UHASSELT

KNOWLEDGE IN ACTION

# Faculty of Business Economics

## Master of Management

***Master's thesis***

***Enhancing Data Quality Assessment in Process Mining: Extending DaQAPOs Functionalities***

**Hale Yurttutan**

Thesis presented in fulfillment of the requirements for the degree of Master of Management, specialization Data Science

**SUPERVISOR :**

Prof. dr. Niels MARTIN

**2023**
**2024**

UHASSELT

KNOWLEDGE IN ACTION

**www.uhasselt.be**
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek


# Faculty of Business Economics

## Master of Management

***Master's thesis***

***Enhancing Data Quality Assessment in Process Mining: Extending DaQAPOs Functionalities***

**Hale Yurttutan**

Thesis presented in fulfillment of the requirements for the degree of Master of Management, specialization Data Science

**SUPERVISOR :**

Prof. dr. Niels MARTIN

# Enhancing Data Quality Assessment in Process Mining: Extending DaQAPO's Functionalities

Hale Yurttutan

Hasselt University

hale.yurttutan@student.uhasselt.be

**Process mining focuses on identifying, tracking, and enhancing processes using information from event logs derived from process execution data captured by information systems. The effectiveness of process mining heavily depends on the quality of these event logs. Recognizing the critical need for high-quality data, this thesis proposes a systematic approach by following the design science research methodology to extend the functionalities of DaQAPO, a prominent tool in data quality assessment. By integrating comparative analysis of various data quality assessment tools with a combined data quality issue framework, this study presents detection strategies and develops functionalities that specifically address event log quality issues not yet covered by DaQAPO. These functionalities are designed to provide a more comprehensive tool for data quality assessment, ultimately facilitating more reliable process mining insights. This work contributes to the field of process mining by bridging the gap between theoretical data quality frameworks and their practical application in enhancing data quality assessment in process mining.**

***Keywords -*** *Process Mining; Event log quality issues; Event log quality assessment; Extension of DaQAPO Functionalities; R*

**Acknowledgements:** I would like to express my deepest gratitude to my supervisor, Prof. dr. Niels Martin, for his valuable guidance and support throughout this study. His expertise and insights were indispensable to my research. I am also grateful to my family for their consistent encouragement and support. Finally, a special thanks to my husband, Uygar, for his endless patience and love.

# 1 Introduction

Process mining focuses on identifying, tracking, and enhancing processes using information from event logs. These logs are derived from process execution data captured by information systems, serving as the foundational data for process mining (Bose et al., 2013). An event log comprises records of past occurrences within process executions, with each instance typically denoting a case, an activity, a specific time, and sometimes an associated resource. This type of data is distinct due to its clear semantics, such as the relationship between cases and events, and its inherent temporal sequence facilitated by timestamps (ter Hofstede et al., 2023).

By delving into operational workflows, process mining proves to be integral across various sectors, including production, logistics, finance, sales, education, consulting, healthcare, maintenance, and government (van der Aalst, 2022b). To illustrate the diverse purposes of process mining briefly, different types of it can be investigated. Process discovery involves automatically extracting process models from event logs, capturing the typical flow of a process. Conformance checking, another type of process mining, compares these models with actual event logs to identify deviations and similarities, providing diagnostic insights. Lastly, model enhancement utilizes event logs and existing models to improve or extend process representations, resulting in refined process models. These types of process mining collectively enable organizations to understand, monitor, and optimize their processes effectively (van der Aalst et al., 2012).

Obtaining accurate insights into business processes with the help of process mining can be challenging because of the data quality of event logs. In practical settings, it is observed that real-life event logs often fall short of data quality expectations, such as inconsistencies in labeling, missing details, and incorrect entries (Martin, 2021). In this context, the principle of "garbage in, garbage out" signifies the importance of input data quality in process mining (Suriadi et al., 2017). This principle means that if the input data is flawed, the resulting analysis will also be flawed. This dependence on data quality emphasizes the need to ensure the data to be used is of sufficient quality. To illustrate in a practical setting, consider a manufacturing process where the timestamps for the "Start Production" activity are inaccurately recorded. It may lead to an incorrect representation of the process duration. As a result, analyses that depend on these durations, such as calculating average process times or identifying deviations from standard operations, may be flawed. Another issue might occur in a hospital setting, where

the logs fail to record which staff member administered each treatment to patients, making it difficult to track accountability and assess performance. These types of data inaccuracies can significantly impact the effectiveness of process mining efforts and the accuracy of the insights derived (van der Aalst, 2012).

Given these challenges, assessing data quality in the context of process mining becomes a crucial practice. This assessment focuses on the identification of data quality issues in event logs (van der Aalst, 2022a). Motivated by these mentioned challenges, research in event log quality assessment has begun to concentrate on developing tools for this purpose. Notable examples of such tools include ProM plugins (Verhulst, 2016; Kherbouche et al., 2016; Dixit et al., 2018; Fischer et al., 2020), QUELI (Andrews et al., 2018), and DaQAPO (Martin et al., 2022), each offering various functionalities for the event log quality assessment. Among these tools, the focus of this study will be to provide improved support for the existing functionalities of DaQAPO. DaQAPO, an R-package, is equipped with an extensive array of tests designed to identify various event log quality issues while also providing the adaptability to identify quality issues specific to a particular application context (Martin et al., 2022). However, there exists a research gap as DaQAPO only covers a subset of data quality issues. This gap limits the effectiveness of the tool in detecting a broader range of issues that can occur in event logs. This research aims to contribute by developing extensions to the current functionality of DaQAPO. These enhancements will be grounded within the combined framework of data quality issues outlined in the literature and will be informed by a comparative analysis of the functionalities of existing data quality assessment tools such as ProM and QUELI. The study categorizes extended functionalities into three areas: resource-related, timestamp-related, and activity-related.

The structure of this thesis is outlined as follows: Section 2 provides an overview of the background and related work. Section 3 outlines the research questions and describes the methodology employed. Section 4 reviews the functionalities of DaQAPO, explores areas for improvement, and examines selected data quality issues, detection strategies, and function outputs. Section 5 presents a demonstration of the developed functionalities on a publicly available event log. Section 6 provides a discussion of the findings. Then, the paper concludes with a summary of key findings and future research directions in Section 7.

# 2 Background and Related Work

This section introduces the fundamental concept of an event log and explores the various data quality issues that can arise within these logs. Then, it examines the tools developed to assess data quality in event logs, including ProM plugins, QUELI, and DaQAPO. Additionally, a comparative analysis of the functionalities of these tools is conducted, focusing on their ability to identify different data quality issues.

## 2.1 Event Log

The starting point for process mining is an event log. An event log contains data regarding the execution of a process, which includes multiple process instances or cases (Mans et al., 2015). Each event record captures a specific action within a process and includes attributes such as the process instance (case ID), the specific process action (activity), the execution time (timestamp), and the transaction type (e.g., start or complete). It may also contain other additional information, such as the resource responsible for the activity (Andrews et al., 2020). For a more concrete example, imagine an event log from a hospital where each entry records the steps of a patient's hospital visit, as seen in Table 1. Initially, patient 520 was registered at the hospital, which was handled by Clerk 12 on November 21, 2017. It captures the start of the registration at 11:59:41 and its completion at 12:05:52. Subsequent steps might include various examinations and treatments by different healthcare professionals, each logged with corresponding times and resources.

Table 1: Illustration of an event log structure.

| Patient_visit_nr | Activity | Timestamp | Transaction type | Resource |
|---|---|---|---|---|
| 520 | Registration | 21/11/2017 11:59:41 | Start | Clerk 12 |
| 520 | Registration | 21/11/2017 12:05:52 | Complete | Clerk 12 |
| 520 | Triage | 21/11/2017 11:59:41 | Start | Nurse 17 |
| 520 | Triage | 21/11/2017 13:32:18 | Complete | Nurse 17 |
| 520 | Clinical exam | 21/11/2017 11:59:41 | Start | Doctor 4 |
| 520 | Clinical exam | 21/11/2017 14:02:34 | Complete | Doctor 4 |
| 520 | Treatment | 21/11/2017 14:38:52 | Start | Nurse 17 |
| 520 | Treatment | 21/11/2017 14:49:21 | Complete | Nurse 17 |
| 521 | Registration | 21/11/2017 18:02:10 | Start | Clerk 6 |
| 521 | Registration | 21/11/2017 18:04:07 | Complete | Clerk 6 |
| … | … | … | … | … |

## 2.2 Data Quality Issues in Event Logs

An event log data quality issue refers to any problem or deficiency in the recorded data that could compromise the correctness, completeness, or relevance of information in an event log (Bose et al., 2013). To illustrate, in a healthcare setting event log, the timestamps for patient treatments might only record the hour and minute, omitting the exact second when the treatment began and ended. This lack of precision can lead to challenges in accurately measuring the duration of treatments, impacting analyses related to treatment efficiency or the identification of bottlenecks in patient flow. A wide range of quality issues can arise in event logs. This section investigates the studies by Bose et al. (2013), Suriadi et al. (2017), and Vanbrabant et al. (2019), which provide an overview of potential data quality issues within event logs.

### 2.2.1 Bose et al. (2013)

Bose et al. (2013) identify four main categories of problems related to the quality of event logs: missing data, incorrect data, imprecise data, and irrelevant data. The study further explains how these identified categories of problems occur across different log entities, in total defining 27 distinct data quality issues.

**Missing Data:** Essential information may be missing from the log, such as events, their specific attributes, or their relationships. This typically indicates problems with the logging system or procedure, where required elements fail to be recorded. For instance, a patient undergoes a series of diagnostic tests, but the event log fails to record the timestamp of a test due to a system malfunction. This missing timestamp could lead to inaccuracies in calculating the total duration of diagnostic procedures for the patient.

**Incorrect Data:** Although data is routinely recorded, there are instances where it may be incorrect. Imagine an event where a patient's medication administration is logged with the wrong drug name due to a clerical error. This might lead to incorrect data analysis regarding drug usage patterns.

**Imprecise Data:** The detail in logged entries may be insufficient, leading to a loss of necessary detail. This can affect the reliability of analyses, especially where precise values are critical. For instance, in a hospital setting, recording the timestamp of medication administration only to the nearest hour may lead to challenges in precisely determining when patients receive their medications. This lack of precision could potentially make it more difficult to monitor the effectiveness of treatments.

**Irrelevant Data:** Some data in the logs may not directly serve the analysis but might need to be transformed or filtered to become useful. In a hospital setting, an event log might capture every single interaction between patients and non-medical staff, such as billing inquiries. While technically accurate, this data may not be relevant to clinical outcome analyses and could clutter the event log, requiring filtering out for studies focused solely on medical treatments (Bose et al., 2013).

### 2.2.2 Suriadi et al. (2017)

Suriadi et al. (2017) introduce a systematic approach to assess data quality in event logs through the identification of event log imperfection patterns. The article identifies 11 event log imperfection patterns, including form-based event capture and homonymous label, as detailed in Table 12 in Appendix A. For example, form-based event capture imperfection pattern highlights a problem in event log capture from electronic forms, such as medical test forms. When users like nurses and doctors click "Save", all form data records at that moment, losing precise activity times. Another imperfection pattern is the homonymous label, which occurs when a recurring activity is labeled identically for a case. For example, in a hospital setting, if "Treatment evaluation" is recorded three times for a case in the log, this repetition might be understood differently. It could indicate that "Treatment evaluation" is actually repeated, or it might indicate a review of the information in "Treatment evaluation", which actually refers to another activity, making the analysis more complex (Suriadi et al., 2017).

### 2.2.3 Vanbrabant et al. (2019)

Vanbrabant et al. (2019) discuss the significance of high-quality input data in creating realistic simulation models. The provided framework categorizes potential data quality problems in EHRs (Electronic Hospital Records) and describes techniques for identifying these issues. It distinguishes between missing and non-missing data. The non-missing data can be further categorized as wrong data and not wrong but not directly usable data. The latter refers to data that is accurate but requires preprocessing to be fit for the intended use. Each category is further subdivided into specific types of data quality problems, such as missing values, incorrect attribute values, violated attribute dependencies, and inconsistent formatting, as explained in Table 2.

Table 2: Data quality framework by Vanbrabant et al. (2019)

| Category | Description |
|---|---|
| **Missing Data** | |
| Missing Values | Mandatory values that are absent for certain records. |
| Missing Attributes | Required attributes for the analysis are not present in the data set. |
| Missing Entities | Entire records (e.g., patients) that are absent. |
| **Wrong Data** | |
| Violated Attribute Dependencies | Data values are incorrect due to their relationship with other attributes. |
| • Logical Order | Sequential activities that are not in the correct order. |
| • Mutual Dependency | Mutually dependent attributes with contradicting values. |
| Incorrect Attribute Values | Data values that are inherently wrong. |
| • Inexact Timestamps | Timestamps recorded inaccurately. |
| • Typing Mistakes | Errors in text fields due to typing errors. |
| • Outside Domain Ranges | Data values outside the expected range. |
| • Other Implausible Values | Incorrect values that do not fit other categories. |
| **Not Wrong but Not Directly Usable Data** | |
| Inconsistent Formatting | Data is correct but formatted inconsistently. |
| Implicit Value Needed | Required input is implied in the data but not explicitly recorded. |
| Embedded Values | More than one value embedded within a single data field. |
| Abbreviations | Use of unclear or useless abbreviations. |
| Imprecise Data | Data is correct but not detailed enough for the specific use case. |

## 2.3 Data Quality Assessment of Process Data

This section focuses on exploring data quality assessment tools such as ProM plugins, QUELI, and DaQAPO. Table 3 presents a comparison of the data quality assessment functionalities provided by these tools. These functionalities represent a comprehensive list derived from the related studies. This functionality comparison highlights each tool's functionalities in the identification of specific data quality issues.

Table 3: Functionality comparison of data quality assessment tools.

| | ProM | | | | QUELI | DaQAPO |
|---|---|---|---|---|---|---|
| | **Verhulst et al. (2016)** | **Kherbouche et al. (2016)** | **Dixit et al. (2018)** | **Fischer et al. (2020)** | **Andrews et al. (2018)** | **Vanbrabant et al. (2018)** |
| Missing values | + | + | + | + | | + |
| Threshold/Expected value range | + | + | + | + | + | + |
| Event order check | + | + | + | + | + | + |
| Attribute dependency | + | | | | | + |
| Predecessor check | | + | + | + | + | + |
| Frequency check | + | + | + | + | + | + |
| Timestamp format | + | | + | | | |
| Correct label | + | | | | | + |
| Synonymous labels | | | | | + | |
| Homonymous labels | | | | | + | + |
| Timestamp granularity | + | + | + | + | | |
| Overlapping events | | | | + | | + |
| Form-based event | | | | | + | |
| Duplicates | + | + | | + | + | + |

The literature presents various ProM plugins designed to identify different data quality issues. These plugins commonly include functionalities such as checking for missing values, threshold checks, event order checks, frequency analyses, and timestamp granularity assessments.

The study by Verhulst (2016) describes the implementation of a ProM plugin for assessing data quality issues such as consistency, correctness, uniqueness, and format checks, using an event log as the input. Unlike other ProM plugins in this literature review, Verhulst (2016) introduces a detection mechanism for event attribute dependency, identifying situations where certain attributes consistently appear together, such as a set of events that always share the same attributes. Furthermore, as a distinct functionality, the study analyzes the length of attribute values and examines the types of data they contain. Moreover, for timestamp format check, the study introduces functionality to identify anomalous timestamps, particularly those dated in the future, ensuring that event timestamps fall within a user-defined timeframe. The output is a scorecard providing an overall quality score for the event log, along with individual scores for each quality aspect. This output may have limitations in addressing context-specific needs because of uniform weighting across all quality dimensions.

Another ProM plugin studied by Kherbouche et al. (2016) assess complexity, accuracy, consistency, and completeness using an event log as the input. This plugin checks global completeness by comparing the occurrence frequencies of activity sets across different cases with their expected probabilities. The study also addresses local completeness, which assesses whether an event log accurately records the predecessor and successor tasks for each case to detect potential sequencing errors. Moreover, the study by Kherbouche et al. (2016) explores the complexity of event logs through two main aspects: structural and behavioral. The structural aspect focuses on the organization and interconnection of events in the log, identifying elements such as loops and duplicate tasks and analyzing the number of events and traces. The behavioral aspect examines the number and types of events per case, variations between activity sets across different cases, and the presence of recurring or unique behavioral patterns. The output of the plugin offers a generalized summary of event log quality by employing standardized metrics. Similar to Verhulst (2016), this approach restricts its effectiveness in assessing context-specific event logs.

The ProM plugin discussed by Dixit et al. (2018) identifies three main indicators of event order imperfections such as timestamp granularity issues, order anomalies, and statistical anomalies using event logs. Similar to Kherbouche et al. (2016), for ordering-based detection, the study examines the frequency of follows and precedes relations between activities to identify activities that may not follow

the anticipated sequence. Additionally, unlike Verhulst (2016), this plugin uses a different strategy to detect timestamp format issues. For example, if all activities within an event log have "day" values that range only between 1-12, with no values from 13–31, this could indicate a misinterpretation of the date format (e.g., confusing dd/mm with mm/dd), leading to problems with event ordering. The output identifies which activities are affected by these issues and includes quantitative details such as the frequency of each issue and examples of specific instances within the log.

Building on these insights, Fischer et al. (2020) introduces an approach for detecting and quantifying timestamp-related issues in event logs. The study defines 15 metrics related to timestamp quality across four abstraction levels (event, activity, trace, log) and four quality dimensions (accuracy, completeness, consistency, uniqueness). The plugin checks these quality dimensions on timestamp data, including the identification of missing timestamps, the granularity of timestamps, and any anomalously dated timestamps, similar to Verhulst (2016), while ensuring each timestamp is uniquely recorded. Additionally, unlike other ProM plugins, this plugin provides a unique functionality by examining overlapping activities per resource to detect instances of resource sharing across different events. Similar to the plugins by Kherbouche et al. (2016) and Dixit et al. (2018), the plugin investigates predecessor checks to identify deviations in activity sequencing. The output provides a quantified assessment of timestamp quality issues in the event log. The approach maintains its relevance and utility across various domains by adjusting the importance of metrics used in this assessment.

Besides ProM plugins, the article by Andrews et al. (2018) introduces QUELI, a query language designed to currently identify five event log imperfection patterns defined by Suriadi et al. (2017) such as form-based event capture, collateral events, inadvertent time travel, synonymous labels, and homonymous label. These detection strategies include identifying form-based event capture and collateral events by recognizing groups of events with nearly identical timestamps and analyzing the frequency of these groups. Another imperfection pattern, inadvertent time travel, involves identifying pairs of activities with unusual temporal ordering and using statistical summaries to determine if the deviations indicate data quality issues, functioning essentially as a predecessor check. Additionally, the synonymous label imperfection pattern addresses multiple distinct labels in the log that refer to the same activity but vary in syntax by finding labels that never appear together in the same case, suggesting they might be synonyms, confirmed through content analysis. Moreover, the homonymous label pattern arises when the same activity label is used for different activities within a case, identified by analyzing the context of each

occurrence to determine if the label represents different activities. Each detection algorithm generates outputs such as statistical summaries of specific features in an event log or the creation of sub-logs that fulfill particular criteria. The output of the QUELI allows users to configure event patterns, threshold values, algorithm parameters, and contextual variables.

Finally, DaQAPO provides a wide range of specific functionalities for data quality assessment. For instance, it can identify frequency violations of activities and detect event order violations, time anomalies, and overlapping activities. DaQAPO's functionalities are characterized by their dynamic nature, offering context-specific assessments of event log quality, such as setting thresholds for identifying duration outliers, specifying conditions for detecting related activities, or performing event order check (Martin et al., 2022). Similar to QUELI, DaQAPO offers a homonymous label check in which the same activity label is used for different activities within a case. The comparison of functionalities presented in Table 3 indicates that DaQAPO lacks functionalities related to timestamp format, timestamp granularity, synonymous labels, and form-based event capture.

# 3 Research Questions and Methodology

This research aims to bridge a gap identified in the current capabilities of DaQAPO. While DaQAPO is powerful at detecting a variety of data quality issues, it requires additions to address a larger variety of data quality issues. The primary research question that guides this effort is: **How can the functionalities of DaQAPO be extended to provide improved support for data quality assessment of process data?** This question further subdivides into several subquestions aimed at targeting the enhancements needed for DaQAPO. The design science research methodology, which involves identifying problems, defining objectives, designing and developing solutions, demonstrating their applicability, and evaluating their effectiveness (Peffers et al., 2007), was employed in this study to systematically explore and develop these enhancements.

*Subquestion 1: Which process data quality issues are currently not supported in DaQAPO?*

Investigating the existing functionalities of DaQAPO is crucial as it sets the foundation for possible extensions. By doing so, what the tool already offers can be understood, and where there may be gaps that hinder comprehensive event log quality assessment can be identified. To achieve this objective, the study employs a structured methodology grounded in a comprehensive review of existing literature and an in-depth analysis of the current functionalities of DaQAPO. This subquestion aligns with the design

science research methodology by identifying the problem and defining the objectives for improvement.

*Subquestion 2: Which assessment functions can be developed to address process data quality issues not yet covered by DaQAPO?*

In this study, the development of new functions is directly guided by the gaps identified in DaQAPO's current functionalities, as established through the extensive literature review and in-depth analysis addressed in subquestion 1. Exploring what further functionalities can be added to DaQAPO addresses the need for continuous improvement and contributes to the comprehensiveness of DaQaPO as a data quality assessment tool. This subquestion is relevant to the design science research methodology as it focuses on designing and developing new functionalities to address the identified gaps.

*Subquestion 3: Can the novel assessment functions generate insights into the data quality of a real-life event log?*

This question emphasizes the practical testing and demonstration of the developed functionalities. By applying the new assessment functions to a different, publicly available event log, the research can showcase the real-life applicability of the developed functionalities. This subquestion is integral to the design science research methodology, as it involves demonstrating and evaluating the effectiveness and utility of the designed functionalities in a real-life context.

# 4 Systematic Functionality Extension for DaQAPO

This section starts by examining DaQAPO's current functionalities. A combined data quality framework is then created to investigate a wider range of data quality issues for functionality extension. Following this, a mapping table aligns each issue identified in the combined framework with DaQAPO's functionalities, highlighting which issues are addressed by DaQAPO and which are not. Finally, the study delves into selected data quality issues, providing descriptions, real-life examples, detection strategies, and details on the functions developed and their outputs.

## 4.1 Mapping Existing Capabilities of DaQAPO

DAQAPO provides generic data quality assessment techniques developed to address various quality issues (Vanbrabant et al., 2019). Table 4 provides a comprehensive summary of DaQAPO's current functionalities, which sets the stage for identifying potential areas for functionality extension for DaQAPO.

Table 4: Summary of current functions of DaQAPO (Vanbrabant et al., 2019; Martin et al., 2022)

| **Function** | **Description** |
|---|---|
| detect_activity_freq_violations | Identifies anomalies in the frequency of specified activities within each case, ensuring that procedures are followed correctly without unnecessary repetition. |
| detect_activity_order_violations | Verifies that a specified sequence of activities is followed in each case, highlighting potential quality issues in procedures essential for patient safety and care quality. |
| detect_attribute_dependencies | Determines if a specific condition (antecedent) reliably leads to a particular outcome (consequent), identifying inconsistencies in process or data recording. |
| detect_case_id_seq_gaps | Checks for gaps or missing numbers in the sequence of case IDs, helping to identify missing cases. |
| detect_cond_activity_presence | Detects missing activities based on conditions. Checks for the consistent recording of specific activities under certain conditions, ensuring adherence to protocols based on patient conditions or specializations. |
| detect_duration_outliers | Identifies instances where the duration of an activity is unusually long or short, indicating potential inaccuracies or anomalies in patient care or process efficiency. |
| detect_inactive_periods | Identifies extended periods without arrivals or activities, which could signal technical failures or other issues, using a user-defined threshold for identifying long gaps. |
| detect_incomplete_cases | Checks for cases missing a complete set of specified activities, indicating potential issues with patient care processes or data entry. |
| detect_incorrect_activity_names | Identifies activity labels not part of the specified list of allowed activities, helping to correct typos, inconsistencies, or incorrect activity labeling. |
| detect_missing_values | Detects and quantifies missing or null values in specified columns, offering statistics on missing values per column and identifying records with missing data. |
| detect_multiregistration | Identifies instances where a single resource is associated with multiple registrations or activities within a short timeframe, pointing out potential batch registrations that may distort event sequences and timings. |
| detect_overlaps | Identifies instances where different activities overlap in time for the same case, indicating potential scheduling issues, multitasking by staff, or workflow inefficiencies. |
| detect_related_activities | Detects missing activities related to others. Verifies expected relationships between activities, ensuring that certain activities occur in conjunction with others as required by care processes or operational protocols. |
| detect_similar_labels | Identifies labels within a specified column that are similar to each other based on an allowed edit distance, helping to correct inconsistencies, typos, or variations in activity recording. |
| detect_time_anomalies | Detects unusual or impossible time-related patterns, such as negative durations or zero durations, which could indicate issues in data recording or process timing. |
| detect_unique_values | Lists all unique entries in a specified column, identifying any inconsistencies or anomalies and understanding the variety of activities recorded and the range of staff involved. |
| detect_value_range_violations | Checks if the values of certain attributes fall within an acceptable range, ensuring data integrity in fields with restricted numeric ranges critical for operational efficiency and patient care. |

Figure 1 illustrates the combined framework, which is developed by comparing and integrating the data quality issues and categories proposed by Bose et al. (2013) and Vanbrabant et al. (2019). This combined framework helps to cover a wider range of data quality issues and ensures a more comprehensive approach for functionality extension. It provides a clear visual representation of the covered areas by DaQAPO and highlights potential gaps. In the figure, the blue rectangles correspond to data quality issues captured by DaQAPO, while the remaining issues represent uncovered data quality issues, indicating potential functionality extensions.

As previously discussed in Section 2.3, the comparison of different data quality assessment tools revealed several uncovered functionalities by DaQAPO: timestamp format, timestamp granularity, synonymous labels, and form-based event capture. To effectively incorporate these functionalities into the combined framework, the corresponding data quality issues are identified as follows: inconsistent formatting for the timestamp format (Vanbrabant et al., 2019), imprecise timestamps for timestamp granularity (Bose et al., 2013), incorrect activity names for synonymous labels, and both incorrect timestamps and irrelevant events for form-based event capture (Suriadi et al., 2017). By investigating these additional data quality issues and their detection strategies given in the literature, the current study ensures that interpretation differences are also included in the identification of functionality extensions. For example, DaQAPO is capable of detecting activities logged with the same timestamp (Martin et al., 2022); however, it does not directly provide the interpretation of form-based activity loggings.

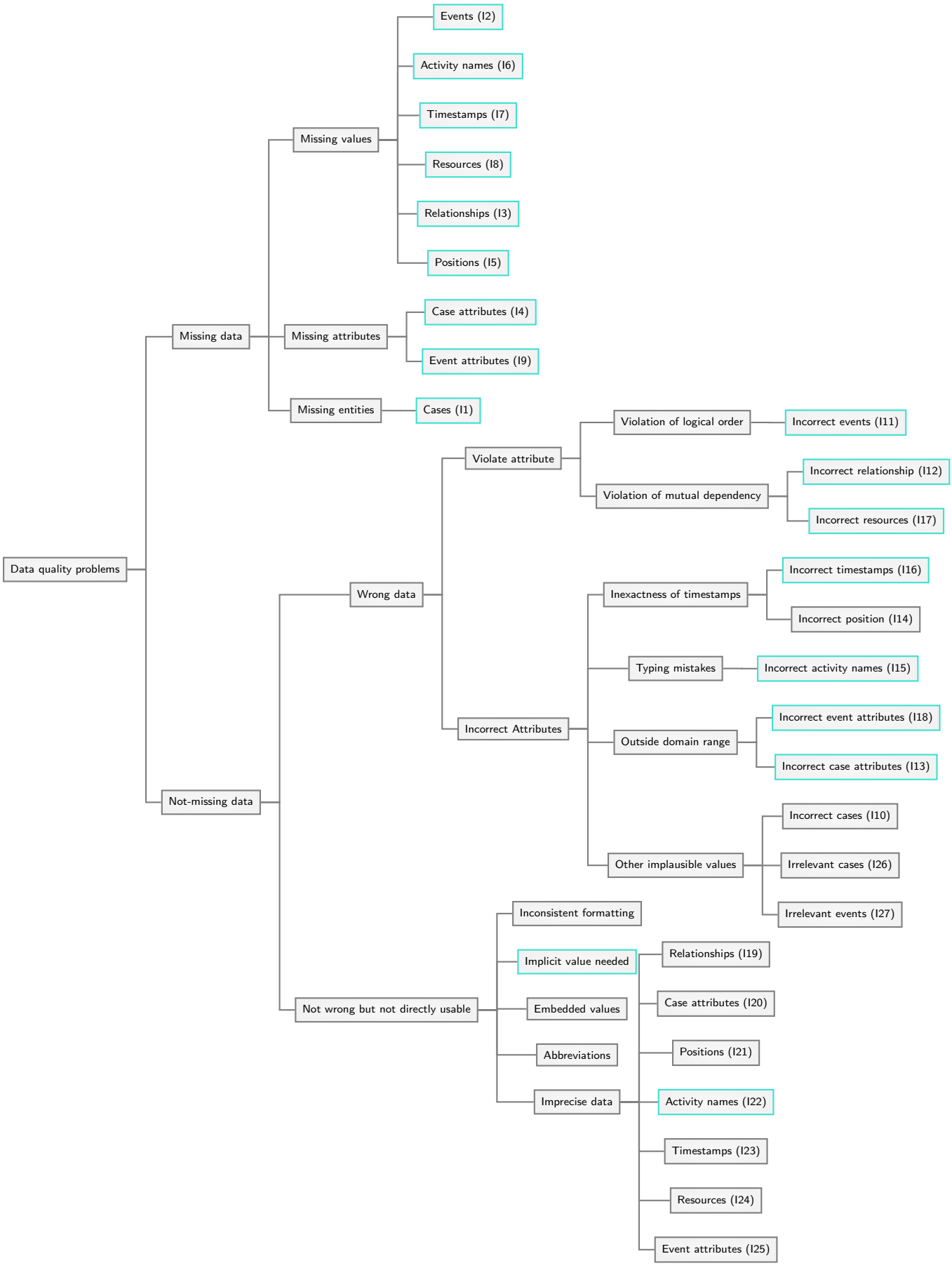


Figure 1: Combined data quality framework by Vanbrabant et al. (2019) and by Bose et al. (2013).

Table 5 presents a matching table for data quality issues within the combined data quality framework and the corresponding DaQAPO functionalities for identifying these issues. The matching is achieved by comparing the definitions of data quality issues with the functionalities of DaQAPO. This comparison systematically differentiates the current functionalities of DaQAPO and highlights potential areas for extension.

Table 5: Summary of quality issues in the combined data quality framework (Bose et al., 2013; Vanbrabant et al., 2019) and associated DaQAPO function (Martin et al., 2022)

| **Quality Issue** | **Explanation** | **DaQAPO Function** |
|---|---|---|
| Missing Cases (I1) | Cases executed in reality are not recorded in the event log, leading to a discrepancy between actual events and recorded data. | detect_case_id_seq_gaps<br>detect_missing_values |
| Missing Events (I2) | Some events that occurred in reality are omitted from the log, potentially leading to incorrect inferences about event relationships. | detect_incomplete_cases<br>detect_missing_values |
| Missing Relationships (I3) | The association between events and cases is unclear, making it difficult to track specific interactions. | detect_missing_values |
| Missing Case Attributes (I4) | Important data attributes like a patient's weight might be missing, affecting analysis quality. | detect_missing_values |
| Missing Position (I5) | Without timestamps, the sequence of events is unclear, complicating process flow analysis. | detect_missing_values |
| Missing Activity Names (I6) | Events recorded without specifying activity names lead to confusion and ambiguity. | detect_missing_values |
| Missing Timestamps (I7) | The absence of timestamps makes it challenging to perform accurate timing and sequencing analysis. | detect_missing_values |
| Missing Resources (I8) | Not recording who performed an activity can lead to an incomplete analysis of organizational structure and resource allocation. | detect_missing_values |
| Missing Event Attributes (I9) | Omitting critical details like the amount in a "register loan" event can lead to inaccurate financial analyses. | detect_missing_values |
| Incorrect Cases (I10) | Cases from a different process logged in error introduce outliers and can mislead the analysis. | - |
| Incorrect Events (I11) | Incorrectly logged events lead to false inferences about the process and its flow. | detect_activity_order_violations<br>detect_related_activities |
| Incorrect Relationships (I12) | Incorrectly linking events to cases can result in inaccurate process models and a misunderstanding of the data. | detect_cond_activity_presence |
| Incorrect Case Attributes (I13) | Errors in recording case attributes can significantly affect decision-making and analysis accuracy. | detect_value_range_violations |
| Incorrect Position (I14) | Events positioned wrongly because of inaccurate timestamps can distort the identified process sequence. | - |

| Quality Issue | Explanation | DaQAPO Function |
|---|---|---|
| Incorrect Activity Names (I15) | Incorrectly labeled activities can mislead process analysis and affect outcome interpretation. | detect_similar_labels<br>detect_incorrect_activity_names |
| Incorrect Timestamps (I16) | Inaccurate timestamps can misconstrue the order of events, affecting the reliability of control-flow analysis. | detect_inactive_periods<br>detect_duration_outliers<br>detect_time_anomalies<br>detect_overlaps |
| Incorrect Resources (I17) | Attributing an activity to the wrong person can result in incorrect understanding of how an organization functions. | detect_multiregistration |
| Incorrect Event Attributes (I18) | Wrong details in event attributes lead to faulty analyses, like incorrect transition guards. | detect_attribute_dependencies |
| Imprecise Relationships (I19) | A too-broad case definition can make unclear the detailed and accurate mapping of events to their respective contexts. | - |
| Imprecise Case Attributes (I20) | Coarse values for case attributes restrict the precision of derived statistics. | - |
| Imprecise Position (I21) | Imprecise ordering of events when some events that occurred in parallel are recorded sequentially, causing inaccuracies in their actual chronological order. | - |
| Imprecise Activity Names (I22) | Coarse activity names result in multiple events with the same name in a trace, which may represent the same or different task instances. | detect_activity_freq_violations |
| Imprecise Timestamps (I23) | Coarse timestamp granularity result in multiple events having the same or inconsistent levels of timestamp detail, leading to unreliable event ordering. | - |
| Imprecise Resources (I24) | Coarse resource information, like recording activity at the department level instead of specifying the individual, restricting detailed organizational relationships. | - |
| Imprecise Event Attributes (I25) | Coarse or imprecise event attributes lead to a loss of detail and potential misinterpretation in process analysis. | - |
| Irrelevant Cases (I26) | Including cases not relevant to the specific analysis adds unnecessary complexity. | - |
| Irrelevant Events (I27) | Events irrelevant to the analysis need filtering out to prevent inaccurate or overly complex models. | - |
| Inconsistent Formatting | Variations in data format among attributes (e.g. month and day portions of the timestamp have been transposed). | - |
| Implicit Value Needed | Missing inherent attribute values can be assigned later based on context, such as duration. | detect_inactive_periods<br>detect_duration_outliers<br>detect_time_anomalies |
| Embedded Value | Single data field contains multiple values and requires splitting for analysis. | - |
| Abbreviations | Abbreviated terms in event log. | - |

## 4.2 Selected Data Quality Issues for Data Quality Assessment Functionality Extension

In this section, the selected data quality issues for data quality assessment are explored. These issues are categorized into three main areas: resource-related, timestamp-related, and activity-related issues.

Investigating resource-related data quality issues is essential, as numerous studies highlight their critical impact on the reliability of process mining insights (Pika et al., 2017; ter Hofstede et al., 2023). These issues are particularly important from the organizational perspective of process mining. Accurate identification and classification of resources according to their roles and organizational units, as well as effective mapping of the social network within the organization, rely on resource-related data quality (van der Aalst et al., 2012). For instance, accurate data can reveal overburdened employees or highlight communication gaps between departments. Addressing these data quality issues allows process mining to uncover inefficiencies, bottlenecks, and opportunities for improvement within business processes (Song & van der Aalst, 2008). Furthermore, it can streamline approval workflows, reduce processing times, or enhance resource allocation. Thus, by ensuring the reliability of resource-related data, companies can fully leverage the potential of process mining to drive significant organizational improvements and achieve competitive advantages (Huang et al., 2012; van der Aalst, 2016).

The accuracy of timestamp data is another critical dimension of data quality that requires rigorous attention for accurate process mining insights. Timestamps are pivotal for accurately sequencing activities, a necessity for developing correct process discovery models that can accurately reflect activity dependency relations (Fischer et al., 2020; Dixit et al., 2018). For instance, a process may be discovered as parallel, while in reality, this might not be the case, especially when timestamps are inaccurate, or events are logged out of order due to system delays or synchronization issues. Such discrepancies not only introduce substantial challenges in accurately processing event sequences but also hinder critical time-related performance analyses, for which an example could be evaluating process Key Performance Indicators (KPIs) (van der Aalst et al., 2008). These KPIs are utilized when verifying that the current process aligns with business rules that consider the timing and sequence of the process activities (Conforti et al., 2020). Furthermore, when events are marked with accurate and precise timestamps, it allows for the identification of bottlenecks, the measurement of service standards, the monitoring of resource utilization, and the prediction of throughput times (van der Aalst et al., 2012).

Finally, activity-related data quality issues have a significant impact on performance analysis and process discovery in business processes (Sadeghianasl et al., 2019). Incorrect or inconsistent activity

names can lead to superfluous relationships and misrepresentation of actual process flows (Chen et al., 2022; Sadeghianasl et al., 2020). Misunderstanding the order and dependencies of activities can lead to inefficient workflows, as it may result in process improvement initiatives that would not address the actual needs (van der Aalst, 2011). Additionally, performance metrics related to specific activities might be incorrect if those activities are not consistently logged. This affects the accuracy of performance analysis, such as determining which activities are bottlenecks (Suriadi et al., 2017). Therefore, activity-related data quality issues also need to be addressed to ensure accurate process mining analysis.

A sample activity log in a hospital setting is used to develop these functionalities, as seen in Table 6. This activity log is a transformed version of the event log described in Section 2.1. Each entry in the log records an instance of an activity, including the start and completion timestamps, as well as the specific resource (i.e., originator) and the case involved (i.e., patient visit number) (Martin et al., 2022).

Table 6: Illustration of an activity log structure.

| Patient_visit_nr | Activity | Resource | Start_ts | Complete_ts |
|---|---|---|---|---|
| 520 | Registration | Clerk 9 | 21/11/2017 16:42:08 | 21/11/2017 16:51:59 |
| 520 | Triage | Nurse 27 | 21/11/2017 17:26:30 | 21/11/2017 18:01:01 |
| 520 | Clinical exam | Doctor 4 | 21/11/2017 18:16:49 | 21/11/2017 18:22:08 |
| 520 | Treatment | Nurse 17 | 21/11/2017 18:26:04 | 21/11/2017 18:55:00 |
| 520 | Treatment evaluation | Doctor 4 | 22/11/2017 10:01:55 | 22/11/2017 10:10:10 |
| 521 | Triage | Nurse 5 | 21/11/2017 17:04:03 | 21/11/2017 17:06:05 |
| ... | ... | ... | ... | ... |

### 4.2.1 Resource-related Data Quality Issues

This section discusses the data quality issues that arise when resource information is imprecise, violates resource consistency rules, or does not accurately reflect resource and activity match.

- **Imprecise Resource**

  *Description and Real-life Example*

  The imprecise resource issue manifests when resources are recorded using generalized or vague identifiers rather than detailed, specific ones (Bose et al., 2013). This issue is exemplified in Table 7, where the log shows broad categories, such as general roles (e.g., "Clerk" or "Doctor"), instead of precise identifiers (e.g., "Clerk 9" or "Doctor 4"). This lack of precision in resources may hinder process mining analysis, making it difficult to assess individual performance and identify resources that are either overburdened or underutilized (Song & van der Aalst, 2008).

Table 7: Illustration of an activity log with imprecise resource.

| Patient_visit_nr | Activity | Originator | Start_ts | Complete_ts |
|---|---|---|---|---|
| 523 | Registration | *Clerk* | 21/11/2017 17:26:30 | 21/11/2017 16:51:59 |
| 523 | Triage | Nurse 27 | 21/11/2017 17:26:30 | 21/11/2017 18:01:01 |
| 523 | Clinical exam | *Doctor* | 21/11/2017 17:26:30 | 21/11/2017 18:22:08 |

*Detection Strategy*

The detection strategy involves a thorough examination of resources to determine whether they encapsulate specific details, such as individual employee identifiers. By identifying the instances where resource naming is not similar in structure to the sample user input, the algorithm highlights entries that do not match the standards for resource identification. For example, assume that employees are identified with numbers that have 7 digits, such as "4022156". If the resource information is given as "402215", it will be flagged because it does not match the number of digits required for preciseness.

*Developed Function and Output*

To address this issue, a function `imprecise_resource` was developed, which detects imprecise resource identifiers. The function checks resource format based on the sample user input. It checks format as a word followed by a number, purely numeric entries, alphanumeric strings without spaces, and two-word combinations. The function then filters the log to identify instances where the resource does not conform to the user-specified format. This helps in identifying the usage of generalized identifiers such as "Doctor" if the user expects a format like "Doctor 1". The output of the function is given in Figure 2. The function itself can be found in Appendix B.1.

```
Sample resource: Nurse 5
Detected format: Word followed by a number
Rows with imprecise resource identifiers:
# A tibble: 4 × 12
   ...1 Patient_visit_nr Activity      Originator
  <dbl>            <dbl> <chr>         <chr>
1    22              523 Clinical exam Doctor
2    23              523 Registration  Clerk
3    41              533 NA            NA
4    42              534 Registration  NA
```

Figure 2: Output for the imprecise resource function.

- **Resource Inconsistency**

*Description and Real-life Example*

This data quality issue involves the violation of the expected resource-related consistency rule. For instance, if there is a rule requiring the same doctor to handle both the clinical examination and treatment evaluation for a case, Table 8 shows an example in which this rule is followed, while Table 9 shows an example in which this rule is violated. The deviation in this scenario is that different doctors are conducting the clinical exam and the treatment evaluation, contrary to the user-defined consistency rule.

Table 8: Illustration of an activity log with consistent resource.

| Patient_visit_nr | Activity | Originator | Start_ts | Complete_ts |
|---|---|---|---|---|
| 536 | Registration | Clerk 9 | 22/11/2017 10:26:41 | 22/11/2017 10:32:56 |
| 536 | Clinical exam | Doctor 1 | 22/11/2017 15:15:40 | 22/11/2017 15:25:02 |
| 536 | Triage | Nurse 27 | 22/11/2017 15:15:40 | 22/11/2017 15:25:01 |
| 536 | Treatment evaluation | Doctor 1 | 23/11/2017 17:32:12 | 23/11/2017 19:01:23 |

Table 9: Illustration of an activity log with inconsistent resource.

| Patient_visit_nr | Activity | Originator | Start_ts | Complete_ts |
|---|---|---|---|---|
| 519 | Triage | Nurse 17 | 21/11/2017 11:59:41 | 21/11/2017 13:32:18 |
| 519 | Clinical exam | *Doctor 4* | 21/11/2017 11:59:41 | 21/11/2017 14:02:34 |
| 519 | Treatment | Nurse 17 | 21/11/2017 14:38:52 | 21/11/2017 14:49:21 |
| 519 | Treatment evaluation | *Doctor 1* | 22/11/2017 11:05:32 | 22/11/2017 11:13:42 |

*Detection Strategy*

The detection strategy is designed to identify specific instances where the consistency rule is violated. It aims to identify cases where different resources are associated with unique activities in the user-defined activity set.

*Developed Function and Output*

The function, `resource_inconsistency`, filters cases where both of the specified activities are present. It further analyses these cases to determine if multiple unique resources are associated with these activities. The output of this function is a tibble displaying case IDs and associated unique resources, which highlights cases of rule violation. For example, in the provided output given in Figure 3, case ID 519 has two different resources, indicating an anomaly in resource allocation according to the established consistency rule. The function itself can be found in Appendix B.2.

```
Cases with resource inconsistency:
# A tibble: 1 × 2
  Patient_visit_nr Originators
             <dbl> <chr>
1              519 Doctor 4, Doctor 1
```

Figure 3: Output for the resource inconsistency function.

- **Resource and Activity Mismatch**

  *Description and Real-life Example*

  This issue arises when an activity is mistakenly associated with a resource. Typically, if a resource is not meant to perform a particular activity, that activity will appear infrequently in logs compared to activities that are central to the resource. For instance, consider a hospital scenario where a doctor is expected to perform surgeries daily but is unlikely ever to be involved in triage procedures. Ideally, the data should show zero instances of the doctor conducting triage, with numerous loggings of surgical procedures. If activities are recorded under a resource with low frequency, it may indicate potential logging errors, such as triage operations being mistakenly logged as part of a doctor's activities.

  *Detection Strategy*

  To detect these anomalies, a frequency analysis is performed, which examines the regularity of a resource's involvement in tasks. The frequency count aims to highlight activities that are potentially mistakenly logged in association with a resource by showing the activities with the lowest occurrence first. The user should then be able to identify activities that potentially do not relate to the associated resource with the aid of the necessary domain knowledge.

  *Developed Function and Output*

  Initially, the function `resource_activity_mismatch` filters resources based on the sample user input, such as "Doctor 1". Following this, the data is grouped by the resource and activity type, and the occurrences of each activity are counted to measure frequency. Finally, the function sorts these activity counts to highlight activities that have lower occurrences. The output of the function, showing the involvements of "Doctor 1", is given in Figure 4. The function itself can be found in Appendix B.3.

```
Resource activities by type:
# A tibble: 3 × 3
  Originator Activity            ActivityCount
  <chr>      <chr>                       <int>
1 Doctor 1   Treatment                       1
2 Doctor 1   Clinical exam                   2
3 Doctor 1   Treatment evaluation            3
```

Figure 4: Output for the resource-activity mismatch function.

### 4.2.2 Timestamp-related Data Quality Issues

This section explores various potential data quality issues related to timestamps, including imprecise timestamps and same timestamp errors.

- **Imprecise Timestamps**

  *Description and Real-life Example*

  The imprecise timestamp issue refers to the lack of timestamp granularity, such as only recording the date or the hour without detailing minutes and seconds. This imprecision could complicate the efforts to understand the sequence of events and carry out performance analysis accurately (Bose et al., 2013). For example, as seen in Table 10, in a hospital setting, precise timestamps would specify the exact second activity starts and completes, such as "21/11/2017 18:30:17" for the completion of a clinical exam. However, the issue of imprecise timestamp arises when it is recorded as "21/11/2017 18:30" or simply "22/11/2017", which results in the loss of information about the precise duration of the activity.

Table 10: Illustration of an activity log with imprecise timestamp.

| Patient_visit_nr | Activity | Originator | Start_ts | Complete_ts |
|---|---|---|---|---|
| 527 | Registration | Clerk 6 | 21/11/2017 18:02:10 | 21/11/2017 18:04:07 |
| 527 | Triage | Nurse 5 | 21/11/2017 18:02:10 | 21/11/2017 18:04:08 |
| 527 | Clinical exam | Doctor 4 | 21/11/2017 18:02:13 | *21/11/2017* |

*Detection Strategy*

The detection strategy involves checking if the timestamps accurately reflect a detailed date and time structure. It identifies imprecise timestamps that deviate from the sample user input, which includes the correct length and composition of day, month, year, hour, minute, and second.

*Developed Function and Output*

The function `timestamp_granularity` is designed to assess and identify imprecise timestamps.

The function checks each logging against a precise datetime format, such as "dd/mm/yyyy hh:mm:ss", provided by the user as an input. If a timestamp fails to match this format, it is flagged as invalid. The function identifies entries with invalid timestamps and lists the activities most commonly associated with this issue. Additionally, it highlights the resources frequently involved in logging imprecise timestamps. The outputs of the function are given in Figure 5, which enables the identification of imprecise timestamps, allowing for focused improvements in data logging. The function itself can be found in Appendix B.4.

```
$InvalidData
# A tibble: 3 x 14
   ...1 Patient_visit_nr Activity      Originator Start_ts            Complete_ts
  <dbl>            <dbl> <chr>         <chr>      <chr>               <chr>
1     4              512 Clinical exam Doctor 7   11/2017 11:27:12    20/11/2017 11:33:57
2    28              525 Triage        Nurse 5    21/11/2017 17:04    21/11/2017 17:06:08
3    32              527 Clinical exam Doctor 4   21/11/2017 18:02:13 21/11/2017
```

(a)

```
$MostFrequentActivities
       Activity Count
1 Clinical exam     1
2        Triage     1
```

(b)

```
$MostFrequentOriginators
  Originator Count
1   Doctor 4     1
2   Doctor 7     1
3    Nurse 5     1
```

(c)

Figure 5: Outputs for the imprecise timestamp function. a) The complete data table with all activities that have timestamp granularity issues. b) Most frequent activities logged in an invalid timestamp format. c) Resources who are most frequently associated with an invalid timestamp format.

- **Same Timestamp Issues**

  *Description and Real-life Example*

  When different activity sets commonly share the same timestamp, it may indicate form-based event logging. Form-based events often arise when events are logged through computerized forms and display identical timestamps for events that occurred previously. Such timestamps more likely reflect the moment the form was submitted rather than the precise times at which individual events occurred (Andrews et al., 2018). For instance, as illustrated in Table 11, the same start timestamp for registration, triage, and clinical examination for each patient may imply a form-based logging for these activities.

Table 11: Illustration of an activity log with same timestamp issue.

| Patient_visit_nr | Activity | Originator | Start_ts |
|---|---|---|---|
| 519 | Registration | Clerk 12 | *21/11/2017 11:59:41* |
| 519 | Triage | Nurse 17 | *21/11/2017 11:59:41* |
| 519 | Clinical exam | Doctor 4 | *21/11/2017 11:59:41* |
| 519 | Treatment | Nurse 17 | 21/11/2017 14:38:52 |
| 519 | Treatment evaluation | Doctor 1 | 22/11/2017 11:05:32 |
| 523 | Registration | Clerk | *21/11/2017 17:26:30* |
| 523 | Triage | Nurse 27 | *21/11/2017 17:26:30* |
| 523 | Clinical exam | Doctor 4 | *21/11/2017 17:26:30* |

*Detection Strategy*

The detection strategy is designed to identify sets of activities that frequently share the same timestamp in event logs. This method aims to identify instances where activities may have been logged through a standardized form.

*Developed Function and Output*

The developed function, `same_timestamp`, groups the log by case ID and timestamp, filtering the log where distinct activities occur at the same time. It then groups this filtered data based on the activity set and the case ID. Later, it aggregates these groups based on the activity sets by counting the number of unique case IDs related to each activity set. The output, shown in Figure 6, lists the grouped activities, the number of occurrences, and the specific cases involved. This output provides insights into patterns that may suggest form-based logging. The function can be found in Appendix B.5.

```
Same timestamp:
# A tibble: 4 × 3
  Activities                          Occurrences Cases
  <chr>                                     <int> <chr>
1 Registration, Triage                          3 522, 527, 535
2 Registration, Triage, Clinical exam           2 519, 523
3 Treatment, Treatment evaluation               1 532
4 Triage, Clinical exam                         1 536
```

Figure 6: Output for the same timestamp function.

### 4.2.3 Activity-related Data Quality Issues

This section examines activity-related data quality issues caused by inconsistent or incorrect recording of activities.

- **Synonymous Labels and Abbreviations**

  *Description and Real-life Example*

  The issue of synonymous labels or abbreviations arises when different terms, or abbreviations are used to describe what are essentially the same activities within an event log (Suriadi et al., 2017; Vanbrabant et al., 2019). For example, in a hospital setting, different departments might use different terms, such as "Registration" and "Admission", to describe the same process of registering a patient. This inconsistency could mislead the outcomes of process discovery and performance analysis because two or more activities in the log that are actually the same may be mistakenly treated as different (Suriadi et al., 2017).

  *Detection Strategy*

  The strategy focuses on identifying pairs of activity labels that do not appear together in any of the cases (Andrews et al., 2018). The absence of co-occurrence suggests that these labels could potentially represent the same activity but are described using different terms or abbreviations. This is because if a synonymous label is present, it would never pair with the actually intended activity label, and these two would be isolated from each other.

  *Developed Function and Output*

  The function `synonymous_label` first converts all activity labels to lowercase to ensure uniformity. It then groups the data by case ID and summarizes it to list unique activities per case ID. By iterating through all unique activities, it checks for pairs that never appear together in the log. If a pair of activities never co-occur, it is added to a list of potentially synonymous labels. The output of the function, shown in Figure 7, displays activities such as "admission", "registration", and "reg" are identified as non-cooccurring. By flagging these instances, the algorithm guides the user, who then assesses the output with domain-specific knowledge to determine if labels are indeed synonymous. The function is given in Appendix B.6

```
Synonymous label:
     Activity1   Activity2
[1,] "admission" "reg"
[2,] "admission" "registration"
[3,] "admission" "treatment"
[4,] "admission" "treatment evaluation"
[5,] "reg"       "registration"
```

Figure 7: Output for the synonymous label function.

- **Incorrect cases**

  *Description and Real-life Example*

  Incorrect case data quality issues occur when the recorded activities for a particular case violate user-defined rules specifying activity sets that should not co-occur. This type of issue could indicate errors in activity logging and may affect process discovery algorithms (Bose et al., 2013). An illustrative example within a healthcare setting might be that certain treatments cannot be combined with specific other treatments due to potential adverse interactions. Consequently, if a patient's log shows the administration of one such treatment and the logging of another incompatible treatment for the same case, the system should flag this as a potential incorrect case issue.

  *Detection Strategy*

  The detection strategy focuses on identifying activities that are mutually exclusive within the same case. Users input a rule specifying activities that should not occur together, and the functionality identifies if any of these activities occurred together within the same case.

  *Developed Function and Output*

  The function, `incorrect_case`, operates by taking a list of activities that are not expected to co-occur within the same case as a user-defined input, such as "Drug A treatment" and "Drug B treatment". It begins by filtering for cases where either specified activity is present. The log is then grouped by case ID to examine sequences of activities within individual cases. The function then checks if both activities are recorded under a single case ID. If such a condition is met, the case number is noted, and a list of cases is displayed in the output of the function as shown in Figure 8. The function is given in Appendix B.7.

```
Rows with incorrect cases:
[1] 541
```

Figure 8: Output for the incorrect case function.

# 5 Demonstration

The functionality extensions to DaQAPO are observed to be effective with the synthetic activity log used in a hospital setting. However, to check the generalizability of the functionalities, it is essential to test with publicly available data. For this purpose, the academic Dutch hospital event log is chosen (van Dongen, 2011). This event log pertains to tests or procedures performed on patients in a gynecology department and contains 150,291 events across various cases. The event log also contains attributes such as timestamps and transaction types (start, complete). Additionally, each activity is associated with a producer code, which identifies the resource involved. For the purpose of this demonstration, the function is slightly modified to match the column names in the event log.

*Resource-related:* `resource_inconsistency`

The `resource_inconsistency` function checks if different producer codes have applied the given activities within a single case. The input to the function includes a set of activities ("aanname laboratoriumonderzoek" and "ordertarief"). The function filters the event log to include only the cases where the activities defined by the user are present. It then checks whether the specified activities were performed by different producer codes within the same case. The output of the function is given in Figure 9. The output remains capable of verifying resource inconsistency. It is important to note that adequate domain knowledge is essential for providing appropriate inputs that yield relevant outputs.

```
Cases with resource inconsistency:
# A tibble: 540 × 2
   case_id  Originators
   <chr>    <chr>
 1 00000000 CRLA, LBAC
 2 00000001 CRLA, LBAC
 3 00000004 CRLA, CRLE, LBAC, CRPO
 4 00000005 CRLA, LBAC
 5 00000007 CRLA, LBAC, SLVE
 6 00000008 CRLA, LBAC, CRLE
 7 00000010 CRLA, CRLE
 8 00000011 CRLA, CRLE
 9 00000012 CRLA, LBAC, CRPO, BLOB
10 00000013 CRLA, LBAC
# i 530 more rows
```

Figure 9: Output for resource inconsistency function on academic Dutch hospital event log.

*Resource-related:* `resource_activity_mismatch`

The resource activity mismatch function requires a resource as sample user input. In the case of the public event log, this resource is chosen as the one with "CHE2" as the producer code. The function output given in Figure 10 reveals that it is applicable to this public event log. The output remains relevant, and it guides the user to assess whether an activity is associated with the resource or not. For example, based on the current output, it could be argued that activities in rows 1 to 6 occurred only once and may not be relevant to the specified resource "CHE2".

```
[1] "Resource activities by type:"
# A tibble: 57 × 3
   producer_code activity                                ActivityCount
   <chr>         <fct>                                           <int>
 1 CHE2          antist. tegen hiv-e of hiv-c                        1
 2 CHE2          high sensitive crp -hscrp- nefelometrisc            1
 3 CHE2          immunoglobuline g                   igg             1
 4 CHE2          immunoglobuline m                   igm             1
 5 CHE2          ph-meting                                           1
 6 CHE2          prostaatspecifiek antigeen -psa- mbv imm            1
 7 CHE2          immunoglobuline a                   iga             2
 8 CHE2          ammoniak - spoed                                    3
 9 CHE2          ethanol -enzymatisch of gaschromatografi            3
10 CHE2          alfa-amylase                                        4
11 CHE2          cholesterol hdl                                     5
12 CHE2          cholesterol totaal                                  6
13 CHE2          creatine fosfokinase mb - spoed                     6
14 CHE2          lipase                                              6
15 CHE2          digoxine                                            7
16 CHE2          ca-19.9 tumormarker                                 8
17 CHE2          melkzuur enzymatisch                                8
18 CHE2          ammoniak                                            9
19 CHE2          ca 15.3 - tumormarker mbv meia                      9
20 CHE2          alfa amylase - spoed                               14
21 CHE2          foliumzuur mbv radioisotopen                       14
22 CHE2          ferritine kwn mbv chemieluminescentie              15
23 CHE2          troponine-t mbv elisa                              17
24 CHE2          bicarbonaat                                        19
25 CHE2          totaal eiwit colorimetrisch - spoed                19
26 CHE2          melkzuur enzymatisch - spoed                       21
27 CHE2          transferrine mbv ics                               21
28 CHE2          triglyceriden - enzymatisch                        22
29 CHE2          haptoglobine                                       23
30 CHE2          ijzer                                              24
31 CHE2          cea - tumormarker mbv meia                        138
32 CHE2          bilirubine kwantitatief totaal of direct          139
33 CHE2          sgpt alat kinetisch - spoed                       153
34 CHE2          sgot asat kinetisch - spoed                       167
35 CHE2          chloride                                          176
36 CHE2          ureum - spoed                                     178
37 CHE2          fosfaat                                           282
38 CHE2          ca-125 mbv meia                                   343
39 CHE2          magnesium  -aas                                   454
40 CHE2          natrium - vlamfotometrisch - spoed                599
41 CHE2          kalium vlamfotometrisch - spoed                   602
42 CHE2          creatinine - spoed                                644
43 CHE2          crp c-reactief proteine                           729
44 CHE2          bilirubine -geconjugeerd                          911
45 CHE2          glucose                                          1111
```

Figure 10: Some parts of the output for resource-activity mismatch function on academic Dutch hospital event log.

*Resource-related:* `imprecise_resource`

The `imprecise_resource` function checks the granularity of the resource to determine if the input data is specific enough based on a sample input provided by the user. In this event log, producer codes are strings like "CRLA" or "CHE2." The function uses the sample input to check the expected format and verifies whether the producer code matches this format. The output is shown in Figure 11, where the function checks the event log and finds no errors. Using the sample input to determine the expected format ensures that the function is fully generalizable and applicable to any event log.

```
Sample originator: CRLA
Detected format: Alphanumeric without spaces
[1] "Rows with imprecise resource identifiers:"
# A tibble: 0 × 98
```

Figure 11: Output for imprecise resource function on academic Dutch hospital event log.

*Time-related:* `same_timestamp`

The `same_timestamp` function detects which activities commonly share the same timestamp, helping users identify potential form-based logging practices as shown in Figure 12. The `same_timestamp` function is fully applicable to the Dutch academic hospital event log. The most frequently occurring set of activities in the output suggests the possibility of form-based logging.

```
[1] "Same timestamp:"
# A tibble: 645 × 3
   Activity_Group                                                                  Occurrences Cases
   <chr>                                                                                 <int> <chr>
 1 vervolgconsult poliklinisch, administratief tarief       - eerste pol                   182 00000021, 00000034, 00000035,
 2 1e consult poliklinisch, administratief tarief       - eerste pol                        16 00000102, 00000553, 00000699,
 3 verlosk.-gynaec. korte kaart kosten-out, 1e consult poliklinisch, administratief tarief       - e...     5 00000151, 00000239, 00000264,
 4 verlosk.-gynaec. korte kaart kosten-out, vervolgconsult poliklinisch, administratief tarief      ...     5 00000146, 00000392, 00000407,
 5 thorax, 1e consult poliklinisch, administratief tarief       - eerste pol                3 00000983, 00000992, 00001047
 6 verlosk.-gynaec. korte kaart kosten-out, vervolgconsult poliklinisch, administratief tarief      ...     3 00000141, 00000196, 00000442
 7 vervolgconsult poliklinisch, administratief tarief       - eerste pol, histologisch onderzoek - b...     3 00000133, 00000173, 00000751
 8 aanname laboratoriumonderzoek, cea - tumormarker mbv meia, ca-125 mbv meia, ordertarief, vervolgc...     2 00000217, 00000874
 9 aanname laboratoriumonderzoek, cea - tumormarker mbv meia, ca-125 mbv meia, ordertarief, vervolgc...     2 00000039, 00000454
10 aanname laboratoriumonderzoek, ordertarief, vervolgconsult poliklinisch                  2 00000183, 00000570
# ℹ 635 more rows
```

Figure 12: Some parts of the output for same timestamp function on academic Dutch hospital event log.

*Time-related:* `imprecise_timestamp`

The `imprecise_timestamp` function evaluates the granularity of timestamps to ensure it meets the required level of precision. The function filters the given public event log to identify entries where the timestamp granularity does not match the expected format, which is determined based on a sample input provided by the user. This expected format includes a standard such as "YYYY-MM-DD HH:MM:SS". The output of the function, shown in Figure 13, shows that the `imprecise_timestamp` function remains fully generalizable and applicable independently of the event log.

```
# A tibble: 63,260 × 4
   case_id  activity                                    lifecycle timestamp
   <chr>    <fct>                                       <fct>     <chr>
 1 00000000 1e consult poliklinisch                     complete  2005-01-03
 2 00000000 administratief tarief       - eerste pol    complete  2005-01-03
 3 00000000 verlosk.-gynaec. korte kaart kosten-out     complete  2005-01-05
 4 00000000 echografie  - genitalia interna             complete  2005-01-05
 5 00000000 1e consult poliklinisch                     complete  2005-01-05
 6 00000000 administratief tarief       - eerste pol    complete  2005-01-05
 7 00000000 simulator - gebruik voor aanvang megavol    complete  2005-01-24
 8 00000000 behandeltijd - eenheid t3 - megavolt        complete  2005-01-31
 9 00000000 teletherapie - megavolt fotonen bestrali    complete  2005-01-31
10 00000000 aanname laboratoriumonderzoek               complete  2005-02-15
# i 63,250 more rows
```

(a)

```
$MostFrequentActivities
                                activity Count
1              aanname laboratoriumonderzoek  6550
2   ligdagen - alle spec.beh.kinderg.-reval.  4561
3               190205 klasse 3b       a205  3880
4                                ordertarief  3855
5               190101 bovenreg.toesl.  a101  2652
6                vervolgconsult poliklinisch  2299
7                    kalium potentiometrisch  1945
8                   natrium vlamfotometrisch  1930
9                hemoglobine foto-elektrisch  1885
10                              creatinine  1812
11           leukocyten tellen elektronisch  1369
12         trombocyten tellen - elektronisch  1226
13         differentiele telling automatisch  1087
14 administratief tarief       - eerste pol   961
15                                glucose   910
16              190021 klinische opname a002   887
17                                calcium   872
18                                  ureum   753
19            bloedgroep abo en rhesusfactor   676
20       kruisproef volledig -drie methoden-   676
21 rhesusfactor d - centrifugeermethode - e   676
22                     sgpt - alat kinetisch   665
23                     sgot - asat kinetisch   664
24         screening antistoffen erytrocyten   660
25                                albumine   655
```

(b)

```
$MostFrequentResources
   producer_code Count
1           CHE2 14004
2           H5ZU 12506
3           CRLA  9076
4           HAEM  6719
5           CRLE  4895
6           BLOB  3418
7           SGNA  2721
8           LBAC  1414
9           SIOG   898
10          LVPT   877
11          CHE1   620
12          SRTH   550
13          SRA6   500
14          F5NO   435
15          CRPO   428
16          OKU1   418
17          SRA1   402
18          URIN   374
19          STOL   367
```

(c)

Figure 13: Some parts of the output for the function that detects issues with timestamp granularity. a) Activities that have timestamp granularity issues. b) Most frequent activities logged in an invalid timestamp format. c) Resources who are most frequently associated with an invalid timestamp format.

*Activity related:* `incorrect_case`

The `incorrect_case` function detects activities that should not co-occur within the same case. In the case of Dutch academic hospital event log, 'bovenbeen' and 'onderbeen' are chosen. These are then given as input to the `incorrect_case` function, which identifies the rows where these two activities co-occur as given in Figure 14. Similar to `resource_inconsistency`, this function also requires domain knowledge to provide relevant input.

```
[1] "Rows with incorrect cases:"
[1] "00000013"
```

Figure 14: Output for incorrect case function on academic Dutch hospital event log.

*Activity related:* `synonymous_labels`

The `synonymous_labels` function identifies pairs of activities that do not appear together within a case. The output of the function, as shown in Figure 15, reveals numerous entries in the event log that are potentially synonymous. This aligns with existing literature, which suggests that the Dutch hospital data does not follow a strict format in the naming of treatments, resulting in a large number of different names for identical tasks (Lopes & Ferreira, 2019). The function highlights these inconsistencies, providing a list of potential synonyms for user evaluation. The final determination of synonymity relies on the user's domain knowledge, which is necessary to confirm whether the detected labels are truly synonymous. Despite this requirement, the `synonymous_labels` function remains fully generalizable and applicable to this public event log. It effectively provides the necessary output for the user to review and standardize activity names.

| | Activity1 | Activity2 |
|---|---|---|
| [1,] | "1-25-oh2-vitamine d mbv hplc" | "17-hydroxyprogesteron mbv ria" |
| [2,] | "1-25-oh2-vitamine d mbv hplc" | "190034 afwezigheidsdag a006" |
| [3,] | "1-25-oh2-vitamine d mbv hplc" | "190055 zware dagverpleging a009" |
| [4,] | "1-25-oh2-vitamine d mbv hplc" | "190101 bovenreg.toesl. a101" |
| [5,] | "1-25-oh2-vitamine d mbv hplc" | "190204 klasse 3a a204" |
| [6,] | "1-25-oh2-vitamine d mbv hplc" | "339849d toegangschir- c.v.d.-catheter v." |
| [7,] | "1-25-oh2-vitamine d mbv hplc" | "abortus - overige technieken" |
| [8,] | "1-25-oh2-vitamine d mbv hplc" | "acth mbv radioisotopen" |
| [9,] | "1-25-oh2-vitamine d mbv hplc" | "afereseplasma gesplitst fq1 en fq2" |
| [10,] | "1-25-oh2-vitamine d mbv hplc" | "albumine - spoed" |
| [11,] | "1-25-oh2-vitamine d mbv hplc" | "albumineratio" |
| [12,] | "1-25-oh2-vitamine d mbv hplc" | "alfa-1-foetoproteine" |
| [13,] | "1-25-oh2-vitamine d mbv hplc" | "alfa-amylase" |
| [14,] | "1-25-oh2-vitamine d mbv hplc" | "alfa amylase - spoed" |
| [15,] | "1-25-oh2-vitamine d mbv hplc" | "alkalische fosfatase - kinetisch - spoed" |
| [16,] | "1-25-oh2-vitamine d mbv hplc" | "aluminium" |
| [17,] | "1-25-oh2-vitamine d mbv hplc" | "ammoniak" |
| [18,] | "1-25-oh2-vitamine d mbv hplc" | "ammoniak - spoed" |
| [19,] | "1-25-oh2-vitamine d mbv hplc" | "amylase" |
| [20,] | "1-25-oh2-vitamine d mbv hplc" | "analgesie - epiduraal door anesthesist" |
| [21,] | "1-25-oh2-vitamine d mbv hplc" | "anesthesie - bij behandeling met radium" |
| [22,] | "1-25-oh2-vitamine d mbv hplc" | "anesthesie - bij spec.onderz.en verr.ge" |
| [23,] | "1-25-oh2-vitamine d mbv hplc" | "anesthesie - bij vervoer beademde patie" |
| [24,] | "1-25-oh2-vitamine d mbv hplc" | "anesthesie - epiduraal met bloedpatch" |
| [25,] | "1-25-oh2-vitamine d mbv hplc" | "anti-chlamydia iga mbv elisa" |

Figure 15: Some part of the synonymous labels function in academic Dutch hospital event log.

# 6 Discussion

This study aims to enhance data quality assessments by addressing uncovered data quality issues by DaQAPO's current functionalities, specifically focusing on data quality issues related to resources, timestamps, and activities. The development of these new functionalities provides broader insights into data quality assessment in process mining. These functionalities address imprecisions in resource data, inconsistencies in resource allocation, resource-activity mismatches, precision issues in timestamps, potential form-based event logging issues, and inconsistent activity naming and activity rule violations within cases.

The developed resource-related functionalities build upon the works of (Pika et al., 2017; ter Hofstede et al., 2023; van der Aalst, 2012; Song & van der Aalst, 2008), who emphasized the importance of accurate and precise resource-related data prior to process mining. Additionally, considering the importance of identification of timestamp-related data quality issues prior to process mining (Fischer et al., 2020; Dixit et al., 2018; van der Aalst et al., 2008), and its application to all ProM plugins discussed in this study, imprecise timestamp has been considered as an extension functionality. Furthermore, the functionalities for the same timestamp and synonymous labels are developed with the influence of additional functionalities covered by Andrews et al. (2018). Moreover, the functionality for identification of incorrect cases is influenced by the study by Bose et al. (2013), highlighting that incorrect cases are outliers and may mislead process mining analysis.

The implications of these enhancements could be significant for both academia and industry. For researchers, the improved functionalities may offer more comprehensive tools for studying and improving event log quality. For practitioners, particularly in sectors like healthcare, manufacturing, and finance, these tools could provide practical solutions for ensuring high data quality. The application of the extended DaQAPO functionalities to the Dutch hospital event log demonstrated their potential generalizability and robustness, suggesting utility beyond synthetic data. This demonstration not only validates the extended functionalities but also highlights the practical applications of these enhanced data quality assessments.

The study acknowledges several limitations, including the dependency of the new functionalities on the quality of the sample input formats. Incorrectly formatted input can significantly limit the utility of these functionalities. Furthermore, certain functionalities require domain-specific knowledge to config-

ure inputs and interpret outputs appropriately, which necessitates a good level of understanding of the specific process being assessed to maximize the benefits of the functionalities. Additionally, although the study enhances support for DaQAPO in assessing event log quality, it still does not address all possible data quality issues. These issues include inconsistent timestamp formats, day and month values that have been mistakenly interchanged, and imprecise case and event attributes, which are recorded with low precision, making the data less useful for detailed analysis.

# 7 Conclusion

This thesis presents a systematic approach that follows the design science research methodology to extend the functionalities of DaQAPO. The research identifies several critical data quality issues not currently addressed by DaQAPO, particularly those related to resources, timestamps, and activities. The development of new functionalities in these areas broadens DaQAPO's ability to identify a wider range of data quality issues. These functionalities were applied on a publicly available event log, demonstrating their effectiveness and practical applicability. From a broader perspective, this study is not merely an extension of DaQAPO's functionalities but also a contribution to the understanding of data quality issues and their assessment in process mining.

Future research in data quality assessment for process mining could explore several promising directions. One area is the integration of machine learning techniques to automatically generate domain knowledge required inputs and to help with the interpretation of the outputs. Additionally, designing a new user-friendly interface could make DaQAPO more attractive to organizational users and facilitate its adoption in the industry. This would allow users to provide relevant feedback, guiding the development of a more robust tool.

## References


Andrews, R., Dun, C., Wynn, M., Kratsch, W., Roeglinger, M., & Ter, A. (2020, 02). Quality-informed semi-automated event log generation for process mining. *Decision Support Systems*, *132*, 113265.

Andrews, R., Suriadi, S., Ouyang, C., & Poppe, E. (2018). Towards event log querying for data quality: Let's start with detecting log imperfections. In *Proceedings of on the move to meaningful internet systems. otm 2018 conferences: Confederated international conference* (pp. 116–134).

Bose, R. J. C., Mans, R. S., & Van Der Aalst, W. M. (2013). Wanna improve process mining results? In *Proceedings of 2013 IEEE Symposium on Computational Intelligence and Data Mining* (pp. 127–134).

Chen, Q., Lu, Y., Tam, C. S., & Poon, S. K. (2022). A multi-view framework to detect redundant activity labels for more representative event logs in process mining. *Future Internet*, *14*(6).

Conforti, R., La Rosa, M., ter Hofstede, A. H. M., & Augusto, A. (2020). Automatic repair of same-timestamp errors in business process event logs. *Business Process Management*, 327–345.

Dixit, P. M., Suriadi, S., Andrews, R., Wynn, M. T., ter Hofstede, A. H., Buijs, J. C., & van der Aalst, W. M. (2018). Detection and interactive repair of event ordering imperfection in process logs. *Lecture Notes in Computer Science*, *10816*, 274–290.

Fischer, D. A., Goel, K., Andrews, R., van Dun, C. G. J., Wynn, M. T., & Röglinger, M. (2020). Enhancing event log quality: Detecting and quantifying timestamp imperfections. *Lecture Notes in Computer Science*, *12168*, 309–326.

Huang, Z., Lu, X., & Duan, H. (2012). Resource behavior measure and application in business process management. *Expert Systems with Applications*, *39*(7), 6458-6468.

Kherbouche, M. O., Laga, N., & Masse, P.-A. (2016). Towards a better assessment of event logs quality. In *Proceedings of 2016 IEEE Symposium Series on Computational Intelligence* (pp. 1–8).

Lopes, I. F., & Ferreira, D. R. (2019). A survey of process mining competitions: the bpi challenges 2011–2018. In *Proceedings of Business Process Management Workshops: BPM 2019 International Workshops, Vienna, Austria* (pp. 263–274).

Mans, R. S., van der Aalst, W. M., & Vanwersch, R. J. (2015). Data quality issues. *Process mining in healthcare: evaluating and exploiting operational healthcare processes*, 79–88.

Martin, N. (2021). Data quality in process mining. In C. Fernandez-Llatas (Ed.), *Interactive process mining in healthcare* (pp. 53–79). Heidelberg: Springer.

Martin, N., Van Houdt, G., & Janssenswillen, G. (2022). Daqapo: supporting flexible and fine-grained event log quality assessment. *Expert Systems with Applications*, *191*, 116274.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77.

Pika, A., Leyer, M., Wynn, M. T., Fidge, C. J., ter Hofstede, A. H., & van der Aalst, W. M. (2017). Mining resource profiles from event logs. *ACM Transactions on Management Information Systems (TMIS)*, *8*(1), 1–30.

Sadeghianasl, S., ter Hofstede, A. H., Suriadi, S., & Turkay, S. (2020). Collaborative and interactive detection and repair of activity labels in process event logs. In *Proceedings of 2020 2nd international conference on process mining (icpm)* (pp. 41–48).

Sadeghianasl, S., ter Hofstede, A. H. M., Wynn, M. T., & Suriadi, S. (2019). A contextual approach to detecting synonymous and polluted activity labels in process event logs. In H. Panetto, C. Debruyne, M. Hepp, D. Lewis, C. A. Ardagna, & R. Meersman (Eds.), *Proceedings of On the Move to Meaningful Internet Systems: OTM 2019 Conferences* (pp. 76–94). Cham: Springer International Publishing.

Song, M., & van der Aalst, W. (2008). Towards comprehensive support for organizational mining. *Decision Support Systems*, *46*(1), 300-317.

Suriadi, S., Andrews, R., ter Hofstede, A. H., & Wynn, M. T. (2017). Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Information systems*, *64*, 132–150.

ter Hofstede, A. H., Koschmider, A., Marrella, A., Andrews, R., Fischer, D. A., Sadeghianasl, S., ... Goel, K. (2023). Process-data quality: The true frontier of process mining. *ACM Journal of Data and Information Quality*, *15*(3), 1–21.

Vanbrabant, L., Martin, N., Ramaekers, K., & Braekers, K. (2019). Quality of input data in emergency department simulations: Framework and assessment techniques. *Simulation Modelling Practice and Theory*, *91*, 83–101.

van der Aalst, W. M. (2011). Introduction. In *Process mining: Discovery, conformance and enhancement of business processes* (pp. 1–25). Springer Berlin Heidelberg.

van der Aalst, W. M. (2012). Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems*, *3*, 1-17.

van der Aalst, W. M. (2016). Process mining: The missing link. In *Process mining: data science in action* (pp. 25–52). Springer.

van der Aalst, W. M. (2022a). Foundations of process discovery. In *Process mining handbook* (pp. 37–75). Springer.

van der Aalst, W. M. (2022b). Process mining: a 360 degree overview. In *Process mining handbook* (pp. 3–34). Springer.

van der Aalst, W. M., Adriansyah, A., De Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., ... Buijs, J. (2012). Process mining manifesto. In *Proceedings of Business Process Management Workshops: BPM 2011 International Workshops* (pp. 169–194).

van der Aalst, W. M., Dumas, M., Ouyang, C., Rozinat, A., & Verbeek, E. (2008, 05). Conformance checking of service behavior. *ACM Trans. Internet Techn.*, *8*.

van Dongen, B. (2011). *Real-life event logs - hospital log.* Eindhoven University of Technology. Retrieved from https://data.4tu.nl/articles/_/12716513/1

Verhulst, R. (2016). Evaluating quality of event data within event logs: an extensible framework. *Eindhoven University of Technology: Eindhoven, The Netherlands*.

# Appendices

## A Event log imperfection patterns

Table 12: Summary of event log imperfection patterns (Suriadi et al., 2017)

| Event log imperfection patterns | Explanation |
|---|---|
| Form-based Event Capture | Occurs when multiple events are captured from electronic-based forms at the same time (when a user clicks a 'Save' button), resulting in them having the same timestamp. This can flatten the temporal ordering of events and may also lead to redundant information if the form is later updated. |
| Inadvertent Time Travel | Refers to events recorded with erroneous timestamps, often due to human error or system misinterpretations. Common examples include mixing up day and month in dates or incorrect data entry close to midnight, leading to unreasonable event sequences. |
| Unanchored Event | Happens when the timestamps in logs are not formatted or interpreted correctly, leading to incorrect event orderings. This often arises from discrepancies between the expected and actual formats of timestamps, causing misinterpretation by tools. |
| Scattered Event | Involves events in a log that contain additional information within their attributes, which can be used to derive new events. This pattern indicates that more detailed activities are embedded within broader event records. |
| Elusive Case | Occurs when events in a log cannot be explicitly linked to their respective case identifiers. It is common in logs derived from systems not originally designed to support process-aware activities, making it challenging to define a case for process mining. |
| Scattered Case | Describes situations where key process steps are recorded across different sources and need to be merged to construct a complete picture. This often involves linking records from various systems to form a consolidated event log. |
| Collateral Events | Refers to multiple events referring to a single process step but recorded independently due to multiple system interactions or system programming that triggers auxiliary events. This can create noise and unnecessary complexity in the event log. |
| Polluted Label | Occurs when an attribute value consists of a mix of fixed and variable text. While part of the value is consistent across events, other parts vary, creating seemingly distinct entries that actually refer to the same thing. |
| Distorted Label | Involves minor syntactical differences between attribute values that should be identical, often due to typos or inconsistent data entry. These small differences can lead to treating similar activities as distinct in analysis. |
| Synonymous Labels | Similar to Distorted Label but focuses on semantic rather than syntactic differences. It refers to different attribute values that mean the same thing but are expressed differently due to data entry variations or system discrepancies. |
| Homonymous Label | Deals with the repetition of an activity within the same case where each occurrence has a different meaning based on the context. This can lead to an incomplete understanding of the process if not addressed, as the log treats all occurrences as the same activity. |

# B Codes

## B.1 Imprecise resource

```
imprecise_resource <- function(data_path, sample_resource) {
  df <- read_excel(data_path)

  cat("Sample resource:", sample_resource, "\n")

  # Determine the expected format based on the sample resource
  if (grepl("^[A-Za-z]+\\u00A0+\\d+$", sample_resource)) {
    chosen_format <- "Word followed by a number"
    cat("Detected format:", chosen_format, "\n")
    check_format <- function(x) {
      grepl("^[A-Za-z]+\\u00A0+\\d+$", x)
    }
  } else if (grepl("^\\d+$", sample_resource)) {
    chosen_format <- "Purely numeric"
    cat("Detected format:", chosen_format, "\n")
    expected_length <- nchar(sample_resource)
    check_format <- function(x) {
      nchar(x) == expected_length && grepl("^\\d+$", x)
    }
  } else if (grepl("^[A-Za-z0-9]+$", sample_resource)) {
    chosen_format <- "Alphanumeric without spaces"
    cat("Detected format:", chosen_format, "\n")
    expected_length <- nchar(sample_resource)
```

```r
    check_format <- function(x) {
      nchar(x) == expected_length && grepl("^[A-Za-z0-9]+$", x)
    }
  } else if (grepl("^[A-Za-z]+\\s[A-Za-z]+$", sample_resource)) {
    chosen_format <- "Two words"
    cat("Detected format:", chosen_format, "\n")
    check_format <- function(x) {
      grepl("^[A-Za-z]+\\s[A-Za-z]+$", x)
    }
  } else {
    cat("Failed to detect format for sample resource:", sample_
        resource, "\n")
    stop("The sample resource format is not supported.")
  }


  # Filter rows that do not match the expected format
  rows_with_imprecise_originator <- df %>%
    filter(!sapply(df$Originator, check_format))

  return(rows_with_imprecise_originator)
}
```

## B.2 Resource inconsistency

```
resource_inconsistency <- function(data_path, activities) {
  data <- read_excel(data_path)

  # Filter data to only include rows with specified activities
  data_filtered <- data %>%
    filter(Activity %in% activities)

  # Check for each patient visit if the specified activities are
     performed by different resources
  resource_inconsistency <- data_filtered %>%
    group_by(Patient_visit_nr) %>%
    filter(n_distinct(Activity) == length(activities)) %>%  #
       Ensure all activities are accounted for
    summarise(
      Originators = paste(unique(Originator), collapse = ", "),
      Unique_Originators = n_distinct(Originator)
    ) %>%
    filter(Unique_Originators > 1) %>%  # More than one unique
       resource found
    select(-Unique_Originators) %>%
    ungroup()

  return(resource_inconsistency)
}
```

### B.3 Resource-activity mismatch

```r
resource_activity_mismatch <- function(data_path, resource_input) {

  data <- read_excel(data_path)

  # Filter data for the specified resource
  filtered_data <- data %>%
    filter(Originator == resource_input)

  # Calculate the ActivityCount directly from the filtered data
  resource_activities_by_type <- filtered_data %>%
    group_by(Originator, Activity) %>%
    summarise(ActivityCount = n(), .groups = 'drop') %>%
    arrange(Originator, ActivityCount)  # Sorting in ascending
       order within each group

  return(resource_activities_by_type)
}
```

### B.4 Imprecise Timestamp

```
imprecise_timestamp <- function(data_path, sample_date) {

  data <- read_excel(data_path)

  # Determine the expected length of the sample date
  expected_length <- nchar(sample_date)

  # function to check for timestamp format based on sample date
     length
  check_date_format <- function(date_string) {
    # Check if the original string matches the expected length
    if (nchar(date_string) != expected_length) {
      return(TRUE) # Invalid format
    } else {
      return(FALSE) # Valid format
    }
  }

  # Apply the function to Start_ts and Complete_ts columns
  data$Invalid_Start_ts <- sapply(data$Start_ts, check_date_format)

  # Check for Complete_ts (or complete column) existence and
     validate if it exists
  if ("Complete_ts" %in% colnames(data)) {
    data$Invalid_Complete_ts <- sapply(data$Complete_ts, check_date
```

```
      _format)
24  } else {
25    data$Invalid_Complete_ts <- FALSE  # If Complete_ts does not
        exist, set to FALSE
26  }
27
28  # Filter invalid dates
29  invalid_dates_data <- data[data$Invalid_Start_ts | data$Invalid_
      Complete_ts, ]
30
31  # Analyze most frequent Activities and Originators with invalid
      timestamps
32  most_freq_activities <- sort(table(invalid_dates_data$Activity),
      decreasing = TRUE)
33  most_freq_originators <- sort(table(invalid_dates_data$Originator
      ), decreasing = TRUE)
34
35  # Transform the output for activities and originators to desired
      format
36  transposed_activities <- as.data.frame(stack(most_freq_activities
      ), stringsAsFactors = FALSE)
37  transposed_activities <- transposed_activities[, c("ind", "values
      ")]
38  names(transposed_activities) <- c("Activity", "Count")
39
40  transposed_originators <- as.data.frame(stack(most_freq_
```

```
      originators), stringsAsFactors = FALSE)
41  transposed_originators <- transposed_originators[, c("ind", "
      values")]
42  names(transposed_originators) <- c("Originator", "Count")
43
44
45  # Results
46  list(
47    InvalidData = invalid_dates_data,
48    MostFrequentActivities = transposed_activities,
49    MostFrequentOriginators = transposed_originators
50  )
51 }
```

### B.5 Same Timestamp

```
same_timestamp <- function(data_path) {

  df <- read_excel(data_path)

  # Convert Start_ts and Complete_ts to datetime objects
  df$Start_ts <- ymd_hms(df$Start_ts, quiet = TRUE)
  df$Complete_ts <- ymd_hms(df$Complete_ts, quiet = TRUE)

  # Filter for patient visits with more than one activity
  df_multiple <- df %>%
    group_by(Patient_visit_nr) %>%
    filter(n() > 1) %>%
    ungroup()

  # Create separate data frames for Start_ts and Complete_ts
  df_start <- df_multiple %>%
    select(Patient_visit_nr, Activity, Timestamp = Start_ts) %>%
    filter(!is.na(Timestamp))

  df_complete <- df_multiple %>%
    select(Patient_visit_nr, Activity, Timestamp = Complete_ts) %>%
    filter(!is.na(Timestamp))

  # Find duplicates within Start_ts
```

```r
df_start_grouped <- df_start %>%
  group_by(Patient_visit_nr, Timestamp) %>%
  filter(n() > 1) %>%  # Ensure there are multiple activities sharing the same timestamp
  summarise(Activities = toString(unique(Activity)), .groups = 'drop')

# Find duplicates within Complete_ts
df_complete_grouped <- df_complete %>%
  group_by(Patient_visit_nr, Timestamp) %>%
  filter(n() > 1) %>%  # Ensure there are multiple activities sharing the same timestamp
  summarise(Activities = toString(unique(Activity)), .groups = 'drop')

# Combine the results
df_grouped <- bind_rows(df_start_grouped, df_complete_grouped)

# Create the x variable with patient visit numbers and activity groups
x <- df_grouped %>%
  group_by(Patient_visit_nr) %>%
  summarise(Activity_Group = toString(unique(Activities)), .groups = 'drop')

# Aggregate activities and count unique case numbers
```

```
46   result <- x %>%
47     group_by(Activity_Group) %>%
48     summarise(Occurrences = n(),
49               Cases = toString(unique(Patient_visit_nr)),
50               .groups = 'drop') %>%
51     arrange(desc(Occurrences))  # Sorting by Occurrences in
         descending order
52
53   # Display the x variable and the result
54   return(result)
55 }
```

### B.6 Synonymous label

```
# Function to find non-cooccurring activity pairs
synonymous_label <- function(data_path) {

  data <- read_excel(data_path
  )
  data$Activity <- tolower(data$Activity)

  grouped_data <- data %>%
    group_by(Patient_visit_nr) %>%
    summarise(Activities = list(unique(Activity))) %>%
    ungroup()

  unique_activities <- sort(unique(unlist(grouped_data$Activities))
     )
  non_cooccurring_pairs <- vector("list", length = 0)

  for (i in 1:(length(unique_activities) - 1)) {
    for (j in (i + 1):length(unique_activities)) {
      activity1 <- unique_activities[i]
      activity2 <- unique_activities[j]
      cooccur <- FALSE

      for (k in 1:nrow(grouped_data)) {
        if (all(c(activity1, activity2) %in% grouped_data$
            Activities[[k]])) {
```

```
24            cooccur <- TRUE
25            break
26          }
27        }
28
29        if (!cooccur) {
30          # Sort the pair to ensure unique combinations
31          pair <- sort(c(activity1, activity2))
32          non_cooccurring_pairs <- c(non_cooccurring_pairs, list(pair
                ))
33        }
34      }
35    }
36
37    # Convert list to dataframe and remove duplicate rows
38    non_cooccurring_pairs_df <- do.call(rbind, non_cooccurring_pairs)
39    non_cooccurring_pairs_df <- unique(non_cooccurring_pairs_df)
40    colnames(non_cooccurring_pairs_df) <- c("Activity1", "Activity2")
41    return(non_cooccurring_pairs_df)
42  }
```

## B.7 Incorrect Case

```
incorrect_case <- function(data_path, rule_activity) {
  dataset <- read_excel(data_path)
  # Filter cases where any of the activities in rule_activity
     occurred
  cases_with_either_activity <- subset(dataset, Activity %in% rule_
     activity)

  cooccurring_cases <- lapply(split(cases_with_either_activity,
     cases_with_either_activity$Patient_visit_nr), function(case_
     data) {
    if (all(rule_activity %in% case_data$Activity)) case_data$
       Patient_visit_nr[1]
  })
  # Clean up the list to remove NULLs and duplicates
  cooccurring_cases <- unname(na.omit(unique(unlist(cooccurring_
     cases))))
  # Print the case numbers
  if (length(cooccurring_cases) > 0) {
    return(cooccurring_cases)
  } else {
    print("No cases with co-occurring specified activities were
       found.")
  }
}
```

### B.8 Main Function

```r
library(dplyr)
library(readxl)
library(lubridate)
library(tidyr)
library(stringr)
library(tibble)
library(knitr)
library(anytime)
library(eventdataR)

source("same_timestamp.R")
source("synonymous_label.R")
source("resource_activity_mismatch.R")
source("resource_inconsistency.R")
source("imprecise_resource.R")
source("incorrect_case.R")
source("imprecise_timestamp.R")
options(width = 400)
# Set working directory to script location
script_directory <- dirname(sys.frame(1)$ofile)
setwd(script_directory)

# Path to the data file
data_path <- 'Hospital Event Log.xlsx'

```

```r
# 1. Same timestamp
events_by_start_ts <- same_timestamp(data_path,"dmy_hms")
cat("Same timestamp:\n")
print(events_by_start_ts, width = Inf)

# 2. Synonymous labels
non_cooccurring_pairs <- synonymous_label(data_path)
cat("Synonymous label:\n")
print(non_cooccurring_pairs)

# 3. Imprecise timestamp
sample_timestamp <- "21/11/2017 11 :22:16"
invalid_dates <- imprecise_timestamp(data_path, sample_timestamp)
cat("Invalid dates:\n")
print(invalid_dates)

# 4. Resource activity mismatch
resource_mismatch <- "Doctor 1"
resource_activities_summary <- resource_activity_mismatch(data_path
    , resource_mismatch)
cat("Resource activities by type:\n")
print(resource_activities_summary, n = 32)

# 5. Resource inconsistency
activities <- c("Clinical exam", "Treatment evaluation")
inconsistent_resource <- resource_inconsistency(data_path,
```

```
    activities)
cat("Cases with resource inconsistency:\n")
print(inconsistent_resource)

# 6. Imprecise resource
sample_resource <- "Nurse 5"
imprecise_resource_rows <- imprecise_resource(data_path, sample_
    resource)
cat("Rows with imprecise resource identifiers:\n")
print(imprecise_resource_rows)

# 7. Incorrect cases
rule_activity <- c("Drug A treatment", "Drug B treatment")
incorrect_cases <- incorrect_case(data_path, rule_activity)
cat("Rows with incorrect cases:\n")
print(incorrect_cases)
```