If there exists an $e \in \mathcal{E}_t$ such that

$$\frac{r_{t|e}}{r_t} \geq \theta, \tag{3}$$

with dominance threshold $\theta = 0.35$, then the contribution of temporal type t is scored from the joint ratio and a dedicated joint weight:

$$\text{contribution}_c(t) = w^{\text{joint}}_{c,(t|e)} \cdot r_{t|e}. \tag{4}$$

Otherwise the marginal weight applies:

$$\text{contribution}_c(t) = w^{\text{marg}}_{c,t} \cdot r_t. \tag{5}$$

The same check is applied symmetrically to each existential type e, testing whether some temporal type t dominates e via $r_{t|e}/r_e \geq \theta$. The total score for class c is then the sum over all 15 slots:

$$\text{score}(c) = \sum_{t \in \mathcal{T}} \text{contribution}_c(t) \; + \; \sum_{e \in \mathcal{E}} \text{contribution}_c(e). \tag{6}$$

The dominance threshold $\theta = 0.35$ was determined empirically: it is low enough to capture dominant joint patterns in structured and semi-structured logs, yet high enough to avoid triggering on incidental co-occurrences in loosely structured logs. Table 5 lists the marginal weights; the joint weights follow the same evidential scheme (Table 4) and are derived from the class definitions together with the empirical distributions shown in Table 3.

Table 4 Meaning of weight values

Value	Meaning
+3.0	Constitutive - feature defines the class
+2.5	Strong secondary - definitionally consistent, not alone sufficient
+2.0	Moderate - common in class, present in adjacent classes
+1.5	Supporting - consistent but non-exclusive
+1.0	Weak supporting
+0.5	Marginal - barely discriminative
0.0	Neutral - either supports nor contradicts the class
-0.5	Marginal contradiction — weak evidence against the class
-1.0	Contradictory — generally inconsistent with the class
-1.5	Moderately contradictory — characteristic of competing classes
-2.0	Strong contradiction — rarely observed in the class
-2.5	Near-exclusionary — strongly indicative of alternative classes
-3.0	Exclusionary — feature fundamentally contradicts the class definition

Table 5 Classification Weights per Relationship Type and Structuredness Class

Domain	**Relationship type**	**S**	**SS**	**LS**
Temporal	Direct ($\prec_d$)	+3.0	+1.5	+1.0
	Direct backward ($\prec_d^{\leftarrow}$)	+2.0	+1.0	+0.5
	No ordering ($\times$)	+1.5	+0.5	0
	Independence ($-$)	+0.5	+2.0	+2.5
	True eventual ($\prec_t$)	0	+2.0	+1.0
	True eventual backward ($\prec_t^{\leftarrow}$)	+2.0	+2.0	+1.0
	Eventual ($\prec$)	+1.5	+2.5	+1.5
	Eventual backward ($\prec^{\leftarrow}$)	+1.5	+2.0	+1.5
Existential	Equivalence ($\Leftrightarrow$)	+2.0	+1.0	+1.0
	Negated equivalence ($\not\Leftrightarrow$)	+2.5	+1.5	+1.0
	NAND ($\bar{\wedge}$)	+1.5	+0.5	+1.0
	Implication ($\Rightarrow$)	+1.5	+1.5	+2.0
	Implication backward ($\Leftarrow$)	+1.5	+1.5	+2.0
	OR ($\vee$)	+1.0	+1.5	+1.5
	Independence ($-$)	−2.0	+1.5	+2.5

Classification Procedure

For a given activity relationships matrix, classification proceeds in three ordered steps.

(1) Structural pre-filters. Before scoring, three structurally unambiguous signatures are checked in sequence. If certain known patterns are detected beforehand by defined pre-filters, they are handled directly without applying the linear scoring function, because the scoring model consistently produces misleading results for these cases.

Unstructured pre-filter: A process is classified as unstructured when all ordered activity pairs share the exact same temporal type and the exact same existential type, and no pair exhibits a direct-follows relationship:

$$\max_{t\in\mathcal{T}} r_t^{\text{temp}} = 1 \;\wedge\; \max_{e\in\mathcal{E}} r_e^{\text{exist}} = 1 \;\wedge\; r_{\prec_d}^{\text{temp}} = 0 \;\Rightarrow\; U \tag{7}$$

where $\mathcal{T}$ is the set of temporal types and $\mathcal{E}$ the set of existential types. The $r^{\text{temp}}_{\prec_d} = 0$ condition excludes purely sequential structured processes (where every pair is direct-follows + equivalence), which would otherwise match the uniform pattern but are definitionally structured.

(2) Weighted scoring. If no pre-filter fires, the process is classified by the argument of the maximum (argmax) over $\{S, SS, LS\}$, meaning that the class with the highest score is returned as output. Note that class unstructured (US) is handled exclusively by its pre-filter:

$$\hat{c} = \underset{c \in \{S,\, SS,\, LS\}}{\arg\max} \; \text{score}(c) \tag{8}$$

(3) Borderline detection. When the margin between the top-two scoring classes falls below a threshold δ, the result is flagged as borderline to indicate an overlapping classification:

$$\text{score}(\hat{c}_1) - \text{score}(\hat{c}_2) < \delta \;\Rightarrow\; \text{output } \hat{c}_1 \,|\, \hat{c}_2 \tag{9}$$

where $\hat{c}_1$ and $\hat{c}_2$ are the first and second ranked classes. We use $\delta = 0.2$ throughout our evaluation.

6.3 Weight Sensitivity Analysis

To assess the robustness of the classification heuristic with respect to its weight parameterization, we conducted a sensitivity analysis examining how the classification outcome responds to perturbations of the class-specific weights defined in Table 5. For each of the 38 scored event logs, we computed the score margin, defined as the difference between the top-ranked and second-ranked class scores (cf. equation 9), and systematically evaluated whether a uniform weight shift of ± 0.5 across any single weight would alter the classification outcome.

Table 6 Score margin distribution across development and testing dataset (38 logs).

Margin range	Logs	%	Interpretation
≥ 2.0	2	6%	Very robust – large separation
1.0–2.0	15	40%	Robust – clear separation
0.5–1.0	8	21%	Moderate – weight shift can flip
0.2–0.5	5	14%	Narrow – small perturbation flips
< 0.2	7	19%	Borderline – near tie

Table 6 summarizes the margin distribution across the full set of development and testing logs (38 scored event logs). The results show that 25 out of 38 logs (66%) achieve a margin of at least 0.5 and higher, making their classification immune to any single 0.5 weight perturbation. None of these robust logs changes its predicted class under any such perturbation. The remaining 34% of logs fall within the boundary region (margin < 0.5), with 7 logs (19%) exhibiting margins below the borderline

threshold $\delta = 0.2$ already applied during classification. These logs are therefore already flagged as borderline cases by the algorithm via equation 9 and are not presented as definitive classifications. Examining the boundary cases qualitatively reveals that they correspond to logs whose process behavior inherently spans two structuredness classes, e.g., semi-structured logs with dominant loosely structured features, or structured logs exhibiting parallel path patterns. This suggests that the small margins reflect genuine ambiguity at class boundaries in the underlying process data rather than a systematic weakness of the weighting scheme. This observation is consistent with the spectrum-based conceptualization of structuredness adopted in this paper, in which classes are defined as overlapping rather than sharply separated.

Overall, the sensitivity analysis provides evidence that the proposed classification approach is robust for the majority of event logs, and that boundary cases are both identifiable through the algorithm's built-in borderline detection mechanism and explainable by the structural characteristics of the underlying processes.

7 Evaluation

We conducted an evaluation with two main objectives: (1) the effectiveness of our proposed algorithm, and (2) the need for an automated classification approach. To this end, we conducted a survey on manually classifying processes according to their level of structuredness. Participants were asked to classify eight directly-follows graphs (DFGs). The DFGs were generated using the process mining tool Disco[3]. We used five real-world event logs taken from literature [32, 33] and the collection of resources for process mining on blockchain data[4] [34], and three manually created synthetic ones. DFGs were used instead of raw event logs or process models like BPMN or DECLARE to avoid bias. For instance, BPMN, being imperative, might lead participants to classify diagrams as structured regardless of their actual nature. All DFGs used for the survey are available in our GitHub repository. Ground truth labels were established using three complementary sources. For synthetic logs, the structuredness class was determined directly from the process model used for generation. For real-world logs, labels were derived from existing literature whenever an explicit characterization of process flexibility or structuredness was available. In cases where literature did not provide a direct classification, the final label was established through expert assessment and discussion among the authors. This multi-source strategy reduces reliance on a single subjective interpretation.

To ensure response quality, we included introductory questions about participants' ability to interpret DFGs and their background in process classification, allowing us to distinguish expert input from general users. Moreover, we asked the participants to rate the difficulty of each classification task.

The survey allows us to gather expert-labeled data that serves as a benchmark for evaluating our algorithm's effectiveness. By comparing the algorithmic classifications with those provided by domain experts, we can assess how closely it aligns with

[3] https://fluxicon.com/disco/
[4] https://ingo-weber.github.io/dapp-data/index.html

human judgment. Furthermore, participants' difficulty ratings help us better understand the cognitive complexity involved in manual classification, highlighting the value of automation in this context.

In the following, we will discuss the results of the survey with regard to the two objectives. In total, 13 participants completed the survey.

7.1 Effectiveness of Proposed Classification Algorithm

10 out of 13 participants declared explicit expert knowledge, ranging from researching process characteristics themselves or working with different modeling approaches. These answers were taken into account when comparing the classification results output by the proposed algorithm with expert assessment.

An overview of the results is shown in Fig. 7. The stacked bar chart summarizes the classification results for each process included in the survey. The result of the classification algorithm is highlighted with a bold box for each process to facilitate better comparison. Dotted lines indicate that the discovered structuredness class only partially matches the ground truth result.

Process P_1 was classified as semi-structured by our algorithm. Looking at the survey results, only 10% of survey participants classified the process as structured, while 80% identified semi-structured or borderline characteristics. The corresponding DFG shows fragments that can be executed at different points in time during execution. A clear execution path is not given. The analysis of Diba et al. [33] confirms this observation. The authors describe the process as semi-structured, identifying subprocesses that can be executed in parallel. Thus, the algorithm's classification aligns with the expert assessment and ground truth derived from literature.

The manually created process P_2 contains semi-structured behavior indicated by small fragments that can occur at multiple positions within a trace. 60% of the survey participants classified it as structured. Notably, 40% identified the semi-structured characteristic of the process as part of a borderline case or semi-structured class, aligning with the algorithm's output.

Regarding process P_3, 80% of the participants identified structured behavior. This is consistent with the assessment by Diba et al. [33] who describe the process as highly structured despite the number of variants. Our classification outputs structured behavior and, thus, aligns with the survey results and literature-based classification.

Process P_4 has structured characteristics as described by Hobeck et al. [35]. The majority of cases follow the same structured execution order. A closer look at the DFG also reveals that fragments can be identified, however, these fragments cannot be combined flexibly. 70% of the participants agree and identified structured behavior. Our algorithm also assigns the log structured characteristics.

Process P_5 is classified as semi-structured by our algorithm. 30% of the participants manually classifying the process identified semi-structured behavior together with loosely structured characteristics. Since Cremerius et al. [32] describe the process as loosely structured. We explain this observation with the fact that the classification approach looks for dominant patterns. For loosely structured patterns, a high percentage of independence is expected. However, if an existential independence is distributed

across multiple temporal types rather than concentrated in a single dominant combination, we are not using the joint weights, thus shifting the classification towards semi-structured.

The event log for process P_6 was created manually and has loosely structured characteristics. 40% of the participants correctly identify the level of structuredness. The proposed algorithm classified this process as loosely structured as well.

For process P_7, our algorithm classifies the log as loosely structured. The underlying event log originates from the incremental game ChickenHunt, where players follow diverse strategies when selecting activities, indicating a high degree of flexibility. This is reflected in the survey results, where 60% of participants identified loosely structured characteristics and none classified the process as structured.

Process P_8 was classified as unstructured by 80% of the participants. Our algorithm produced the same classification result, demonstrating the highest level of agreement with the expert assessment.

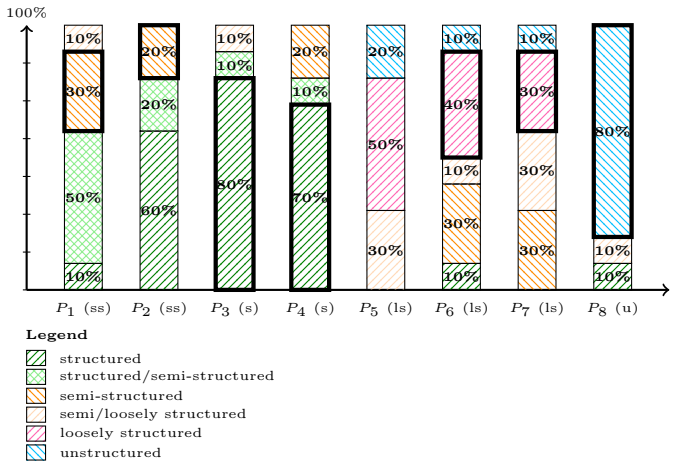


Fig. 7 Process classification results from the survey (s: structured, ss: semi-structured, ls: loosely structured, u: unstructured). Borderline predictions marked with dashed borders around both competing classes.

7.2 Need for Automated Process Classification

As part of the survey, participants were asked to assess the perceived difficulty of manually classifying business processes. Fig. 8 provides an overview of the results, displaying the perceived difficulty in relation to the correct level of structuredness. Only responses from participants who understood the task were considered, resulting in a total of 12 valid answers. The heatmap indicates that processes at both extremes—highly structured and unstructured—were generally perceived as easy or very easy to classify. This observation aligns with the results shown in the stacked bar chart in Fig. 7, where most participants correctly classified these processes and reported the task as easy.

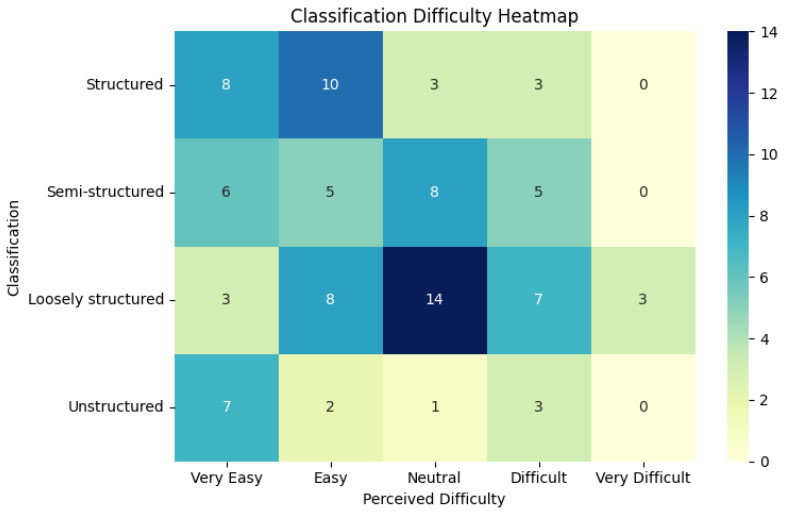


Fig. 8 Heatmap showing the assessment of the difficulty of manual classification in relation to the actual level of structuredness of processes P_1 to P_8

In contrast, processes with intermediate levels of structuredness were perceived as more challenging to classify, as reflected in the heatmap. While semi-structured processes were considered somewhat easier to classify than loosely structured ones, a clear trend emerges indicating that loosely structured processes are particularly difficult to assess. This difficulty is also evident in the responses related to processes P_6–P_7 shown in Fig. 7. Participants identified the processes as non-structured, but we observed variance in the levels of loosely to unstructured processes. We explain this with the perceived difficulty, which tends toward neutral or difficult. Thus, these findings indicate a distinct challenge in differentiating between intermediate levels of structuredness in manual business process classification.

8 Discussion & Conclusion

Organizations continuously face the challenge of balancing standardized and efficient business processes with the need for flexibility and innovation, a tension commonly discussed in the context of the devil's quadrangle of business process management [11]. Process maturity is a key concept in this regard, as higher levels of maturity are typically associated with standardized, efficient, and non-cost-intensive process behavior, whereas lower levels of maturity are characterized by higher variability and flexibility but higher cost and execution time. Additionally, organizations must select appropriate process representations for different BPM activities, such as process analysis, simulation, monitoring, and communication. This selection includes choosing among imperative, declarative, and hybrid process modeling paradigms, each suited to different behavioral characteristics. For both managing process maturity and selecting suitable modeling paradigms, the level of structuredness of a business process constitutes an informative indicator, as it reflects the degree of behavioral regularity,

control-flow stability, and predictability observed in process execution data. In current practice, assessments of process maturity and behavioral characteristics are largely performed through manual analysis of event logs by domain experts, which introduces subjectivity and depends heavily on individual expertise and experience.

To address these challenges, this paper proposes an automated approach for classifying business processes according to their level of structuredness, providing a more objective and reproducible basis for process characterization. The approach relies on a fine-grained analysis of activity relationships derived from event logs, distinguishing between temporal and existential activity dependencies, thereby enabling a more sensitive and nuanced classification of process structuredness. Moreover, we provide a set of labeled event logs that can be used for future research and new structuredness-aware classification algorithms. The underlying algorithm relies on a score-based mechanism with weight parameters derived from class definitions, expected process behavior, and empirical observations from development data. Our evaluation provides initial evidence for the validity and usefulness of the proposed approach, and the results indicate that activity dependency distributions contain sufficient information to characterize process structuredness. This finding provides an operational foundation for future structuredness-aware process mining techniques and process analysis methods.

Overall, the algorithm correctly classifies event logs originating from real-world and synthetic datasets, demonstrating its applicability. Due to the dominance mechanism, which is designed to detect a single, clearly correlated joint pattern, distributed flexibility, a characteristic of real-world event logs (cf. P_5), does not trigger a joint-ratio contribution, and the corresponding slots are instead scored from their marginal ratios. This behavior illustrates that real-world event logs can exhibit flexibility in more varied and less concentrated forms than the synthetic logs used for weight calibration, where individual dependency patterns tend to be more clearly separated. Nevertheless, the underlying dominance principle remains valid, as it correctly captures the strongly concentrated joint patterns characteristic of structured and semi-structured processes, and the P_5 case instead reflects a genuine characteristic of the observed process behavior rather than a shortcoming of the mechanism itself.

Further empirical validation and refinement of the weighting scheme may improve robustness and generalizability. Nevertheless, our sensitivity analysis showed that the weighting scheme is stable for 66% of the used logs, whereas the remaining cases reflect class-boundary ambiguity rather than weighting instability. Additionally, the computational effort of the discovery process increases with the number of activities, making the approach less efficient for large processes. The dependence of the presented classification approach on the underlying discovery is also reflected in its interaction with noisy event data. In the current implementation, this problem is addressed before dependency discovery through automatic trace-level filtering inspired by Conforti et al. [31]. On the synthetic noisy benchmark, this preprocessing restores the clean-log discovery result completely. At the same time, the reliability of this filtering depends on the presence of a clear gap in the variant frequency distribution; in cases where no such gap is detected, the algorithm falls back on a fixed, size-aware floor that is not informed by the log's actual distribution. For highly variable real-world logs, this

means that rare-but-legitimate variants are more likely to be preserved when a distributional cliff is present, but remain at risk of being discarded whenever no such cliff exists. Calibrating this fallback behavior, for example by validating whether a detected gap reliably reflects a genuine noise boundary rather than a small-sample artifact, remains an important direction for future work. Potentially higher classification accuracy could be achieved by supervised machine learning techniques; however, their application provides limited insight into which behavioral characteristics contribute to a classification outcome. Because structuredness represents a theory-driven process characteristic rather than a purely predictive target variable, the proposed score-based approach explicitly links structuredness classes to interpretable activity dependency patterns, thereby supporting both classification and process analysis. Future work may investigate machine learning techniques once larger labeled datasets become available. Additionally, to the best of our knowledge, no existing approach directly operationalizes the structuredness classes used in this paper, limiting opportunities for a direct baseline comparison.

Evaluation results are compared with a manually classified ground truth derived from static directly-follows graphs (DFGs), which may not fully capture the nuances of real process behavior. An interactive classification approach in which users actively explore event logs could yield more accurate results; however, it would also significantly increase user effort. To mitigate this issue, we supplemented real-world event log classification with evidence from existing literature. This trade-off results in a lower experimental complexity and reduced user burden, while still providing sufficient information for evaluation. Nevertheless, we can derive first insights regarding the approach's value for processes exhibiting mixed structured and flexible behavior. For these processes, the results show that manual assessments of process structuredness are particularly challenging, whereas agreement is higher for processes located at the extremes of structuredness. Unstructured processes are easy to identify, as they are characterized exclusively by independence relations and thus contain no behavioral constraints. However, this class is primarily a theoretical construct rather than an empirically grounded category. To remain consistent with the structuredness levels introduced by [2], we nevertheless included it in the classification algorithm. Future work may investigate whether such processes occur in real-world settings or whether the class should be reconsidered or omitted in practical applications.

With the proposed classification algorithm, this paper contributes toward a more systematic investigation of process structuredness as a process characteristic. The paper opens a relevant avenue for future research to investigate how such classification outcomes can be operationalized and integrated into BPM practices. Furthermore, we would like to encourage the community to investigate the practical usefulness of structuredness-aware classification in organizational settings. Case studies can assess how identified dependency types and the level of structuredness support decision-making and process analysis within individual organizations.

Supplementary information. The discovery algorithm, the score-based classification algorithm, and datasets used for development, testing, and evaluation are publicly available on GitHub: https://github.com/INSM-TUM/armature

Ethics Declaration. Not applicable.

Funding Declaration. The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Competing Interests Declaration. The authors declare no competing interests.

References

[1] Weske, M.: Business Process Management - Concepts, Languages, Architectures, Third Edition. Springer, Berlin, Heidelberg (2019). https://doi.org/10.1007/978-3-662-59432-2

[2] Ciccio, C.D., Marrella, A., Russo, A.: Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches. J. Data Semant. **4**(1), 29–57 (2015) https://doi.org/10.1007/S13740-014-0038-4

[3] Aalst, W.M.P.: Process Mining - Data Science in Action, Second Edition. Springer, Berlin, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4

[4] Adamo, G., Francescomarino, C.D., Ghidini, C., Maggi, F.M.: Beyond arrows in process models: A user study on activity dependences and their rationales. Inf. Syst. **100**, 101762 (2021) https://doi.org/10.1016/J.IS.2021.101762

[5] Hashmi, M., Governatori, G., Lam, H.-P., Wynn, M.T.: Are we done with business process compliance: state of the art and challenges ahead. Knowledge and Information Systems **57**(1), 79–133 (2018)

[6] Aguilar-Savén, R.S.: Business process modelling: Review and framework. International Journal of Production Economics **90**(2), 129–149 (2004) https://doi.org/10.1016/S0925-5273(03)00102-6 . Production Planning and Control

[7] Mili, H., Tremblay, G., Jaoude, G.B., Lefebvre, É., Elabed, L., Boussaidi, G.E.: Business process modeling languages: Sorting through the alphabet soup. ACM Computing Surveys (CSUR) **43**(1), 1–56 (2010)

[8] Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. CoRR **abs/1705.02288** (2017) 1705.02288

[9] Slaats, T.: Declarative and hybrid process discovery: Recent advances and open challenges. J. Data Semant. **9**(1), 3–20 (2020) https://doi.org/10.1007/S13740-020-00112-9

[10] Tregear, R.: Business process standardization. In: Brocke, J., Rosemann, M. (eds.) Handbook on Business Process Management 2, Strategic Alignment, Governance, People and Culture, 2nd Ed. International Handbooks on Information Systems, pp. 421–441. Springer, Berlin, Heidelberg (2015). https://doi.org/10.1007/978-3-642-45103-4_18

[11] Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management, Second Edition. Springer, Berlin, Heidelberg (2018). https://doi.org/10.1007/978-3-662-56509-4

[12] Kipping, G., Djurica, D., Franzoi, S., Grisold, T., Marcus, L., Schmid, S.J., Brocke, J., Mendling, J., Röglinger, M.: How to leverage process mining in organizations - towards process mining capabilities. In: Ciccio, C.D., Dijkman, R.M., del-Río-Ortega, A., Rinderle-Ma, S. (eds.) Business Process Management - 20th International Conference, BPM 2022, Münster, Germany, September 11-16, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13420, pp. 40–46. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-16103-2_5

[13] Wurm, B., Schmiedel, T., Mendling, J., Fleig, C.: Development of a measurement scale for business process standardization. In: Bednar, P.M., Frank, U., Kautz, K. (eds.) 26th European Conference on Information Systems: Beyond Digitization - Facets of Socio-Technical Change, ECIS 2018, Portsmouth, UK, June 23-28, 2018, p. 21 (2018)

[14] Andree, K., Bano, D., Weske, M.: A closer look at activity relationships to improve business process redesign. Softw. Syst. Model. **24**(1), 69–86 (2025) https://doi.org/10.1007/S10270-024-01234-5

[15] Andree, K., Burke, H., Beest, N., Tosatto, S.C., Pufahl, L.: Why behavioral representations matter: A framework for comparing their expressiveness in business process analysis. In: Business Process Management. Springer, Cham (accepted for publication)

[16] Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley (2005). https://doi.org/10.1002/0471741442

[17] Aalst, W.M.: Business process management: a comprehensive survey. International Scholarly Research Notices **2013**(1), 507984 (2013)

[18] Fahland, D., Lübke, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: The issue of understandability. In: Enterprise, Business-Process and Information Systems Modeling, BPMDS, EMMSAD, Held at CAiSE 2009, Amsterdam, The Netherlands, June 8-9, 2009. Proceedings. LNBIP, vol. 29, pp. 353–366. Springer, Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01862-6_29

[19] Rosenfeld, A.: BPM: structured vs. unstructured. BPTrends (2011)

[20] Montali, M.: The condec language. In: Specification and Verification of Declarative Open Interaction Models. LNBIP, vol. 56, pp. 59–83. Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14538-4_3

[21] Dongen, B.F., Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., Aalst,

W.M.P.: The prom framework: A new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings. Lecture Notes in Computer Science, vol. 3536, pp. 444–454. Springer, ??? (2005). https://doi.org/10.1007/11494744_25

[22] Berti, A., Zelst, S., Schuster, D.: Pm4py: A process mining library for python. Software Impacts **17**, 100556 (2023) https://doi.org/10.1016/j.simpa.2023.100556

[23] Imran, M., Ismail, M.A., Hamid, S., Nasir, M.H.N.B.M.: Complex process modeling in process mining: A systematic review. IEEE Access **10**, 101515–101536 (2022) https://doi.org/10.1109/ACCESS.2022.3208231

[24] Maggi, F.M., Slaats, T., Reijers, H.A.: The automated discovery of hybrid processes. In: BPM - 12th International Conference, Haifa, Israel, September 7-11, 2014. Proceedings. LNCS, vol. 8659, pp. 392–399. Springer, ??? (2014). https://doi.org/10.1007/978-3-319-10172-9_27

[25] Smedt, J.D., Weerdt, J.D., Vanthienen, J.: Fusion miner: Process discovery for mixed-paradigm models. Decis. Support Syst. **77**, 123–136 (2015) https://doi.org/10.1016/J.DSS.2015.06.002

[26] Schunselaar, D.M.M., Slaats, T., Maggi, F.M., Reijers, H.A., Aalst, W.M.P.: Mining hybrid business process models: A quest for better precision. In: Business Information Systems - 21st International Conference, BIS 2018, Berlin, Germany, July 18-20, 2018, Proceedings. LNBIP, vol. 320, pp. 190–205. Springer, ??? (2018). https://doi.org/10.1007/978-3-319-93931-5_14

[27] Fong, E.N., Jefferson, D.K.: Reference models for standardization. Computer Standards & Interfaces **5**(2), 93–98 (1986) https://doi.org/10.1016/0920-5489(86)90075-9

[28] Iacometta, C., Ciccio, C.D.: Declarative process mining with minerful, reloaded. In: Proceedings of the BPM Demos & Resources Forum 2025. CEUR Workshop Proceedings, vol. 4032, pp. 328–335 (2025)

[29] Burattin, A.: PLG2: multiperspective process randomization with online and offline simulations. In: Azevedo, L., Cabanillas, C. (eds.) Proceedings of the BPM Demo Track 2016 Co-located with the 14th International Conference on Business Process Management (BPM 2016), Rio de Janeiro, Brazil, September 21, 2016. CEUR Workshop Proceedings, pp. 1–6. CEUR-WS.org, ??? (2016)

[30] Aalst, W.M.P., Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed Parallel Databases **14**(1), 5–51 (2003) https://doi.org/10.1023/A:1022883727209

[31] Conforti, R., La Rosa, M., Hofstede, A.H.M.: Filtering out infrequent behavior from business process event logs. IEEE Transactions on Knowledge and Data Engineering **29**(2), 300–314 (2017) https://doi.org/10.1109/TKDE.2016.2614680

[32] Cremerius, J., Pufahl, L., Klessascheck, F., Weske, M.: Event log generation in MIMIC-IV research paper. In: Process Mining Workshops - ICPM 2022 International Workshops, Bozen-Bolzano, Italy, October 23-28, 2022, Revised Selected Papers. LNBIP, vol. 468, pp. 302–314. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-27815-0_22

[33] Diba, K., Remy, S., Pufahl, L.: Compliance and performance analysis of procurement processes using process mining. In: International Conference on Process Mining (2019)

[34] Bandara, H.M.N.D., Bockrath, H., Hobeck, R., Klinkmüller, C., Pufahl, L., Rebesky, M., Aalst, W.M., Weber, I.: Event logs of ethereum-based applications: A collection of resources for process mining on blockchain data. In: BPM (PhD/Demos), pp. 161–165 (2021)

[35] Hobeck, R., Klinkmüller, C., Bandara, H.M.N.D., Weber, I., Aalst, W.M.P.: Process mining on blockchain data: A case study of augur. In: BPM - 19th International Conference, Rome, Italy, September 06-10, 2021, Proceedings. LNCS, vol. 12875, pp. 306–323. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85469-0_20