



Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs



S. Suriadi^{a,*}, R. Andrews^a, A.H.M. ter Hofstede^{a,b}, M.T. Wynn^a

^a Queensland University of Technology (QUT), GPO Box 2434, Brisbane, Australia

^b Eindhoven University of Technology, Eindhoven, The Netherlands

ARTICLE INFO

Keywords:

Process mining
Data mining
Data quality
Event log quality
Patterns
Systematic data pre-processing
Event log preparation

ABSTRACT

Process-oriented data mining (process mining) uses algorithms and data (in the form of event logs) to construct models that aim to provide insights into organisational processes. The quality of the data (both form and content) presented to the modeling algorithms is critical to the success of the process mining exercise. Cleaning event logs to address quality issues prior to conducting a process mining analysis is a necessary, but generally tedious and *ad hoc* task. In this paper we describe a set of data quality issues, distilled from our experiences in conducting process mining analyses, commonly found in process mining event logs or encountered while preparing event logs from raw data sources. We show that patterns are used in a variety of domains as a means for describing commonly encountered problems and solutions. The main contributions of this article are in showing that a patterns-based approach is applicable to documenting commonly encountered event log quality issues, the formulation of a set of components for describing event log quality issues as patterns, and the description of a collection of 11 event log imperfection patterns distilled from our experiences in preparing event logs. We postulate that a systematic approach to using such a pattern repository to identify and repair event log quality issues benefits both the process of preparing an event log and the quality of the resulting event log. The relevance of the pattern-based approach is illustrated via application of the patterns in a case study and through an evaluation by researchers and practitioners in the field.

1. Introduction

In today's information systems, it is not uncommon for the execution of processes to leave historical traces recorded in many forms, including audit trails, system logs, databases, and paper-based records. Both data mining and process mining analytics exploit such historical data to uncover useful information relevant to the domain about the systems and processes from which the historical data is collected. Process mining, as an emerging research discipline, aims at utilising information in *event logs* to discover, monitor and improve processes [45]. Though emerging, process mining has already been conducted in many organisations in a variety of domains. For example, Mans et al. [24] identified 40 scholarly papers that report on applications of process mining (33 focusing on process discovery and 7 on process conformance) in various areas of health care. Process mining has yielded promising outcomes in painting evidence-based, end-to-end views of processes including: the actual manner in which processes were carried out; the (non-)conformance of processes to guidelines/regulations/legislation; performance and bottlenecks; and the identifi-

cation of pain points to facilitate targeted and effective process improvement initiatives.

The truism *garbage-in garbage-out* is as relevant to process mining as it is to other forms of computerised data analysis. In process mining, the 'in' is an event log, so it is important to prepare the event log such that it is fit for the purpose of the analysis. The preparation of event logs necessarily involves cleaning raw data sources from noise with the aim of minimising information loss and producing an event log that is of 'high quality', i.e. the event log is valid in the context of the domain from which the raw data was collected and is valid for the purpose of the analysis (see Fig. 1). Where such is the case, the data itself is not an impediment to the analysis producing a model that agrees with reality.

In this paper we point out that process mining event logs have some unique data quality considerations that can be dealt with through the use of a patterns-based approach. The use of patterns as a means of understanding, and communicating the characteristics of an apparently chaotic domain is a fundamental human behaviour and has been widely applied in diverse areas, including business analysis [13],

* Corresponding author.

E-mail addresses: s.suriadi@qut.edu.au (S. Suriadi), r.andrews@qut.edu.au (R. Andrews), a.terhofstede@qut.edu.au (A.H.M. ter Hofstede), m.wynn@qut.edu.au (M.T. Wynn).

¹ Some of this work was done while the author was employed at Massey University, New Zealand.

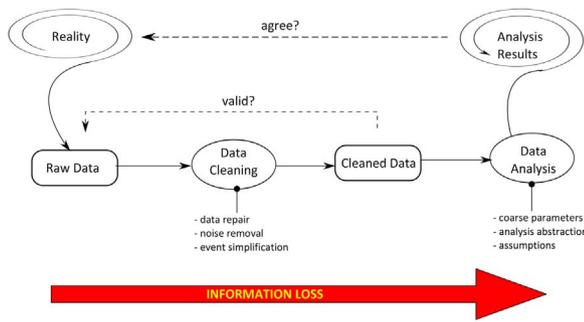


Fig. 1. Garbage-in Garbage-out: the need for data and results validation.

software design [15,52] and systems security [22]. The benefit of using a patterns-based approach is that it allows core solutions to recurring problems to be developed and, most importantly, applied over and over again [3]. The main contributions of this article are showing that a patterns-based approach is applicable to documenting commonly encountered event log quality issues, the formulation of a set of components for describing event log quality issues as patterns, and the description of a collection of *event log imperfection patterns* distilled from our own experiences in preparing event logs that capture some specific, commonly-encountered event log data quality issues. The impact of the presence in an event log of each imperfection pattern in terms of the data quality dimension(s) most severely impacted is also explained. We show how checking for the ‘signatures’ of these patterns in an event log can reveal the existence of the associated quality issues and inform appropriate remedial action(s). The patterns thus form a knowledge repository facilitating a systematic approach to event log preparation that is *independent* of the *type* of analysis to be performed with benefits to both the process of preparing the log and the quality of the resulting log.

Whilst advocating for the use of patterns, we, like other pattern collection authors [15,8,14], acknowledge that the pattern collection described in this paper cannot be deemed to be complete (i.e. in our case, to offer coverage of all possible quality issues that may afflict an event log). Indeed, it is not even possible to determine the degree of coverage afforded by this pattern collection to the spectrum of possible problems that may afflict an event log, as it not possible to absolutely determine the complete set of such problems. Thus, even if none of the patterns manifest in a log, it is not certain that the log is 100% correct. We maintain however, that the patterns provide a way to check for some commonly occurring, and from a process mining perspective, high priority issues, which, when systematically and routinely addressed, provides a ‘ground level’ quality assurance and allows researchers and practitioners to devote effort to uncovering domain and log specific quality issues. As more patterns are included in the collection, the ‘ground level’ quality will naturally rise. Lastly, we refer to Martin Fowler, another (highly cited) patterns author, who in [14], with regard to completeness, states in relation to his Enterprise Application Architecture patterns collection: “*This pattern collection is by no means a comprehensive guide to enterprise architecture development. My test for this book is not whether it’s complete, but merely if it’s useful.*”

The remainder of the paper is organised as follows. In Section 2 we consider event log basics, patterns basics and some previously published views on data quality from an information systems and process mining point of view. In Section 3 we discuss work related to data cleaning including some process mining-specific approaches. In Section 4 we describe a process mining-focused data quality framework [6] that may be used to classify quality issues of a process mining event log. Sections 5 and 6 contain the main contributions of this paper. In Section 5 we propose a set of 11 event log imperfection patterns distilled from our own experiences in preparing event logs. Examples provided in Section 6 illustrate the use of the patterns via their

application in a case study. Section 7 presents an evaluation, by practitioners in the field, of their recognition of, and perceived importance and usefulness of the patterns.

2. Background

Event logs used in process mining have a structure designed to allow representation of key attributes of events that occurred over multiple executions of a given process. These event logs may then be presented to a process mining algorithm with the aim of analysing and mapping the process (discovery), determining the extent to which actual execution of the process agrees with the intended execution (conformance) or highlighting areas for improvement in the process (enhancement) [44]. Irrespective of the specific type of analysis, it is true that the success of the analysis depends on the quality of the data used in the analysis. In this section we describe the characteristics of an event log and, with the notion of *garbage-in garbage-out* in mind, review some existing notions of data quality. We also describe patterns as a mechanism for describing commonly encountered problems and solutions, pattern languages as a structured method for describing patterns and discuss previous applications of patterns in different domain settings.

2.1. Event log basics

An event log suitable for process mining contains data related to a *single process*. The event log consists of a set of cases (or traces) and is constructed as a case table in multiple record-case format (see Fig. 2). Each case consists of the sequence of events carried out in a single execution of a process (process instance). Each unique sequence of events from the beginning to the end of a process instance is referred to as a *variant*. Each case/trace belongs to exactly one variant. A variant may describe one or more cases/traces. Cases within a log are uniquely identified by a case identifier. Irrespective of the type of process mining analysis undertaken, each event is minimally characterised by a case identifier which informs the case to which the event relates, and an ‘activity’ label describing the related ‘action’. Many process mining analyses, e.g. *process discovery*, require an attribute that allows ordering of events, e.g. a timestamp describing when the event occurred. Other types of analysis require that the log contains relevant supporting attributes. For instance, it is not possible to discover the social network of resources contributing to the process unless event data is enriched with resource information.

To conduct a process mining analysis, an event log needs to contain, at minimum, enough information such that every activity can be ascribed to a case and can be ordered via, for example the timestamp. Often, the final event log is made up of data captured in multiple raw source logs extracted from different systems used in supporting the process being analysed. In this paper, we draw a clear distinction between the final, *cleaned* event log and the contributing *raw* source logs.

Definition 2.1 (Attribute, Event, Event Log). Let \mathcal{E} be the *event universe*, i.e. the set of all possible event identifiers. Events may be characterised by various *attributes*, e.g. an event may belong to a

CaseID	Attributes Activity	Timestamp	Resource	Location
1	Present at ED	2014-10-03 07:54	Jason	Emergency
1	Triage request	2014-10-03 07:57	Jason	Emergency
1	Triage	2014-10-03 08:03	Susan	Emergency
1	Medical assign	2014-10-03 08:10	Jason	Emergency
1	Blood tests	2014-10-03 09:34	Sarah	Laboratory
1	Admit to hospital	2014-10-03 10:02	George	Ward
2	Present at ED	2014-10-03 08:12	Jason	Emergency
2	Triage request	2014-10-03 08:16	Jason	Emergency
2	Triage	2014-10-03 08:20	Ross	Emergency
2	Medical assign	2014-10-03 08:30	Jason	Emergency
2	Discharge to home	2014-10-03 08:50	Jason	Emergency

Fig. 2. Event log fragment.

particular case, have a timestamp, correspond to an activity, and can be executed by a particular person.

Let $AN = \{a_1, a_2, \dots, a_n\}$ be a set of all possible attribute names. For each attribute $a_i \in AN$ ($1 \leq i \leq n$), \mathcal{D}_{a_i} is its domain, i.e. the set of all possible values for the attribute a_i .

For any event $e \in \mathcal{E}$ and an attribute name $a \in AN$: $\#_a(e) \in \mathcal{D}_a$ is the value of attribute named a for event e . If an event e does not have an attribute named a , then $\#_a(e) = \perp$ (null value).

Let \mathcal{D}_{id} be the set of event identifiers, \mathcal{D}_{case} be the set of case identifiers, \mathcal{D}_{act} be the set of activity names, \mathcal{D}_{time} be the set of possible timestamps, and \mathcal{D}_{res} be the set of resource identifiers. For each event $e \in \mathcal{E}$, we define a number of standard attributes:

- $\#_{id}(e) \in \mathcal{D}_{id}$ is the event identifier of e ;
- $\#_{case}(e) \in \mathcal{D}_{case}$ is the case identifier of e ;
- $\#_{act}(e) \in \mathcal{D}_{act}$ is the activity name of e ;
- $\#_{time}(e) \in \mathcal{D}_{time}$ is the timestamp of e ; and
- $\#_{res}(e) \in \mathcal{D}_{res}$ is the resource who triggered the occurrence of e .

An event log $\mathcal{L} \subseteq \mathcal{E}$ is a set of events. This definition of an event log allows the log to be viewed as a table, thus allowing the application of relational algebra to the log.

2.2. Data quality dimensions

In order to assess the quality of an event log, it is necessary to define a model of quality suitable to process mining. The literature contains many data quality models, most of which relate to information systems in general and do not deal directly with specific requirements of process mining. Data quality is frequently discussed in the literature as a multi-dimensional concept. Wand and Yang [48] summarise the results of a literature review [49] according to the most cited quality dimensions. They further categorise these dimensions according to whether they relate to the ‘external’ view of an information system (which is concerned with the purpose of the information system) or the ‘internal’ view (which addresses the construction and operation of the information system necessary to meet the external view needs as defined by a set of requirements and business rules). Their quality dimensions are: *Completeness* (all lawful states in the real-world system can be represented in the information system); *Unambiguity* (no two states of the real-world system should be mapped into the same state in the information system); *Meaningfulness* (all states in the information system can be mapped back to a state of the real-world system); and *Correctness* (during operation, all real-world states are mapped to the correct information system state, i.e. mapping the information system state back to the real-world system results in the ‘correct’ real-world system state).

Batini and Scannapieco [5] define at least the following dimensions for data quality: *Accuracy* (a measure of the closeness between a recorded value and the real-life phenomenon that the recorded value aims to represent). Accuracy is further broken down into (i) *Syntactic accuracy* the closeness of a recorded value to the elements of the corresponding definition domain, and (ii) *Semantic accuracy* the closeness between a recorded value and the true value. Here semantic accuracy corresponds with the concept of *correctness*; *Completeness* (“the extent to which data are of sufficient breadth, depth and scope for the task at hand” [50]); *Consistency* (“captures the violation of semantic rules defined over (a set of) data items”); *Currency*, *Timeliness & Volatility* (dimensions concerned with “change and updates to data in time”); and *Synchronisation between different time series* (“concerns proper integration of data having different timestamps”).

The ISO/IEC 25012 standard [1] aims to define a “general data quality model for data retained in a structured format within a computer system”. The standard defines data quality as “the degree to which the characteristics of data satisfy stated and implied needs

when used under specified conditions” and categorises quality attributes into fifteen characteristics from two different perspectives: inherent (the data itself) and system dependent (the ways in which data is “reached and preserved within a computer system”).

Some of the characteristics defined in the standard are: *Accuracy* (the degree to which data correctly represent the true value of the intended attributes of a concept or event in a specific context of use). The standard also considers accuracy from syntactic and semantic points of view; *Completeness* (data have values for all expected attributes and related entity instances in a specific context of use); *Consistency* (data are free from contradiction and are coherent with other data in a specific context of use); *Credibility* (users regard data as believable/reliable in a specific context of use); and many others.

2.3. Patterns basics

In the late 1970s, Christopher Alexander's *The Timeless Way of Building* [2] and *A Pattern Language: Towns, Buildings, Construction* [3] introduced the notion of a pattern as a means of describing problems (and their solutions) commonly faced in our built environment. In [3], Alexander provides the rationale for using patterns as “Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over without ever doing it the same way twice”.

Since then, Alexander's pattern concept has made its way into many disciplines. The highly cited work by the so-called “Gang of Four” [15] is credited with initiating interest in design patterns for use in object-oriented programming. In Fowler's works on analysis patterns [13] and enterprise architecture patterns [14], the author makes the point that patterns are not contrived by “academic invention” but rather discovered from experience. This point is reinforced by Grady Booch in the foreword to [4]. Also in [4], the authors cite the *Rule of Three* where a solution becomes a pattern after it has been verified in three different systems.

Our principal aims in writing this article are manifold. Firstly, we wish to propose *patterns* as a novel way of considering data quality issues that may be encountered in process mining event logs. Secondly, we wish to discuss the notion, at a conceptual level, in such a way that it is accessible to a broad cross-section of the process mining community. We also aim to raise awareness among practitioners of the patterns-based approach to event log quality issues with the hope that it would stimulate practitioners to verify the existence of the patterns in their own data sets. In [46], the authors state that a pattern description needs to strike a balance between generality and precision. A textual/graphical representation is a convenient, language and implementation independent representation that promotes awareness and makes patterns easily understandable without overwhelming the reader with technical considerations. Further, such a presentation style does not impose strict semantics on the patterns, (thereby limiting their generality). To the best of our knowledge, this is the first work describing the application of patterns to event log quality issues. As such, and in keeping with other germinal pattern articles [15,13,47] we felt that a discursive, rather than formalised style was better suited to achieving these particular ends. Other widely read pattern collections that use informal, component based descriptions can be found in [3,15,13,14,19,47].

Our final aim for this article was to show that patterns could form the basis of a systematic approach to cleaning event logs. As Fowler says, “conceptual models are only useful to software engineers if they can see how to implement them” [13]. A mathematical and logical formulation can address issues of ambiguity in textual/graphical descriptions and assist researchers and designers reason about solution issues [42,52]. Accordingly, for each pattern in the collection, we include a ‘lightweight’ formal specification. Going beyond this level of specification is, we believe, outside the scope and intent of this paper.

3. Related work

In this section, we review previous work relating to data quality for process mining analyses. In particular, we focus our related work section on advances within the process mining community that (i) raise the issue of event log quality, and (ii) propose methods to overcome event log quality problems (including noise removal). For completeness, we also, briefly, bring into the discussion existing work within the data mining community in general and argue why such approaches are not applicable to address the needs of process mining.

The issue of event log quality has been raised quite some time ago. The Process Mining Manifesto, [45], defines a star rating (1-star to 5-star) of the maturity of event logs where maturity refers to readiness for process mining analysis. The authors state that event logs rated as 3, 4 or 5-star are suitable for process mining analysis, while 1- and 2-star rated logs are probably not suited for use in a process mining analysis. A 5-star rated log (excellent quality, trustworthy and complete) is characterised as being recorded automatically, systematically and reliably with all events and their attributes having clear semantics. At the other end of the scale, 1-star rated logs (poor quality) are characterised as having events that do not correspond to reality and may be incomplete (i.e. missing events). Such logs often result from manual recording of data. With this rating system in mind, we contend that the data imperfection patterns proposed in this article capture some of the issues typically found in lower-rated event logs. By applying the remedies for each data imperfection pattern (also proposed in this article), we hope to allow the community to be able to improve the quality of event logs in general.

Bose et al. [6] identify four broad categories of issues affecting process mining event log quality: *Missing data* (where different kinds of data are missing from the event log); *Incorrect data* (data is provided, but, based on contextual information, is logged incorrectly); *Imprecise data* (where logged entries are too coarse); and *Irrelevant data* (where logged entries are irrelevant as is, but may be used to derive relevant entities through filtering, abstraction or some other pre-processing). The authors then show where each of these issues may manifest themselves in the various entities of an event log. A distinction between our work and the work proposed in [6] is that our event log imperfection patterns, while more or less falling within Bose et al.'s four broad categories of data quality issues, explore and highlight event log problems at a more detailed level. In particular, we propose a *concrete manifestation* of typical event log quality issues that are commonly found in industry-based logs informed in large part through our experiences in conducting analyses in healthcare [35,40] and insurance [41] domains.

Mans et al. [25] propose a process mining data spectrum to classify event data generated by different types of systems generally found in a Hospital Information System (HIS). The authors observe that a HIS comprises of a mixture of information processing systems designed for different purposes and used in different ways. Some systems, such as *administrative systems*, record services delivered to patients, often for billing purposes. However, the issue here is that data is often entered manually and is generally performed some time after the service has been delivered to the patient. While coarse-grained and imprecise recording of timestamps (often at the day level granularity) is not an issue for administrative purposes, it is mostly *too* coarse for process mining analysis purposes. Nevertheless, the knowledge of how such systems are being used forms the basis of some of our event log imperfection patterns and their remedies (e.g. the *collateral events* and *scattered events* patterns detailed in Section 5). Other systems, such as *medical devices*, automatically record information about the state of the devices as well as the task being performed in a fine-grained manner (milliseconds). Event logs from such devices are, on the other hand, often too fine-grained for process mining analyses as there could be hundreds if not thousands of events recorded for even a very short period of time. Nevertheless, the knowledge of the data provenance in

this case also forms the basis of data imperfection patterns (e.g. the *elusive case* and the *synonymous labels* patterns detailed in Section 5). Overall, the work by Mans et al. [25] describes event log quality as a two-dimensional spectrum with the first dimension concerned about the *level of abstraction of the events* and the second one concerned with the *accuracy of the timestamp*. The latter dimension is divided into three sub-dimensions (i) *granularity*, (ii) *directness of registration* (i.e. the currency of the timestamp recording) and (iii) *correctness*. In this work, the spectrum is used in the context of a case study to elaborate on the data challenges faced in a process mining analysis and whether event data collected from HISs allows for answering questions frequently posed by medical professionals in a process mining analysis. Our data imperfection patterns (proposed in this article) do cut across these dimensions proposed by Mans et al. [25]; however, as stated before, our approach explores the issue of event log quality at a much finer-grained level by proposing concrete manifestations of data issues that are, theoretically at least, semi-automatable in terms of their detection and correction.

Where event logs contain no exceptional data and noise, process mining algorithms can discover accurate models and can extract useful process-related insights. However, the presence of exceptional data (noise) in the event log leads to unnecessarily complex discovered models that do not accurately reflect the underlying process. In [18], in describing various types of noise that may be encountered in an event log, the author firstly refers to work in [51] on *syntactic* noise before introducing the notion of *semantic* noise. Here syntactic noise arises from errors in logging and includes such things as missing head, tail or episodes of traces and the inclusion of so-called *alien events* within traces. Semantic noise, the author contends, is introduced to the log on purpose through either decisions made about which events are logged (resulting in noise classed as *imbalance between event importance and granularity*), or decisions made about *customisations* to the supporting information system or *modifications* made to the relevant process model. The author indicates that syntactic noise may be alleviated through careful extraction and pre-processing of log data, customisation noise may be addressed through *semantic pre-processing*, e.g. the use of an ontology to map different customisations, and the employment of *modification-aware mining algorithms* to detect and deal with both *evolutionary* and *ad-hoc* modifications. Within the process mining community, various techniques, many based on the statistical notion of *outlier detection*, have been proposed to discover and remove noise. In [27,16] and [9], frequency-based approaches to detecting noise are proposed. In [27], the frequencies of direct-follow and eventual-follow events are used as input to machine learning techniques that are used to induce association rules for causal and exclusive/parallel relations between events in the log. In [16], the authors firstly mine for frequently-occurring patterns that can be used to characterise traces in the log and then use a cluster-based outlier identification approach to identify traces that hardly belong to any of the computed clusters i.e. traces that do not follow any of the frequent behaviours evident in the log. By contrast, in [9], the authors describe an approach that detects and automatically removes noise at the direct-follow event level. Here an event log is firstly modeled as an automaton that captures all direct-follow relations in the log. Arcs in the automaton that represent low frequency event transitions are pruned from the automaton. The log is then aligned with the automaton and those events not fitting the automaton are deemed 'outliers' and removed from the log. The work by Rogge-Solti [39] can be seen as exploiting mathematically-based models to repair event logs. In this work, the author uses a type of stochastic Petri net [26] to reason about path probabilities from which 'predictions' about the most likely missing events can be made. Furthermore, the authors also use Bayesian network modeling [32] to compute the most likely timestamps for these missing events. We argue that while the approaches proposed by these authors to discover usable process models from noisy event logs are theoretically founded, the reliance on the concept of statistical

Table 1
Manifestation of quality issues in event log entities [6].

		Event log entities								
		Case	Event	Relationship	Case attrs.	Position	Activity name	Timestamp	Resource	Event attrs.
Event log quality issues	Missing data	I1	I2	I3	I4	I5	I6	I7	I8	I9
	Incorrect data	I10	I11	I12	I13	I14	I15	I16	I17	I18
	Imprecise data			I19	I20	I21	I22	I23	I24	I25
	Irrelevant data	I26	I27							

outlier may very well result in rather ‘sterile’ event logs in which interesting events, which may be of great interests to domain experts, have been inadvertently removed. This raises the issue of the *validity of event logs* as depicted in Fig. 1. On the other hand, the data imperfection patterns proposed in this article do take into account, to a certain extent, the *broader state of the event log as a whole* before suggesting remedial actions. In fact, our data imperfection patterns apply and extract rules for data cleaning that are *informed by experiences of process mining analysts themselves*. The *main advantage* of using data imperfection patterns as the first step of data cleaning exercises is that *we may still retain interesting events that may have otherwise been lost using statistical-based approach*.

Within the general data mining community, the issue of data quality has been rather extensively discussed as proven by the plethora of literature in this field, such as, to name a few, the work by Kim et al. [20], Gschwandtner et al. [17], Rahm and Do [37], Müller and Freytag [31], Oliveira et al. [34], and Batini and Scannapieco [5]. Nevertheless, we contend that the *nature of event log used in process mining is distinct* from the nature of the data that is typically used in data mining. Process mining uses an event log as the starting point of analysis. A distinct characteristic of an event log is that there exists *temporal constraints* among events, both from a case perspective and a resource perspective: the former restricts the types of actions that can be executed at any given point in time (as they are dependent on the completion of the preceding activities), while the latter restricts who and when an activity can be executed depending on resources availability and capability. Consequently, each row in an event log (i.e. an event) has temporal relationships with other events, either through resource or case constraints. This is in contrast to other types of log typically used in traditional data analytics (such as data mining or even basic spreadsheet analysis) whereby the concept of case or resource temporal constraints does not exist. (In process mining, the data is structured as multiple-record cases, whereas in data mining, the data is structured as single-record cases.) Therefore, while we appreciate the richness of work in this domain, the distinct characteristics of the event log require us to address event log quality issues differently. Having said that, it is part of our future work to incorporate and extend existing data cleansing work from the traditional data mining domain to repairing event logs.

4. Process mining data quality framework

It is clear that while there exist many conceptual overlaps in the various quality frameworks, there is no consensus on either the set of dimensions that constitute a data quality framework or the definition of the individual dimensions. Further, as stated in the preceding section, it is clear that there are significant differences between the data in an event log and data derived from a typical information system. These differences are unrecognised in today’s literature on data quality. The need to deal with temporal data dependencies in process mining opens up new notions of ‘quality’. It follows that a data quality framework for process mining will be somewhat different from any data quality framework so far proposed for information systems. For instance, in the Process Mining Manifesto [45], with reference to the quality of events, the authors state

“Events should be *trustworthy*, i.e., it should be safe to assume that the recorded events actually happened and that the attributes of events are correct. Event logs should be *complete*, i.e., given a particular scope, no events may be missing. Any recorded event should have well-defined *semantics*. Moreover, the event data should be *safe* in the sense that privacy and security concerns are addressed when recording the events. For example, actors should be aware of the kind of events being recorded and the way they are used.”

In this paper we adopt the quality framework described by Bose et al. [6] and summarised in Table 1 below, to classify the quality issues arising from the presence of the data imperfection patterns (in an event log) described in this paper. By way of explanation, consider the quality issue *I3 - Missing data, relationship*. This refers to the situation where it is not possible to associate an event with its case (process execution instance) because the case identifier is missing from the event record. A fundamental requirement of process mining is that events can be associated with cases. Where this is not the case, a process mining analysis is not possible. A complete description of each quality issue can be found in the related technical report referenced in [7].

5. Event log imperfection patterns

Inspired by the problems that one may encounter in transforming raw data source logs into an event log that is ‘clean’ and usable for process mining analysis, we extract a set of *event log imperfection patterns* commonly encountered in pre-processing raw source logs. The use of patterns as a means of understanding, and communicating the characteristics of an apparently chaotic domain is a fundamental human behaviour. The pattern approach is used in diverse areas including software design [15,52], workflow functionality [47], security [22], insider threat protection [30,29,28] and architecture/town planning [3]. Our rationale for adopting the pattern approach to event log cleaning is neatly summed up in [3] where the authors state that each pattern in a set “describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”.

We accept the definition of a pattern in [38] as “the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts” and argue that the use of patterns provides a basis for the systematic investigation of event logs for the presence and remediation of data quality issues that will negatively impact a process mining analysis. The patterns described here come from our own experiences in preparing event logs from raw sources and occurred frequently enough and are sufficiently generic (non-analysis specific) to warrant abstraction as a pattern. We provide detailed descriptions for each of the event log imperfection patterns that we propose, including the manifestation of the pattern within an event log, its detection, the affected data quality dimension(s) (as described in Section 4), the effect of the imperfection pattern on subsequent process mining analyses, remedial action for the detected imperfection, and, for some patterns, the side effects of the remedial action. We also provide a formal description of the manifestation of each pattern.

Table 2
Relationship between individual patterns and quality framework.

Pattern	Quality issue(s)
Form-based Event Capture	I16, I27
Inadvertent Time Travel	I6
Unanchored Event	I23, I6
Scattered Event	I2
Elusive Case	I3
Scattered Case	I12
Collateral Events	I27
Polluted Label	I15, I17
Distorted Label	I15
Synonymous Labels	I15
Homonymous Label	I22

Table 2 shows, the dimension(s) of the quality framework affected by each log imperfection pattern. In our pattern collection, event log imperfection patterns are described using the following components:

- *Description*: outline of the pattern and how and where the pattern may be introduced into a log
- *Real-life Example*: example of the pattern drawn from practical experience
- *Affect*: consequence of the existence of the pattern on the outcomes of a process mining analysis
- *Data Quality Issues*: type of data error and the event log entities affected by the pattern
- *Manifestation and Detection*: strategy to detect the presence of the pattern in a log
- *Remedy*: how the pattern may be removed from a log
- *Side-effects of Remedy*: possible, undesirable consequences of application of the remedy
- *Indicative Rule*: formal description of the way the pattern may be detected in a log.

The indicative rule is structured to illustrate the general conditions under which the pattern may manifest in the log, i.e. the indicative rule is not meant to exhaustively describe all possible manifestations of the pattern. Further the rule only indicates that the pattern may be present in the log but does not address the *pervasiveness* of the pattern (extent to which the log is affected by the pattern). The indicative rule also serves to distinguish between patterns that, at some level of generalisation, could be seen as similar. For instance, both *form-based event capture* and the *collateral events* patterns manifest similarly in the log (multiple events recorded with similar timestamps). Their respective rules reflect the different contexts associated with each pattern's manifestation and detection. For example, form-based event capture typically manifests as multiple events with similar completion times, while collateral events typically manifests as multiple events with similar created times.

Pattern. Form-based Event Capture

Description. This pattern may occur where the data in an event log is captured from electronic-based forms (e.g. a patient test results form). Users (such as nurses and doctors) click a 'Save' button to trigger the recording of the data captured by the form, often with the undesirable side effect of associating all data captured by the form with the same timestamp (the time the user clicked the 'Save' button). The more interesting information about the ordering of activities from which the values of the data inserted into the forms were derived, such as the time a blood sample was taken, is flattened into one timestamp. This method of data recording can result in an additional problem. When the form is updated at some point in the future, the system may actually store *all* data captured by the form again, even though only a few data items were actually changed. As a result, the log data stored by such a system contains redundant information due to the fact that

Table 3
An example of the 'Form-based Event Capture' pattern.

CaseID	Event	Timestamp	Description
ID1	Primary Survey	2012-11-23 15:42:38
ID1	Airway Clear	2012-11-23 15:42:38
ID1	2012-11-23 15:42:38
ID2	Primary Survey	2012-11-24 09:58:33
ID2	Airway Clear	2012-11-24 09:58:33
ID2	2012-11-24 09:58:33
ID2	Procedure 1	2012-11-24 09:58:33	Completed on 2012-11-24 06:58:34

some data items in the form were not changed but were re-recorded in the storage with a new timestamp (the time when the update event happened).

Real-life Example. We encountered this data imperfection pattern in event logs extracted from a number of Australian-based hospitals. An example of such a log is provided in Table 3.

Here, the first three events have the same timestamp. Furthermore, we know that these three events were derived from a form-based system because the first event, named 'Primary Survey', is the name of the form that was being used. A similar observation can be made for the last four events.

Affect. This imperfection pattern 'flattens' the temporal ordering of events as they are all assumed to have happened at the same time resulting in the actual ordering of events being lost. Ignoring this data pattern may result in the discovery of complex process models as all events sharing the same timestamp in a case will likely be treated as parallel events, and consequently, be modeled in a parallel manner. Modeling parallel events graphically tends to increase the number of arcs (and the often unavoidable cross-cuttings of arcs) in the models, thus making them difficult to comprehend. Finally, this imperfection pattern may produce unnecessary duplication of events as certain events may be re-recorded as a result of the updates of a few other data items within the same form. This situation is likely to result in the extraction of misleading process mining analysis results due to the existence of events in the log that did *not* actually happen.

Data Quality Issues. I16 – Incorrect data: timestamp, I27 – Irrelevant data: event – The temporal flattening introduced into an event log through the occurrence of this pattern negatively impacts the attribute accuracy of the log in that the timestamps of the events reflect the saving of the form rather than the performance of the event. Further, if the system records all fields on the form rather than only those fields that have changed, the trace completeness may be affected through erroneous inclusion of events that did not actually happen in the case.

Manifestation and Detection. This pattern's signature is the existence of groups of events in a log with the same case identifier and timestamp value. The signature of this pattern can be detected by searching the event log for groups of events with the same case identifier and timestamp value. Alternatively, the log can be searched for the presence of 'marker' events with activity names similar to field labels known to exist on the same form (assuming that such information about the forms can be obtained from system users). If found, the timestamps of the 'marker' events can be checked to see if they are the same. Regardless, the regular occurrences of groups of events that share the same timestamp value is already a good indication of the presence of the 'Form-based Event Capture' pattern.

Remedy. The simplest remedial action is to aggregate all events, within each group of events having the same timestamp, into *one event* only. An additional attribute can be created for this event with a complex data structure to capture relevant data from the aggregated events. This approach removes all other events that have been recorded from one form, thus reducing the amount of parallelism in the discovered models. However, such a remedy can only be applied if it

is sensible to represent the information collected from the form as one process step. For example, if all events with the same timestamp reflect nothing more than a nurse performing various types of medical checks on a patient where all of those checks fall under the umbrella of ‘vital signs checks’ activity, then we can aggregate them into one event named ‘Vital Sign Checks’. However, if those events with similar timestamps contain two or more distinct and/or important process steps that need to be explicitly considered in the analysis, important information may be lost through simple events aggregation. Instead, each group of events will need to be aggregated into two or more events reflecting the distinct steps taken. These aggregated events, will however, still share the same timestamp.

From the process mining case studies in which we have been involved, the following interesting ‘variant’ of the ‘Form-based Event Capture’ pattern has been observed. There is a set of events with the same timestamp and case identifier (e.g. case identifier ‘ID2’ in Table 3), however, the relevant event timestamp information was actually recorded in a column that is different from the ‘timestamp’ column. The last row of Table 3 depicts this scenario whereby the form recorded a ‘Procedure 1’ event, but the actual timestamp regarding the completion of the procedure was recorded within the ‘Description’ field itself. In this situation, we were able to partially sequentialise the events by using the information extracted from the ‘Description’ column. A more complex situation occurs when there is an update to only some data items of a form, triggering the recording of a new set of events, each with the same timestamp (some of which may be duplicate events because their values did not require any updates). Properly addressing this scenario requires firstly an understanding of how updates are recorded in the event log. There are two situations: (i) the updates of one or more data items in a form result in the recording of all fields as events in the log, or (ii) the updates of one or more fields in a form will result in the recording of only those fields whose values have changed. In the former case, it is necessary to identify the specific information that has changed. In both cases, it is important to note what process step(s) may have taken place in order for the data items in the forms to be updated, and then aggregate those changed data items into one or more events as needed.

Side-effect of Remedy. Where a process requires a form update, it may be the case that, for a given activity, the ‘new’ value of a data item is the same as the ‘old’ value of the data item. Where the form logging mechanism writes out all data values in the form, the fact that the ‘old’ and ‘new’ values of the data item are the same makes it difficult to determine whether the activity was carried out a second time or the data item was simply re-written as part of the form update process. In this scenario, we may lose the ‘update’ action on those fields where ‘new’ values and ‘old’ values are the same.

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $AN_{\mathcal{L}}$ be a set of attribute names found in \mathcal{L} and \mathcal{D}_a be the set of all possible values of $a \in AN_{\mathcal{L}}$. Let $caseid \in AN_{\mathcal{L}}$ be the case identifier attribute and \mathcal{D}_{case} be the set of possible case identifiers in log \mathcal{L} . Let $time \in AN_{\mathcal{L}}$ be the event timestamp attribute and \mathcal{D}_{time} be the set of possible timestamps in log \mathcal{L} . Here we make use of the extension operator χ defined in [43] that extends a relational expression to include a new attribute representing the count of rows. The form-based event capture pattern may be present in \mathcal{L} if there exists a non-empty set of timestamps that have multiple events recorded. That is, if:

- $\sigma_{count > \theta} \chi(\mathcal{L}, \{caseid, time\}, count)$ is not empty, where θ is some significance level that reduces the likelihood of returning instances where unrelated events are recorded coincidentally.

Pattern. Inadvertent Time Travel

Description. This pattern captures the situation where certain entries in a log are recorded with an erroneous timestamp due to the ‘proximity’ of the correct data value and the incorrect data value. Here ‘proximity’ refers to a situation where two timestamp values are so

Table 4

An example of the ‘Inadvertent Time Travel’ pattern.

Episode ID	Activity	Timestamp	...
ID1	Arrival first hospital	2011-09-08 00:30:00
ID1	Injury	2011-09-08 23:47:01
...
ID1	Operation	2011-09-09 16:30:00

close to each other that human error can inadvertently result in incorrect timestamp values to be recorded.

A typical example of a proximity error pattern is the recording of incorrect timestamps for events that happen just after midnight. Often, the date portion of the timestamp is incorrectly recorded as the date of the previous day, while the time portion of the timestamp is recorded correctly. Another example of this pattern is the recording of incorrect timestamps of events due to users simply pressing the wrong key(s) on a keyboard i.e. inadvertently pressing keys adjacent to the intended key(s).

Real-life Example. We found this pattern in data from an Australian hospital emergency department where most data entry was performed manually.

An event log sample including this pattern is provided in Table 4, and is illustrated in Fig. 3. Here the event ‘Arrival first hospital’ occurred as the first event in the case. However, common sense would dictate that an injury event precedes being sent to a hospital. Notice that if the date of the ‘Arrival first hospital’ activity is changed to ‘2011-09-09 00:30:00’, the event would be reordered and become the second activity of the case making the trace appear more believable as the ‘Arrival first hospital’ activity then occurs after the ‘Injury’ event.

Affect. The existence of this pattern will result in incorrect models being discovered as the models are likely to allow behaviours that did not occur in reality. This pattern will also impact the accuracy of time-related performance analysis results (e.g. working time and waiting time), although the impact may not be serious if such patterns occur in only a relatively small number of cases.

Data Quality Issue. I16 – Incorrect data: timestamp – The incorrect timestamp values introduced into an event log through the occurrence of this pattern negatively impacts the attribute accuracy of the log in that the temporal ordering of the events no longer reflects the actual ordering of events.

Manifestation and Detection. The pattern typically manifests itself by the existence of a number of cases in which event ordering deviates significantly. That is, given two activities a_1 and a_2 with a strict temporal ordering property such that a_1 should always occur before a_2 , there exists at least a case with two events with activities a_1 and a_2 with incorrect ordering. We call these two events the *misplaced events*. Furthermore, once such sequence(s) of misplaced events are discovered, it is necessary to go back to the log and test if, by changing the timestamps for those misplaced events (according to some known rules), those events can be ‘placed’ back to their proper ordering in the context of the *overall trace*. If this is the case, it is quite reasonable to say that such a pattern exists in the log. For example, consider a process with only three sequential activities A, B, and C. If the log contains a trace with the following event ordering: A (2011-09-25 21:56:23), B (2011-09-25 00:23:11), and C (2011-09-26 01:34:56), the



Fig. 3. An illustration of the ‘Inadvertent Time Travel’ pattern.

discovered process model will have an arc from B to A , and from A to C . However, in the domain, the ‘ground truth’ is that B can only occur after A and before C . If the date component of the timestamp of B is modified from ‘25’ to ‘26’, activity B will be in the proper order. In this scenario, the ‘Inadvertent Time Travel’ pattern exists in the event log.

Remedy. Addressing this pattern in a generic manner requires knowledge of the minimum restrictions applicable to the ordering of all activities in the log. With this knowledge, each trace in the log can be examined to identify the existence of traces that violate the minimum ordering restrictions. Once discovered, the timestamp of events in the traces that violate the ordering restriction can be ‘fixed’ by applying various remediations for standard proximity errors (e.g. adding or subtracting one day from the timestamp, or flipping the value of the timestamps based on proximity of the keys in a standard keyboard layout). If one of those fixes re-orders the events in the trace such that the trace no longer violates the ordering restrictions, the erroneous old timestamp value should be replaced with the new value. Otherwise, it may be that the original timestamp value does not suffer from the ‘Inadvertent Time Travel’ pattern after all (though the value may still be incorrect).

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $AN_{\mathcal{L}}$ be a set of attribute names found in \mathcal{L} and \mathcal{D}_a be the set of all possible values of $a \in AN_{\mathcal{L}}$. Let \mathcal{D}_{case} be the set of all case values in \mathcal{L} , \mathcal{D}_{time} be the set of possible timestamps in log \mathcal{L} , \mathcal{D}_{act} be the set of all activity values in \mathcal{L} and $\sqsubset \subseteq \mathcal{D}_{act} \times \mathcal{D}_{act}$ be a strict partial order.

The *inadvertent time travel* pattern may be present in log \mathcal{L} if there exists events e_1 and e_2 such that:

- $\#_{case}(e_1) = \#_{case}(e_2) \wedge \#_{time}(e_1) < \#_{time}(e_2) \wedge \#_{act}(e_2) \sqsubset \#_{act}(e_1)$

Pattern. Unanchored Event

Description. This pattern refers to a situation where the timestamp values of an event log are recorded in a format that is different from that which is expected by the tools used to process the event log. Consequently, the loading of the event log into those tools will result in incorrect interpretation of timestamp values. Typical format variations include the confusion between month-day vs. day-month format, the use of colon (‘:’) symbol vs. dot (‘.’) symbol to separate between hour, minute, and second information, and the varying manner in which timezone information is encoded. This pattern is likely to occur when the event log is constructed from multiple sources.

Real-life example. This pattern is commonly encountered while conducting data pre-processing. For example, Table 5 (top-part) shows a sample of a few events that we received from an Australian hospital. The original timestamps were presented in day-month-year format. These events were then imported into a database which expected month-day-year format during import. The first three events of the original data set were imported incorrectly using the month-day-year

Table 5
An example of the ‘Unanchored Event’ pattern.

Original data			
CaseID	Activity	Timestamp	Notes
1234567	Progress note	01/09/2013 21:53:25
1234567	Medical note	02/09/2013 01:11:25
1234567	Therapy	12/11/2013 16:08:23
1234567	Discharge letter	14/11/2013 16:43:29
Parsed data			
CaseID	Activity	Timestamp	Notes
1234567	Progress note	09/01/2013 21:53:25
1234567	Medical note	09/02/2013 01:11:25
1234567	Discharge letter	11/14/2013 16:43:29
1234567	Therapy	12/11/2013 16:08:23

format *without* causing parsing errors due to the ambiguity of the values. That is, the original dates are: 1st September 2013, 2nd September 2013, and 12th November 2013. Here, as the ‘day’ portions of the timestamps do not go beyond the number “12”, they can be re-interpreted as the value for the ‘month’ component without causing a parsing problem. The first three imported events have the timestamp values interpreted incorrectly as 9th January 2013, 9th February 2013, and 11th December 2013.

Note that the last event of the original data with the timestamp of 14th November 2013 was imported correctly as the day portion of the timestamp (which is ‘14’) is unambiguous: it has to refer to a day, not a month. Therefore, the timestamp for the event ‘Discharge letter’ was imported correctly. It is worth noting that, despite the apparent inconsistencies in the formats of the original timestamps, *no warnings* were issued by the database during data import. This lack of warning could easily be missed by analysts, especially when one deals with millions of events.

Affect. Incorrect timestamp values will adversely affect process mining results. From the example given in Table 5, it is not difficult to see how a process model discovery exercise for example, may output process models that are substantially different from reality in both event ordering and case duration.

Data Quality Issues. I23 – Imprecise data: timestamp, I16 – Incorrect data: timestamp – The incorrect timestamp values introduced into an event log through the occurrence of this pattern negatively impacts the attribute accuracy of the log in that the temporal ordering of the events no longer reflects the actual ordering of events.

Manifestation and Detection. This pattern generally manifests itself when processing an event log. Some tools produce error messages when used to load an event log with an incompatible timestamp format. However, tools, such as Microsoft Excel, are quite ‘relaxed’ in the way they parse timestamp information (as they often have built-in intelligence to detect the correct timestamp format) and may not produce any warning or error messages, although the data may have been loaded incorrectly. If the latter, the detection of this data imperfection is more difficult. This pattern may also be detected through the discovery of process models with unexpected, and often incorrect, ordering of activities and the extraction of unreasonably long or short working and/or waiting times. Another indicator is the existence of missing timestamp information across many events in the log – due to the tool not being able to parse the timestamp information correctly and ignoring the values altogether. Lastly, the pattern may manifest through elements of the timestamp having values outside the expected range of the element, or spanning only part of the expected range of the element. For instance, ‘day’ values being only in the range [1..12] (indicating month and day portions of the timestamp have been interchanged).

Remedy. To address this problem, it is necessary to ensure the tools used to import the event log do not inadvertently mis-interpret the timestamp information without producing any warnings. While tedious, the simplest way to handle this situation is to *prevent* the tools from interpreting certain fields in the log as ‘timestamp’ information in the first place. This can be achieved by adding a few characters, such as asterisks, before and after the timestamp values to force the ‘switching off’ of the built-in timestamp interpretation mechanism that many tools have. Following this, the file can be edited (with a text editor) and appropriate ‘string’ manipulation techniques (such as find and replace) applied to reformat the string values as timestamps.

Side-effects of Remedy. In practice, there could indeed be events that were executed in a sequence that did not meet the ‘expected’ ordering restrictions. Hence, this remedy may result in the loss of interesting deviant behaviours in the log.

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $AN_{\mathcal{L}}$ be a set of attribute names found in \mathcal{L} and \mathcal{D}_a be the set of all possible values of $a \in AN_{\mathcal{L}}$. For each domain \mathcal{D}_{a_i} of an attribute, there is an expected set of values $E_{a_i} \subset \mathcal{D}_{a_i}$. The unanchored event pattern may exist in log \mathcal{L} if:

Table 6
An example of the ‘Scattered Event’ pattern.

Event log 1			
Case ID	Activity	Timestamp	Description
1234567	Surgical Procedure	21/09/2011 08:11:25	Stent insertion and angiography
1234567	Procedure start-time	21/09/2011 08:11:25	0:2011092010480000:0.000000:0:0
1234567	Procedure end-time	21/09/2011 08:11:25	0:2011092010590000:0.000000:0:0

Event log 2			
CaseID	Activity	Timestamp	Description
1234567	Order a surgical procedure	18/09/2011 13:26:32	< AttributeValue 1 >, < Attribute Value 2 >, ...

- $|\sigma_{a_i \notin E_{a_i}}(\mathcal{L})| > 0$ (i.e. there are unexpected values in the attribute. For instance, in a timestamp, there are ‘month’ values outside the expected [1..12] range.)
- $|\Pi_a \sigma_{a_i \in E_{a_i}}(\mathcal{L})| < |E_{a_i}|$ (i.e. there are values that would be expected for the attribute that are not found in the log. For instance, in a timestamp, the ‘day’ values include only [1..12].)

Pattern. Scattered Event

Description. This pattern refers to events in an event log which have attributes that contain further information that can be used to derive new events. In other words, there is information contained within an existing event log that can be exploited to construct additional events, but, the information is hidden in attribute values of several events.

Real-life Example. We encountered this pattern (illustrated in Table 6) in event logs obtained from an Australian hospital. The entries from the first event log shown in Table 6 record events from a form (possibly named the ‘Surgical Procedure’ form); hence, we can see that all of the entries have the same timestamp (which is 21/09/2011 08:11:25). Interestingly, there are two events in this event log (i.e. the ‘Procedure start-time’ event and the ‘Procedure end-time’ event) that actually give us further information about two other events that have happened.

In particular, if we look at the corresponding values in the ‘Description’ attribute column, we can discern timestamp values, e.g. the string ‘2011092010480000’ can be reformatted to ‘2011-09-20 10:48’. Furthermore, in the second log, there is also an entry for an order event for the same patient with the timestamp of 18/09/2011 13:26:32, and within the corresponding ‘Description’ attribute, there exist further attribute values that are relevant to the surgical procedure being ordered. By combining the information from these two event logs, we can re-create the following scenario: a surgical procedure was ordered on the 18/09/2011, and the record of the procedure was entered into the system on the 21/09/2011 (through a form). Within this form, it was recorded that the procedure started on 20/09/2011 10:48 and finished on 20/09/2011 10:59. Most interestingly, we can now re-construct two new events: one event to capture the start of the procedure, and another event to capture the end of the procedure. Additional information about these two events can be discerned using the corresponding attribute values obtained from the second log.

Affect. The existence of this pattern is not likely to add additional overheads or hinder the process mining exercise. However, it represents untapped information that could enrich the insights obtained from a process mining exercise.

Data Quality Issue. I16 – Missing data: event – The occurrence of this pattern affects the trace completeness through omission of events that actually happened in the case but, due to the logging mechanisms in the underlying information system, were not recognised as such.

Manifestation and Detection. This pattern manifests itself through event attribute(s) (other than the timestamp attribute) where part of the attribute value could be interpreted as a timestamp and (possibly) part as additional information. To detect this pattern, one needs to determine one or more attribute name(s) whose values can guide us in extracting the values needed to form a new event. We call these attribute names as the *guiding* column(s). The values that need to be extracted to form new events also come from one or more attribute(s) of the same event – we call these the *target* column(s). For example, in the example log above, the ‘activity’ attribute is the guiding column, while the ‘description’ attribute is the target column.

More importantly, the value(s) of the guiding column(s) need to contain ‘marker’ value from which necessary *activity name* and *timestamp* information can be extracted to form a new event. In the example above, the ‘marker’ values include ‘Surgical Procedure’, ‘Procedure start-time’, and ‘Procedure end-time’. The first value leads us to extract the name of the surgical procedure performed (i.e. the activity name of an event), while the second and third values allow us to extract the start and end timestamps for the procedure (i.e. the timestamp value of an event).

Generally, the detection of this imperfection pattern requires manual effort and the assistance of domain experts as this ‘hidden’ information (from which new events can be derived) could be encoded in practically any attribute value of an event log.

Remedy. Given the variety of manners in which this pattern may manifest itself, a generic solution to fix this issue is unlikely. Nevertheless, once the location of the information from which new events can be re-constructed is known, it is possible to develop a tool to automate the creation of the new events.

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $g_{act}, g_{ts} \in AN$ be guiding column names, $t_{act}, t_{ts} \in AN$ be target column names, $v_{act} \in \mathcal{D}_{g_{act}}$ be a marker value that guides the extraction of a new activity name from the target column t_{act} , and $v_{ts} \in \mathcal{D}_{g_{ts}}$ be a marker value that guides the extraction of a new timestamp value from the target column t_{ts} . Finally, let $W \subseteq \mathcal{D}_{t_{act}}$ be a set of possible new activity name values.

A scattered event log imperfection pattern may exist if:

- $|\sigma_{g_{act}=v_{act}} \Pi_{g_{act}, t_{act}}(\mathcal{L})| \approx |W|$ (i.e., there are approximately as many unique combinations of the marker value for activity name with its possible values as the number of possible new activity names in W), and
- given $\mathcal{L}_{ts} = \sigma_{t_{ts} \in W \wedge g_{ts}=v_{ts}} \Pi_{t_{ts}}(\mathcal{L})$, the values returned in \mathcal{L}_{ts} should contain a pattern that suggests timestamp values.

Pattern. Elusive Case

Description. This pattern refers to a log in which events are not explicitly linked to their respective case identifiers. This pattern is often seen in an event log that is derived from a system that is not process-aware, or is not meant to support activities in the context of a process (e.g. a GPS tracking system, a web traffic log, or industrial devices).

Consequently, the concept of a ‘case’ cannot be trivially defined by simply using the information in the log as-is. Nevertheless, the notion of a ‘case’ exists and can be discerned, especially by domain experts (e.g. a user web-browsing session or a journey from a geographical location A to another location B).

Real-life Example. We encountered this pattern in a GPS tracking data set. In this data set, we have a series of GPS-related events, such as when a vehicle entered and exited a particular geographical area, when the ignition of the vehicle was turned on or off, and when the vehicle went into a sleep mode. Table 7 shows an anonymised snippet of the data set. As can be seen, events recorded in that table do not have any identifiers that can be used to group them into distinct cases.

Affect. Process mining requires each event be attributable to a case. The existence of this imperfection pattern will prevent a process mining analysis being conducted until the concept of what comprises a case has been resolved.

Table 7
Example of the ‘Elusive Case’ pattern.

Vehicle	Event Type	Timestamp
Van1	Ignition on	2011-02-07 07:58:00
Van1	Exit Area A	2011-02-07 08:00:00
Van1	Enter Area B	2011-02-07 09:01:39
Van1	Exit area B	2011-02-07 09:22:01
Van1
Van1	Enter Area X	2011-02-07 15:54:08
Van1	Ignition off	2011-02-07 18:00:00
Van1
Van1	Enter Area X	2011-02-08 08:00:00
Van1
Van1	Enter Area Z	2011-02-08 10:00:00
Van1	2011-02-08 01:02:56

Data Quality Issue. I16 – Missing data: relationships – The relationship between events and cases is missing.

Manifestation and Detection. This pattern manifests itself in an event log where events cannot be linked to one another in any meaningful way, due to the lack of information necessary to group events into cases. The ‘Elusive Case’ pattern can be said to occur where there does *not* exist an attribute that is common to all events for which the value of the attribute(s) in each event can be used to group events into a case (i.e. all events relate to the same process instance and collectively capture all activities of a case that one can reasonably expect to occur). This pattern can also be detected by randomly tagging one or more attributes as the ‘case identifier’ attribute(s) of an event log and then attempting to discover a process model. One could do so when loading an event log into a process mining tool such as Disco.² If the resulting model captures a complete process model with reasonable temporal dependencies between events, the tagged attribute can be reasonably used as the ‘case identifier’ attribute for the log. In this situation, the ‘Elusive Case’ does not exist in the log. If there is/are no attribute(s) that, when tagged as case identifier, can deliver an acceptable process model, it is likely that the ‘Elusive Case’ pattern exists in the log.

Remedy. To address this situation, the cases to which events belong need to be correctly identified. In most instances, this can be done by correlating information in the event log with information from another source. For example, in the GPS data set, a case was defined as a pre-arranged journey of a particular vehicle from the exit of the vehicle from a location designated as the starting point of the journey, to the entry of the vehicle to the last designated location. Information from an associated journey planner table (which is produced on a daily basis) was correlated with the GPS events seen in the event log allowing events in the original event log to be mapped to their respective case identifier (the journey identifier) (see Table 8).

Side-effects of Remedy. Side effects of such a remedy include (i) incorrect mapping of events to their cases, and (ii) the omission of many events in the log as they cannot be mapped to any particular cases resulting in the loss of potentially important information.

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $AN_{\mathcal{L}}$ be a set of attribute names found in \mathcal{L} and \mathcal{D}_a be the set of all possible values of $a \in AN_{\mathcal{L}}$. Let \mathcal{D}_{time} be the set of possible timestamps in log \mathcal{L} , \mathcal{D}_{act} be the set of all activity values in \mathcal{L} and $\sqsubseteq \subseteq \mathcal{D}_{act} \times \mathcal{D}_{act}$ be a strict partial order.

The *elusive case* pattern may be present in log \mathcal{L} if, for any attribute a , there exist events e_1 and e_2 such that:

- $\#_a(e_1) = \#_a(e_2) \wedge \#_{time}(e_1) < \#_{time}(e_2) \wedge \#_{act}(e_2) \#_{act}(e_1)$ (i.e. for any attribute chosen as candidate case identifier, it is always possible to find events that contradict the strict partial order).

Table 8
Example of the ‘Elusive Case’ pattern.

Journey	Vehicle	Event Type	Timestamp
J1	Van1	Ignition on	2011-02-07 07:58:00
J1	Van1	Exit Area A	2011-02-07 08:00:00
J1	Van1	Enter Area B	2011-02-07 09:01:39
J1	Van1	Exit Area B	2011-02-07 09:22:01
J1	Van1
J1	Van1	Enter Area X	2011-02-07 17:54:08
J1	Van1	Ignition off	2011-02-07 18:00:00
J1	Van1
J2	Van1	Exit Area X	2011-02-08 08:00:00
J2	Van1
J2	Van1	Enter Area Z	2011-02-08 10:00:00
J2	Van1	Ignition off	2011-02-08 10:02:56

Pattern. Scattered Case

Description. This pattern describes a situation where key process steps are missing in the event log being analysed (thus not giving the complete picture of the activities involved in a case) but are recorded elsewhere (e.g. in a different system). This pattern then is concerned with constructing a complete picture of the cases in a log by merging information from different sources. The key challenge here is ascribing information extracted from different sources to the correct event log case identifier when each source may have its own unique identifier (the so called ‘record linkage’ problem [33]).

Real-life Example. We encountered this pattern in one of our hospital logs. We can see in Table 9 that the Event Log table contains two case identifiers. These two case identifiers initially seem to belong to two different patient flows. Similarly, if we look at the Order table, we see the two case identifiers again. However, by looking at those two cases individually, it is easy to see that the information recorded in each case is incomplete. The case identifier ‘1234567’ contains only activities that were conducted within the emergency department of the hospital, while the case identifier ‘8912345’ contains only activities that were conducted within the hospital.

Because we are interested in analysing the end-to-end patient flow, we need both the emergency department and hospital activities to be analysed. However, at this point, we can see that those two so-called ‘sub-processes’ are identified using different case identifiers. To establish the link between the two separate case identifiers (which should be considered as one case instead), we rely on the information from the Case Summary table where there exists a ‘Master Record Number’ field. This field enables us to create the link between the two different case identifiers. Also note that this ‘Master Record Number’ field does not exist in the Event Log or in the Order table.

Affect. If not addressed properly, this event log imperfection pattern will result in discovered process models representing only a fraction of the total process due to the event log containing incomplete trace information.

Data Quality Issue. I112 – Incorrect data: relationship – The associations between events and cases are logged incorrectly from the domain perspective. That is, within each contributing system’s log, events are correctly ascribed to cases, but as there is no common case identifier at the domain level, when the events from the contributing system’s logs are combined to form a consolidated process level log, it is not possible to properly merge events into cases.

Manifestation and Detection. The manifestation of this pattern is as ‘gaps’ with regard to activities recorded in an event log. For example, for all cases in a log, the time when a blood test was ordered is recorded, but, *expected* successor activities such as the drawing of the blood sample and the return of the blood test results are not recorded. In this situation, it is clear that there is a segment of information that is missing from the event log which will need to be ‘patched’ by drawing the required information from other logs. Where all the activities involved in the process being analysed are known, detection of this

² <http://fluxicon.com/disco>

Table 9
An example of ‘Scattered Case’ pattern.

Event Log			
CaseID	Activity	Timestamp	Notes
1234567	Progress Note	7/09/2013 00:50:30
1234567	Medical Note	7/09/2013 00:52:25
1234567	ECG	7/09/2013 03:52:48
8912345	FBC	12/09/2013 15:59:32
8912345	Therapy	13/09/2013 10:20:01

Order Table			
CaseID	Activity	Timestamp	Notes
1234567	Order ECG	7/09/2013 00:53:45	
8912345	Order FBC	12/09/2013 14:04:51

Case Summary Table			
CaseID	Visit Type	Timestamp	Master
1234567	Emergency	6/09/2013 23:47:00	MRN1234567
8912345	Hospital	8/09/2013 13:45:00	MRN1234567

pattern involves comparing the list of unique activity names recorded in each log against expected activity names. The absence of one particular event log that provides all the activity names expected is a sign of the existence of such a pattern. A further check would involve matching each activity with known predecessor and successor activities. The respective frequencies of the activity, the predecessor activity and the successor activities should match. A discrepancy could indicate at least partial trace incompleteness.

Remedy. This imperfection pattern can be redressed through application of an appropriate record linkage technique so that events from various sources can be merged into one log with events from the various sources being properly attributed to cases. The approach to merging will depend on the availability (or not) of some unique identifier. Where all data sources use exactly the same case identifiers, the source logs can simply be merged into one event log. Where individual source logs use different case identifiers, but there is a global unique identifier that can be used to establish the link between the different case identifiers, e.g. through the use of the ‘Master Record Number’ values in the Case Summary table (as shown in Table 5), the global unique identifier should be used as the merge key with the case identifier being set as either the global unique identifier or one of the existing case identifiers. If no form of unique identifier exists, a standard record linkage technique can be used to perform the match. An outline of the standard record linkage and de-duplication process first described by Newcombe [33] and subsequently formalised by Fellegi and Sunter [12], is presented in [36]. An overview of various merge-purge techniques is presented in [11].

Side-effects of Remedy. Where no global unique identifier can be determined, i.e. a record linkage algorithm has been applied, it is possible that events will not be properly attributed to cases due to false-positives and false-negatives generated by the linkage algorithm.

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $\mathcal{C}_{\mathcal{L}}$ be the set of all cases

Table 10
An example of the ‘Collateral Events’ pattern.

CaseID	Activity	Timestamp
1234567	Adjust recovery cost	19/06/2014 12:15:18
1234567	Adjust recovery cost	19/06/2014 12:16:53
1234567	Email	19/06/2014 12:19:25
....
1234567	Pay assessor fee	19/06/2014 12:22:48
1234567	Adjust admin cost	19/06/2014 12:22:48

that exist in \mathcal{L} , $E_A \in AN$ be a set of *expected* activities (provided through domain knowledge) in log \mathcal{L} , and let $a_i \rightarrow a_j$ where $a \in AN$ and $a_i, a_j \in \mathcal{D}_a$ represent an ‘eventually follows’ relationship between activities such that if a_i occurs in log \mathcal{L} for some case $c \in \mathcal{C}_{\mathcal{L}}$ activity a_j must eventually occur in \mathcal{L} for the same case at some point after a_i . The eventually follows relationship is transitively closed and the set of eventually follows activity pairs must be provided from domain knowledge. The *scattered case* pattern may exist in \mathcal{L} , if, for ‘many’ cases $c \in \mathcal{C}_{\mathcal{L}}$:

- activities occurring in $\Pi_{act} \sigma_{case=c}(\mathcal{L})$ are a subset of E_A^3 ; and
- for some activities a_1, a_2 and $a_1 \rightarrow a_2$ or $a_2 \rightarrow a_1$, we find a_1 occurring in $\Pi_{act} \sigma_{case=c}(\mathcal{L})$ but **not** a_2 .

Pattern. Collateral Events

Description. This pattern captures the situation where, within an event log, there are multiple events which essentially refer to one particular process step within a case. This could result from (i) the event log having been constructed using data from multiple systems, each of which with their own way of recording the same process activity, (ii) the underlying system used is programmed to automatically fire a set of auxiliary events when a specific event (such as the payment of a bill to a supplier) occurs, and/or (iii) the audit trail-like nature of the log which records detailed low-level user activities (such as the opening and closing of a form) so that it is common to see duplication of events within a very short time period (e.g. when a user is switching back and forth between two forms). Regardless, as these events exist independently from each other, their labels are likely to be different and their timestamps may also differ slightly.

Real-life Example. We encountered this pattern in an Australian-based insurance organisation. As shown by the sample events in Table 10, these events all happened within a relatively short time period (a few minutes difference between events at most). At the same time, they all referred to a particular process step relating to finalising a payment to the insurance claim assessor. Nevertheless, due to the communication between multiple systems, this process step resulted in multiple events being recorded in the log. In fact, the events shown in Table 10 were derived from four different sources.

Affect. From a process mining perspective, this pattern tends to create unnecessary noise in the data. These ‘collateral’ events often refer to trivial low-level activities which do not contribute much to the extraction of meaningful insights about processes. Consequently, it is not uncommon for this pattern to result in highly-complex process mining results (e.g. overly complex models) that hinder the extraction of meaningful conclusions.

Data Quality Issue. I27 – Irrelevant data: event – The inclusion of multiple low-level activities results in the masking of the actual business process step through the inclusion of multiple events in the log.

Manifestation and Detection. This pattern’s signature is the log containing groups of activities with timestamps that are very close to each other (e.g. within seconds). Further, the labels of these activities may be very similar to each other or are of logical consequences from one another, and, from the perspective of a domain expert, these events do not represent a distinct process step. For example, the submission of an insurance claim via an online claim submission system may fire a set of notifications and emails to both the customer and the claim handler(s), resulting in the occurrence of multiple events within a very short period of time. The detection of this pattern requires the knowledge of domain experts who are familiar with both the functionality of the systems (from which the event log is extracted) and the overall steps of the process being analysed. However, from our

³ We note that the left hand side of the expression represents a mapping, not a set, but for readability we do not force the distinction.

experience, the existence of multiple events occurring within a short period of time is already an indication of the existence of this pattern. Furthermore, the existence of a very high number of events in most of the traces in an event log (e.g. more than 100 events) is also a good indication of the existence of this pattern.

Remedy. The remedy for this pattern involves the development of a knowledge base that records information about those activity names that, when occurring together within a short time period, should be merged into one single activity. The name of the merged activity should be specified in the knowledge base too, including the timestamp that should be used (e.g. take the earliest or the latest timestamp). Essentially, this remedy will reduce the total number of unique activities in an event log. While this remedy seems similar to the remedy for the ‘Synonomous Labels’ pattern (discussed later) the type of knowledge base required is quite different. The knowledge base for this pattern requires the knowledge of not just the domain expert, but also the knowledge of process analysts/system analysts who understand how process steps are created by the system used and subsequently stored as events in the logs, including the set of ‘collateral’ events that are triggered by the system. By contrast, the knowledge required to address the ‘Synonomous Labels’ pattern generally requires only the knowledge of a domain expert who understands the variations in the naming of activities.

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log and a be a dedicated timestamp column (typically *create time*). Let B be a partition of \mathcal{D}_{act} into logical business steps. The *collateral events* pattern may exist in \mathcal{L} if there exists a case c and a timestamp t such that there is an element $b \in B$ where:

- $|\sigma_{count} > \theta \chi(\sigma_{case=c \wedge a=t \wedge act \in b}(\mathcal{L}), \{a\}, count)| > 0$ and θ is some significance level.

Pattern. Polluted Label

Description. This pattern refers to the presence of a group of event attribute values that are structurally the same, yet are distinct from each other due to differences in the exact attribute values that further qualify the meaning of the value.

Real-life Example. We have encountered this pattern in two data sets that we have analysed. In an event log from an Australian insurance organisation, we found the existence of this pattern for the ‘Activity’ attribute as shown in Table 11. Here we see that the ‘fixed words’ are *Notification of Loss* and *Incident No..* The remainder of the string (noted with a series of ‘X’s) is mutable.

Affect. Where the pattern exists and affects the attribute that serves as the activity name, the process mining analysis will result in the discovered process models over-fitting the event log as there will be too many specific activities that should have firstly been abstracted out. In general, if this pattern exists and affects attributes such as case identifiers, activity names, and resource identifiers that are critical for process mining analyses, the quality of the results will be negatively impacted as the set of values will have cardinality greater than the number of allowed values in real-life.

Data Quality Issue. I15 – Incorrect data: activity name, I17 –

Table 11
Example of the ‘Polluted Label’ pattern.

CaseID	Activity	Timestamp
xxxx	Notification of Loss – XXXX Incident No. xxxxxx	xxxx-xx-xx xx:xx:xx
xxxx	Notification of Loss – XXXX Incident No. xxxxxx	yyyy-yy-yy yy:yy:yy
xxxx	Notification of Loss – XXXX Incident No. xxxxxx	zzzz-zz-zz zz:zz:zz
.....	Notification of Loss – XXXX Incident No. xxxxxx

Incorrect data: resource – The existence of this pattern in the log, particularly where it affects the activity name, effectively masks the underlying process step through the incorrect logging of the activity name.

Manifestation and Detection. The signature of this pattern is the attribute value being composed of a mixture of immutable boiler-plate text and mutable text that occurs at predictable points among the immutable text. So, if there are two known fixed words ($word_1, word_2$), the ‘Polluted Label’ pattern can be expressed with the following regular expression: $[.]^*(word_1)[.]^*(word_2)[.]*$. The existence of this pattern can be detected by either (i) checking the number of distinct values of each attribute (an unexpectedly high number of distinct values is a good indication of the existence of this pattern) followed by inspection of the values themselves to identify if there are those mutable and immutable parts within the values of that particular attribute, or (ii) using a (semi-)automated tool to cluster the values of that particular attribute, and within each cluster, to extract a regular pattern in a form that is similar to the one described above.

Remedy. The detection of the ‘Polluted Label’ pattern implies that the immutable word(s) of the pattern are known or can be determined. The mutable words can be removed and the immutable words rearranged to form one standard activity name. (Mutable words can be preserved as attribute values as required.)

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $AN_{\mathcal{L}}$ be a set of attribute names found in \mathcal{L} and \mathcal{D}_a be the set of all possible values of $a \in AN_{\mathcal{L}}$. Let $const: \mathcal{D}_a \rightarrow \Omega$ be an operator that returns the immutable component of $v \in \mathcal{D}_a$.

The *polluted label* event log imperfection pattern may exist in \mathcal{L} if:

- $|\sigma_{a=v}(\mathcal{L})| \ll |\sigma_{a=const(v)}(\mathcal{L})|$ (i.e. there are many more events that contain the immutable component of v than the actual value v .)

Pattern. Distorted Label

Description. This pattern refers to the existence of two or more values of an event attribute that are not an exact match with each other but have strong similarities syntactically and semantically. This pattern typically occurs through incorrect data entry or inadvertent changing of attribute values during data pre-processing or extraction.

Real-life Example. We encountered this pattern in one of the logs obtained from an Australian insurance company. In this log, we encountered the following entries in the activity attribute of the log:

- ‘a/w inv to cls.’ vs. ‘a/w inv to cls’ (note the existence of a dot in the first value)
- ‘XX - Further Information Required’ vs. ‘XX - Further Infomation Required’ (note the missing ‘r’ in the word ‘Information’ in the second value.

Affect. The impact of this pattern on process mining analysis is similar to that described for the ‘Polluted Label’ described earlier and the ‘Synonymous Label’ pattern (see later). That is, where this pattern applies to the activity name, the readability and validity of process mining results are negatively impacted. Also, performance analysis results will be affected through having two or more activities in the log that should be treated as the same activity, actually being considered as separate.

Data Quality Issue. I15 – Incorrect data: activity name – The activity label does not accurately reflect the process step that generated the log entry.

Manifestation and Detection. This pattern's signature is the existence of minor differences in the letters of some attribute values, e.g. ‘Follow-up a call’ vs. ‘follow-up a call’ where the only difference between the two values is the capitalization of the first letter. The presence of this pattern can be detected by either (i) selecting all the distinct values of each attribute in an event log, sorting them alphabetically and checking for multiple consecutive rows with values

that are similar but not exactly the same, or (ii) applying string similarity search, e.g. [21,23]. This pattern may exist if the string similarity search returns positive results.

Remedy. Where the pattern manifests as sporadic capitalization of letters, the attribute values can be transformed to use only lower-case or upper-case letters. However, if there are multiple factors that cause the existence of such a pattern, e.g. the (non-)existence of certain characters and the use of short-hand notations, then we need to use automated string similarity search to group those values that are syntactically similar, and replace them with a single value. Manual intervention may be needed to remove those values that were found to be syntactically similar but which are not similar semantically.

Side-effects of Remedy. The side-effects of the remedies recommended above should be minimal if manual intervention, as suggested above, is followed properly.

Indicative Rule. Let $\mathcal{L} \subseteq \mathcal{E}$ be an event log, $AN_{\mathcal{L}}$ be a set of attribute names found in \mathcal{L} and \mathcal{D}_a be the set of all possible values of $a \in AN_{\mathcal{L}}$. Let $sim: \mathcal{D}_a \times \mathcal{D}_a \rightarrow [0..1]$ be a function that returns the syntactic similarity between pairs of elements of \mathcal{D}_a . The *distorted label* event log imperfection pattern may exist in log \mathcal{L} if:

- $\sigma_s > \theta \prod_{a,a',s:sim(a,a')\sigma_{id \neq id' \wedge a \neq a'}} (\prod_{id:id,a:a}(\mathcal{L}) \bowtie \prod_{id':id,a':a}(\mathcal{L}))$ is not empty, where θ is some significance level.

Pattern. Synonymous Labels

Description. The ‘Synonymous Label’ pattern refers to a situation where there is a group of values (of certain attributes in an event log) that are syntactically different but semantically similar. This pattern is commonly encountered where an event log has been constructed by merging data from sources that do not share a common schema thus allowing the same real-world activity to be recorded with different labels in each source.

Real-life Example. We encountered this pattern when we attempted to merge a number of hospital logs for the purpose of comparing their patient flows. In Hospital A, the activity used to capture the first consultation of a patient with a doctor is named ‘DrSeen’. In Hospital B, the same activity is recorded as ‘Medical Assign’. These two labels have the same meaning but they are quite different syntactically. The issue of labels being represented differently has been widely studied in other literature, especially in the area of ‘label matching’ (e.g. [10]).

Affect. Where this pattern affects the activity name, the readability and validity of process mining results are negatively impacted due to the inclusion of ‘behaviours’ in the discovered process models that should have been ‘merged’ (as they involve activities that share the same semantics). This pattern may also impact the performance analysis results through having two or more activities in the log that should be treated as the same activity, actually being considered as separate.

Data Quality Issue. I22 – Imprecise data: event attributes – The existence of multiple names for the same attribute creates ambiguity in an event log.

Manifestation and Detection. This pattern's signature is the existence of multiple values of a particular attribute that seem to share a similar meaning but are nevertheless, distinct. For example, a particular resource is identified as ‘jsmith’ in some events and ‘Jason Smith’ in others. These two distinct values, in reality, refer to the same resource. Detection of this pattern may require the establishment of a knowledge base that stores the list of ‘acceptable’ values for each attribute in the log. Such a knowledge base will allow checking of the values of each attribute against the corresponding list of ‘acceptable’ values. The ‘Synonymous Label’ pattern exists when two or more attribute values correspond to the same value in the ‘acceptable’ list.

Remedy. Where syntactic differences between labels are minor, a text similarity search can be applied to group those events that have strong similarity in their labels, and then replace them with a pre-defined value. Where the syntactic differences are quite substantial

(e.g. ‘DrSeen’ vs. ‘Medical Assign’), the use of an ontology will allow replacement of the labels with just one value (either one of the synonyms can theoretically be used as the label substitute).

Side-effects of Remedy. A label could be incorrectly mapped to another label such that the meaning of the original label somewhat deviates from the original meaning (or intent) of the label. This is likely to happen when the ontology used is flawed or when two labels share strong syntactic similarities but differ semantically, e.g. ‘drawn vs. dawn’.

Indicative Rule. Let $\mathcal{L} \in \mathcal{E}$ be an event log. For a given attribute name $a \in AN$, \mathcal{D}_a is the set of all syntactically-distinct values of a , and $Sem \subseteq \mathcal{D}_a \times \mathcal{D}_a$ is the set of pairs of values of attribute a that are semantically similar. Sem is symmetric, irreflexive, and transitive. The *synonymous label* pattern may exist in \mathcal{L} if:

- $\sigma_{id \neq id' \wedge Sem(a,a')} (\prod_{id:id,a:a}(\mathcal{L}) \bowtie \prod_{id':id,a':a}(\mathcal{L}))$ is not empty.

Pattern. Homonymous Label

Description. This pattern describes a situation where an activity is repeated multiple times within a case (same activity label applied to each occurrence of the activity), but the interpretation of the activity, from a process perspective, differs across the various occurrences. This pattern is likely to occur when an audit trail log is used to construct an event log. An audit trail log typically contains low-level records of ‘things that have been executed’ by users. The influence of contextual factors (such as the number of times an activity has been repeated within the same case) on the meaning of the activity being recorded is not captured.

Real-life Example. We have seen this pattern occurring in a hospital log that we have analysed. A snippet of the log is provided in Table 12. Here the activity ‘Triage Assessment’ is recorded three times in the log. These events were captured from a form called the ‘Triage Assessment Form’. By taking into account contextual information such as the events preceding the second and the third occurrences of the ‘Triage Assessment’ activity, it became clear that the repeated activities should be interpreted differently.

The second ‘Triage Assessment’ happened after the patient was discharged. Therefore, it is unlikely that the second occurrence of this activity means that the patient was being triaged again. Also note that the third occurrence of the activity happened just four minutes after the second one. We can make an educated guess that the second and third occurrences of the ‘Triage Assessment’ activity actually refer to a medical officer *reviewing* the information captured in the triage form about one week after the patient was discharged to conduct further activities, e.g. giving the patient a follow-up call.

Affect. The existence of this pattern will result in discovered process models painting an incomplete picture of the process being analysed. Incomplete because the repeated activities, which actually have different meanings, are ‘grouped’ into one, thus ‘hiding’ certain process information. This is because most, if not all, process discovery algorithms do not treat repeated activities as separate activities in the discovered model.

Data Quality Issue. I2 – Imprecise data: activity name – The activity names are too coarse to reflect the different connotations associated with the recorded events.

Table 12
Example of the ‘Homonymous Label’ pattern.

CaseID	Activity	Timestamp
1234567	Triage Assessment	06/09/2013 12:33:17
1234567	Progress Note	06/09/2013 13:10:23
1234567	Discharged	06/09/2013 13:15:00
1234567	Triage Assessment	13/09/2013 07:24:36
1234567	Triage Assessment	13/09/2013 07:28:51

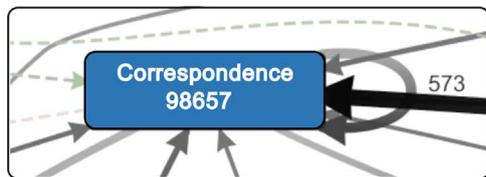


Fig. 4. A symptom of the existence of the ‘Homonymous Label’ pattern.

Manifestation and Detection. The signature of this pattern is the existence of an activity within a discovered process model that has many incoming arcs, often including a self-loop arc and arcs from other activities (as shown in Fig. 4).

In reality, each repetition of the ‘Correspondence’ activity may refer to correspondence of a different type (e.g. sending a quote, scheduling a meeting, etc.). The presence of these arcs alone is an indication (but not confirmation) of the existence of the pattern in the log. A second indicator is the ratio of the number of times a particular activity occurs in a log and the total number of cases in the log. A high ratio indicates that the activity is repeated many times within a case and requires investigation to check if different interpretations are possible for each repetition of the activity. Domain knowledge (as to different meanings being ascribed to repeated occurrences of the activity) is required to conclusively detect the existence of this pattern.

Remedy. This pattern can be addressed by explicitly relabeling repeated activities with a context sensitive name. Doing so requires firstly identifying the different contexts under which an activity can be repeated, developing a formula to assign the appropriate context to those homonymous activities, and then differentiating between them by adding context information into the label of those activities. For example, the second and third occurrences of the activity ‘Triage Assessment’ in the event log shown in Table 12 can be renamed to ‘Triage Assessment (Review)’, thus distinguishing between the actual triage assessment activity and the review of the triage assessment form activity.

Side-effects of Remedy. The decoupling of activity labels proposed in this remedy may result in too many distinct activity names, thus reducing the readability of the discovered process model and making the process mining analyses more complex and laborious.

Indicative Rule. Let $\mathcal{L} \in \mathcal{E}$ be an event log. For a given attribute name $a \in AN$, \mathcal{D}_a is the set of all syntactically-distinct values of a , and $\mathcal{H} \subseteq \mathcal{D}_a$ is the set of homonyms where \mathcal{H} may be populated using process context information. The *homonymous label* pattern may exist

in \mathcal{L} if:

- $\sigma_{a \in \mathcal{H}}(\mathcal{L})$ is not empty.

6. Case study

We have applied the data imperfection patterns in a process mining case study with an Australian-based hospital analysing their emergency department treatment process for patients with chest-pain symptoms.

6.1. Preliminary

The raw data from the hospital consisted of four tables. The *encounter* table contained a summary of 2136 patient encounters that occurred over a certain 18-month period. This table included patient arrival times, discharge times, discharge destinations, encounter types, and other non-identifying patient data. The data in the *encounter* table was recorded in a typical key-value format with each row containing the details of a specific encounter. The *emergency* table contained events recorded from the system used to manage key Emergency Department activities, e.g. triage time and assignment of a patient to a doctor. The *emergency* table was provided in an event-log format and contained 16,587 events. The *clinical* table was produced from the hospital system which recorded all clinical events. Data from this table was very fine-grained (almost like an audit-trail), was formatted as an event-log and contained 189,994 events. The *orders* table contained records of all orders that had been placed in relation to a particular encounter, e.g. blood test orders and X-ray imaging orders. The data in this table was structured as an event-log and consisted of 8974 events.

6.2. Event Log Imperfection Patterns observed in the hospital data

Unanchored Events – All tables. To remove ambiguity and to ensure events could be properly ordered, the first thing we did was reformat the timestamp values in the raw data to our standard format using a spreadsheet program. Had we not anticipated the possible existence of the *Unanchored Event* pattern, it would have been easy for the timestamp values in our data to be interpreted incorrectly. We experimented with loading one of the tables into the process mining tool, Disco (www.fluxicon.com/disco). Disco initially loads a sample of the data (normally the first 1000 events) and allows the user to allocate each field of the data to an event log attribute (e.g. case identifier,

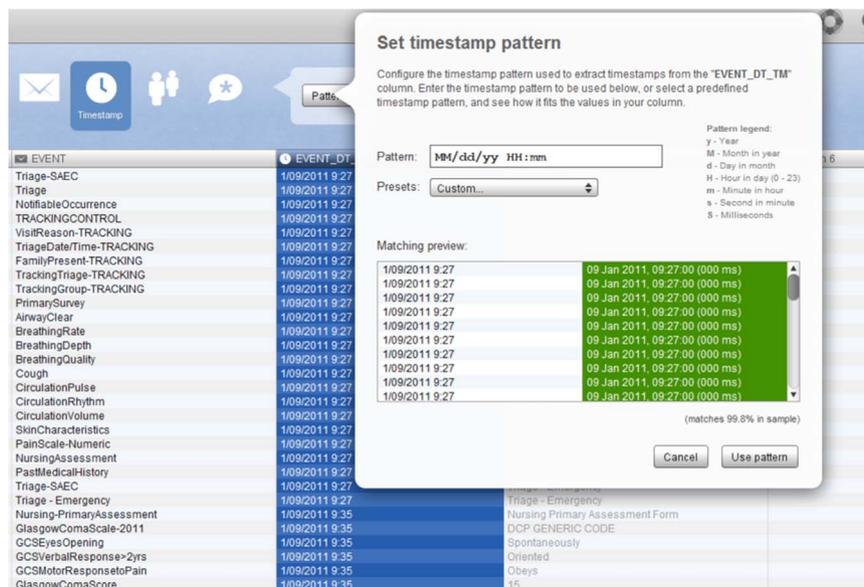


Fig. 5. A potentially-undetected mistake in the specification of the timestamp format.

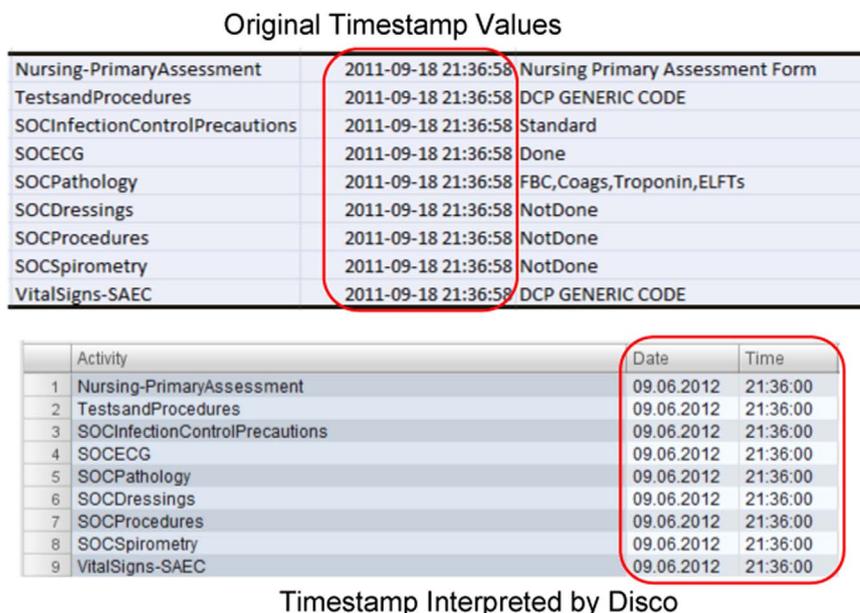


Fig. 6. An example of how the ‘Unanchored Event’ pattern could have manifested in our data set.

activity name, resource identifier, and timestamp). When a field is tagged as being ‘timestamp’, users can specify the timestamp format. Fig. 5 shows that the default timestamp format was month-day-year. However, the timestamp format in the log was day-month-year. Because the first 1000 events in the log contain dates with timestamp values that can be validly interpreted using either format (i.e. the date ranges from 1 September to 9 September), the Disco tool would allow the month-day-year selection. When the complete event log was imported, Disco attempted to interpret those timestamp values that could not be validly interpreted in the month-day-year into another timestamp. For example, Fig. 6 shows how the original date of 18 September 2011 was incorrectly interpreted as 6th September 2012. Because we had anticipated such an issue, we avoided this potential problem.

Elusive Case – Encounter table. In this case study, we intended to analyse end-to-end patient flow from the arrival of a patient at the emergency department (ED) to their discharge from the hospital. When ED patients require inpatient treatment, they are admitted to hospital under a new encounter identifier. To properly track end-to-end patient flow, we needed to be able to link an ED encounter identifier with its corresponding hospital encounter identifier for patients who were later admitted to the hospital. (Note that a row in the *encounter* table does not explicitly link an ED encounter with its corresponding hospital encounter). Further, the events recorded in the other tables (i.e. the *emergency*, *clinical*, and *order* tables) were also linked only to their respective encounter identifier. Therefore, the encounter identifier attribute on its own cannot be used as the case identifier.

Our data also contained each patient's medical record number (MRN) which uniquely identifies a patient. However, it is not uncommon for a patient to have multiple ED encounters. Therefore, if we had used MRN as the case identifier, multiple patient flows (as per our intended definition of a patient flow) may be captured within a case. Thus, by itself, the MRN could not be used as the case identifier either. After consideration, none of the other attributes could be used as case identifier.

This situation indicates the presence of the *Elusive Case* pattern because none of the attributes that are common to all events in all of the tables can be used to group events into their respective cases. The pattern remedy calls for correlating information in the event log with information from another source. In this instance, we used attributes from the *encounter* table including the encounter type, the discharge

destination, the admission time, and the discharge time as well as the MRN to create a heuristic rule which states that: (1) for each row in the *encounter* table where (i) the encounter type is ‘emergency department’ encounter and (ii) the discharge destination is ‘discharge to the same hospital’ (we called this row the ‘ED’ row), (2) find within the same *encounter* table another row where (i) the encounter type is ‘hospital’ encounter, (ii) the admission timestamp is the same day as the discharge timestamp of the ‘ED’ row, and (iii) the MRN value of this row is the same as the MRN value of the ‘ED’ row (we called this row the ‘hospital row’). If a ‘hospital row’ is found for a given ‘ED row’, replace the encounter identifier of the ‘hospital row’ with the encounter identifier of the ‘ED row’. The application of this rule remedied the *Elusive Case* issue by making the ED encounter identifier the case identifier. This remedy did not completely resolve the issue as there were a few rows in the tables that we could not match properly. In two encounters with the same MRN, emergency and hospital encounters were registered with exactly one day difference. We assumed this was a data entry error (possibly *Inadvertent Time Travel*) and considered these encounters to be one case and merged accordingly. One encounter was removed as it was the only hospital encounter (other than the ones with the assumed data entry error) that did not have a corresponding emergency encounter registered on the same day.

Form-based Event Capture – Clinical Table. The *clinical* table contained an abnormally high number of events in comparison to the number of events in other tables. We further observed that there were many events with the same case identifier and timestamp. Further, the activities for those events with the same case identifier and timestamps were often repeated (i.e. there were occurrences of more or less similar sequences of events with the same timestamp and case identifier, e.g. *PrimarySurvey*, followed by *AirwayClear*, *BreathingRate*, *BreathingDepth*, ...).

This fits the criteria for the existence of the *Form-based Event Capture* in the data set which was confirmed when we matched the activities in the log with the activities on the actual electronic forms used by the clinicians.

We remedied this problem by collapsing all events that (i) were likely to belong to the same form and (ii) had been recorded as a result of the same action triggered by the user (e.g. exiting a form or clicking on a ‘Save Form’ button) into one event (whenever possible without inducing any loss of important events) or multiple events (if necessary in order to retain important events). The latter often occurred when a

C	D	E
EVENT	EVENT_DT_TM	EVENT_RESULT
Triage-SAEC	01/09/2011 09:27:34	Triage Form
Triage	01/09/2011 09:27:34	DCP GENERIC CODE
NotifiableOccurrence	01/09/2011 09:27:34	No
TRACKINGCONTROL	01/09/2011 09:27:34	DCP GENERIC CODE
VisitReason-TRACKING	01/09/2011 09:27:34	ChestPain
TriageDate/Time-TRACKING	01/09/2011 09:27:34	0:2011090109200000:0.000000:0:0
FamilyPresent-TRACKING	01/09/2011 09:27:34	No
TrackingTriage-TRACKING	01/09/2011 09:27:34	4-Semi-Urgent
TrackingGroup-TRACKING	01/09/2011 09:27:34	SAEC
PrimarySurvey	01/09/2011 09:27:34	DCP GENERIC CODE
AirwayClear	01/09/2011 09:27:34	Yes
BreathingRate	01/09/2011 09:27:34	Normal
BreathingDepth	01/09/2011 09:27:34	Normal
BreathingQuality	01/09/2011 09:27:34	Normal
Cough	01/09/2011 09:27:34	Nil
CirculationPulse	01/09/2011 09:27:34	Normal
CirculationRhythm	01/09/2011 09:27:34	Regular
CirculationVolume	01/09/2011 09:27:34	Normal
SkinCharacteristics	01/09/2011 09:27:34	Normal
PainScale-Numeric	01/09/2011 09:27:34	2
NursingAssessment	01/09/2011 09:27:34	DCP GENERIC CODE
PastMedicalHistory	01/09/2011 09:27:34	Ptpresents/coconstant2/10L)sidedchestpainfor1/52 .Notreproducible.Nilhxfsame.Notrelievedwithpositioning.Deniesothersymptoms.

Fig. 7. ‘Form-based Event Capture’ snippet.

user closed two or more forms almost simultaneously (e.g. through a ‘close all’ functionality).

We exploit the SQL ‘Group By’ operator to address this issue. For each case, we grouped activity names *by timestamp* on the basis that form-based events would have been recorded with the same timestamp. Thus, for each timestamp of a case, we obtain a list of activities that occurred for that particular timestamp. We note that in this example, drawn from a single encounter identifier, the opening of the form, as well as the transition between data tabs (subforms) can be seen in the log (see Fig. 7). Collapsing the log events can thus be performed at either the form level or the tab (subform) level. Having performed the above cleaning, we managed to reduce the number of events in the *clinical* table from 189,994 to 14,778.

Collateral Events – Order and Clinical Tables. The *Collateral Events* pattern was noticed within the Clinical and the Orders tables. In the Clinical table, a single event recording a set of blood tests is related to multiple events (one per blood test) in the Orders table. Table 13 shows an example of this pattern.

This situation needs to be addressed, otherwise, activities in the log may be interpreted as being ‘fragmented’ while in reality, this is not the case. These fragmented and scattered activities can be safely seen as

Table 13

An example of the ‘Collateral Events’ pattern in the hospital log.

CaseID	Activity	Timestamp	Table	Detail
1234567	SOCPathology	21/09/2011 09:20:53	Clinical	FBC,INR,Troponin, ELFTs
1234567
1234567	FBC	21/09/2011 09:25:58	Orders	21/09/2011 09:25 FBC
1234567	ELFTs	21/09/2011 09:25:58	Orders	21/09/2011 09:25 ELFTs
1234567	Troponin	21/09/2011 09:25:58	Orders	21/09/2011 09:25 Troponin
1234567	INR	21/09/2011 09:25:58	Orders	21/09/2011 09:25 INR

one activity that was ‘mildly interrupted’ during its execution. We addressed this pattern by (i) constructing a knowledge base that listed all the different blood tests that could be ordered, (ii) using the knowledge base to merge all blood test orders with the same timestamp into a single activity called ‘Blood Tests (Ordered)’, and (iii) removing the ‘SOCPathology’ activity from the log as it only informed us that

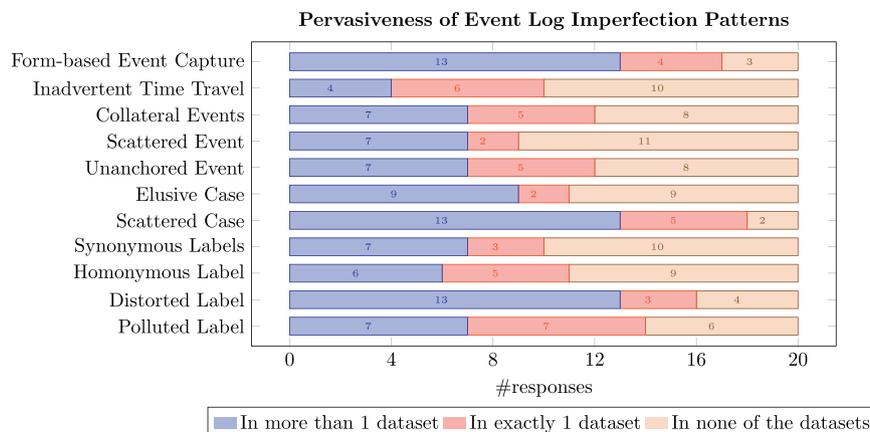


Fig. 8. Pervasiveness of Event Log Imperfection Patterns.

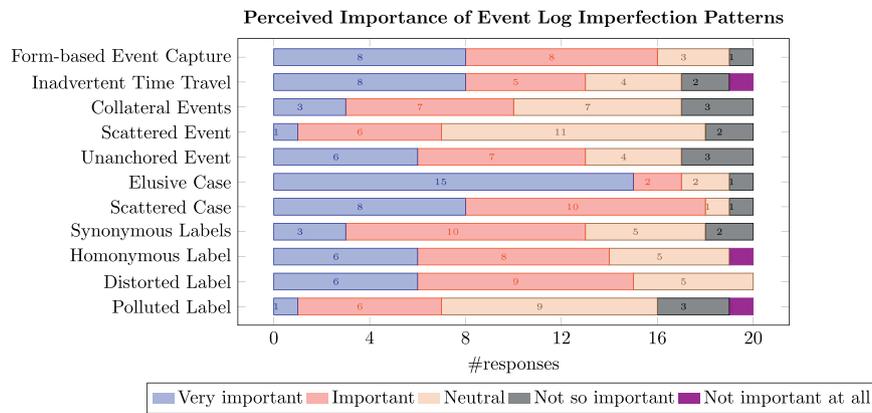


Fig. 9. Perceived importance of Event Log Imperfection Patterns.

blood tests were to be requested whereas the records in the Orders table show when the orders were actually made.

Scattered Event – Clinical Table. The *Scattered Event* pattern was detected in the Clinical table in activities relating particularly to surgical procedures. The pattern was detected by firstly observing that the activities ‘SN-Proc-StartTime’ and ‘SN-Proc-StopTime’ contained timestamp-like values as an attribute. Further investigation showed that the ‘SN-Surgical-Procedures’ and ‘SN-Proc-Primary-Surgeon’ activities also contained information useful in reconstructing a single activity (as shown in Table 14).

The pattern was remedied by (i) building a knowledge base of surgical procedures identified by a 5 digit code plus text description from the attribute values of the ‘SN-Surgical-Procedures’ activity (ii) for each occurrence of the ‘SN-Surgical-Procedures’ activity in the Clinical table, using the attribute values of the ‘SN-Surgical-Procedures’ and ‘SN-Proc-StopTime’ with the same case identifier and timestamps to reconstruct the surgical procedure event record.

Scattered Case – Order Table. This pattern was illustrated in the hospital data used to describe the Scattered Case pattern earlier.

Homonymous Label – Clinical Table. This pattern was illustrated in the hospital data used to describe the Homonymous Label. Note that the pattern also affected other activities (e.g. medical note, nursing primary assessment) where there was the initial event (observation and recording) with subsequent accesses to review/update the record.

6.3. Occurrence in other data sets

As shown in Table 15, the patterns were also observed in other data sets that we have analysed. The Hospital2 data set was derived from another Australian hospital and includes 884 cases of patients pre-

Table 14 An example of the ‘Scattered Event’ pattern in the hospital log.

Scattered events			
CaseID	Activity	Timestamp	Detail
1234567	SN-Surgical-Procedures	8/09/2011 09:13:25	38306Transluminalstentinsertion
1234567	SN-Proc-Primary-Surgeon	8/09/2011 09:13:25	*Anonymised*
1234567	SN-Proc-StartTime	8/09/2011 09:13:25	0:2011090709340000:0.000000:0:0
1234567	SN-Proc-StopTime	8/09/2011 09:13:25	0:2011090710010000:0.000000:0:0
Reconstructed event			
Case ID	Activity	Timestamp	Resource
1234567	Transluminal stent insertion	7/09/2011 10:01:00	*Anonymised*

sending at the Emergency Department with multiple trauma injuries.

Both the Insurer1 and Insurer2 data sets deal with insurance claims handling. The Insurer1 data set consisted of approximately 500,000 distinct work items (events) and the Insurer2 data set included 17,750 cases. We note that, as assessed against the log maturity scale in [45], the hospital logs rate as 2-star and exhibit many of the characteristics of logs derived from HIS as described in [25]. The insurer logs were of a 3-4-star log maturity rating yet even these logs exhibit many of the imperfection patterns described in this paper. The Hospital1 data set was chosen as the exemplar for the patterns largely because it exhibited the majority of the patterns described in this paper.

7. Evaluation

A user evaluation of the event log imperfection pattern set described in this paper was conducted via a questionnaire to determine 1) the pervasiveness of the patterns in event logs used by respondents in process mining analyses, 2) the importance attached by respondents to recognising the existence of each pattern in a process mining analysis, and finally, 3) the perceived usefulness of the pattern approach in characterising event log quality issues. The questionnaire was targeted at process mining researchers and practitioners and was distributed via a number of channels including (i) IEEE Task Force on Process Mining; (ii) LinkedIn Process Mining group; and (iii) contacts in universities, research institutions, companies and government departments known to be active in the field of process mining. The questionnaire preface outlined the background and motivation for the research. Participants were asked to answer a number of questions regarding their demographics. The questionnaire presented each of the 11 event log imperfection patterns described in this paper with respondents being asked to indicate how many times they had seen the pattern in data sets they had analysed, and the importance they attached to recognising the existence of the pattern given its potential impact on a process mining analysis. The perceived importance is measured using a Likert-like scale (Very important, Important, Neutral, Not so important, Not important at all).

7.1. Evaluation results

Of the 23 people that started the questionnaire, a total of 20 people completed it, which translates to a response rate of 87% including 15 academics/researchers and 5 practitioners with varying levels of data mining and process mining experience. 90% of respondents indicated they had worked in a role that involved analysing data or data manipulation for 1 or more years with 45% of respondents indicating more than 5 years involvement.

Respondents were ‘Familiar’ with both data mining and process mining, predominantly use WEKA as a data mining tool (R and Rapid Miner were also popular tools) with ProM and Disco being the most

Table 15
Occurrence of event log imperfection patterns in other data sets.

	Form-based event capt.	Inadvertent time travel	Collateral events	Scattered event	Unanchored event	Elusive case	Scattered case	Synonymous labels	Homonymous labels	Distorted label	Polluted label
Data sets											
Hospital1	✓		✓	✓	✓	✓	✓	✓	✓		
Hospital2	✓		✓	✓	✓	✓	✓	✓			
Insurer1		✓	✓		✓		✓	✓		✓	✓
Insurer2			✓		✓		✓	✓			✓

popular process mining tools among respondents. The average number of patterns observed by any respondent being 7 (of the 11 patterns). All patterns have been seen by at least 45% of respondents with 4 patterns having been observed by at least 70% of the respondents. The most frequently observed patterns were *Scattered Case*, *Form-based Event Capture* and *Distorted Label* (seen by 90%, 85% and 80% respectively of respondents). The *Scattered Event*, *Inadvertent Time Travel* and *Synonymous Labels* patterns were least frequently observed (seen by only 45%, 50% and 50% respectively of respondents).

All patterns except the *Polluted Label* and *Scattered Event* patterns were rated as ‘Important’ or ‘Very important’ by at least 50% of the respondents. All respondents rated the collection of patterns as being ‘Useful’ (55%) or ‘Very useful’ (45%) in terms of characterising event log quality issues.

While the small sample size precludes statistically significant conclusions being drawn from the results, the responses indicate agreement with the notion that the patterns exist as a real phenomenon (according to the “rule of three” [4]) and that the patterns are important and useful.

8. Conclusion

Contemporary logs typically have many issues that need to be resolved before they can serve as input for analysis. They may contain imprecise or even incorrect data, certain important events may not have been recorded (e.g. certain transaction types may not be recorded), or they contain data that needs to be interpreted carefully (e.g. timestamps). While an event log may exhibit a number of specific problems, there are many issues that recur frequently and which can be resolved using a variety of known remedies. To address the cleaning of a log in a systematic manner, this paper documents a collection of typical problems one may encounter in event logs as well as associated remedies that can be used to rectify them. This documentation takes the form of patterns, collectively providing a repository of knowledge to deal with data imperfections. While the patterns are informed by problems that may manifest themselves in the context of data mining, they are targeted to the field of process mining and the specific issues that may arise there. The patterns remain agnostic of any subsequent form of analysis (e.g. process discovery or process conformance) and can thus be applied in the first stages of a process mining trajectory. The patterns were validated using a number of event logs, with varying degrees of ‘maturity’, from practice. These event logs demonstrate that (i) the patterns identified do indeed occur, (ii) that higher ‘maturity’ ratings do not guarantee that logs will be free of imperfections, and (iii) event logs require attention before embarking on specific analysis tasks.

We recognise that, as the patterns described in this paper were drawn from only 4 data sets, it is likely that more such patterns will be identified through analysis of other data sets. We do not claim an exhaustive listing of event log imperfection patterns, but do claim that the *pattern plus recommended remedy* approach is a significant contribution to the systematic preparation of logs for process mining analyses. This view was supported by the respondents to the evaluation questionnaire. Despite the small sample size, the majority of respondents had significant work experience in analysing data and approximately half of the respondents indicated they had analysed more than 20 event logs.

This paper directly addresses several of the challenges facing process mining raised in the Process Mining Manifesto [45]. In particular, from a data quality perspective, C1: *Finding, merging and cleaning event data* and C2: *Dealing with complex event logs having diverse characteristics*. Furthermore, the use of data imperfection pattern-based approach to cleaning event logs can be seen as the first step towards addressing two other challenges listed in the manifesto: C10 and C11 which deal with improving usability and understandability, respectively, of process mining to non-experts. At present, there

is a lack of agreed systematic approach (and its related supports) to undertaking an end-to-end process mining project (from data pre-processing, analysis, and results interpretation).

We argue that our pattern-based approach contributes to the development of this systematic methodology on the basis that (i) event log cleaning is a necessary first step towards achieving high quality input, (ii) the patterns can be applied in a (semi-)automated manner, and (iii) the patterns are independent of the purpose of the analysis.

There are a number of avenues for future work. One of these involves the development of a suite of metrics describing the pervasiveness of the patterns in a log. The patterns affect a log at different levels including attributes, events and cases, thus the various pervasiveness metrics would determine the number of attributes, events and cases affected by individual patterns and the pattern collection overall. (Note that it is then possible to determine the fraction of the log impacted by the patterns by dividing the count of the respective log elements affected by the total number of the respective element in the log.) Another avenue is the development of a formal framework that can be used to specify both existing patterns and as yet undescribed patterns. The metrics and framework will constitute the basis of automated support in detecting the presence of patterns and applying suitable remedies to fix any of the problems detected. The metrics may provide guidance with remedy selection and prioritisation. Additional possible future work is the documentation of pattern collections for preparing data for specific analysis tasks in the area of process mining and further substantiation of the approach through feedback from more researchers and practitioners.

Acknowledgements

The research for this paper was supported by Australian Centre for Health Services Innovation (AusHSI) Stimulus Grant – Project Reference no. SG0009-000450.

References

- [1] ISO/IEC 25010:2011: Systems and software engineering – Systems and software product Quality Requirements and Evaluation (SQuaRE) – System and software quality models, 2011.
- [2] C. Alexander, *The Timeless Way of Building*, Oxford University Press, 1979.
- [3] C. Alexander, S. Ishikawa, M. Silverstein, *A Pattern Language: Towns, Buildings, Constructions*, Oxford University Press, 1977.
- [4] D. Alur, D. Malsk, J. Crupi, *Core J2EE Patterns (Core Design Series): Best Practices and Design Strategies*, Sun Microsystems Inc, 2003.
- [5] C. Batini, M. Scannapieco, *Data Quality: Concepts, Methodologies and Techniques*, Springer, 2006.
- [6] J.C. Bose, R.S. Mans, W.M.P. van der Aalst, Wanna improve process mining results? It's high time we consider data quality issues seriously, in: *Proceedings of the IEEE CIDM*, Singapore, IEEE, 2013, pp. 127–134.
- [7] J.C. Bose, R.S. Mans, W.M.P. van der Aalst, Wanna improve process mining results? It's high time we consider data quality issues seriously, Technical Report BPM-13-02, BPM Center Report, (<http://bpmcenter.org/wp-content/uploads/reports/2013/BPM-13-02.pdf>), 2013.
- [8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, *Pattern-oriented software architecture: a system of patterns Vol 1*, Wiley, 1996.
- [9] R. Conforti, G. Fortini, M. La Rosa, A.H.M. ter Hofstede, Noise filtering of process execution logs based on outliers detection, *Trans. Knowl. Data Eng. PP (99)*, 2016, 1-1.
- [10] M. Ehrig, A. Koschmider, A. Oberweis, Measuring similarity between semantic business process models, in: *Proceedings of the Fourth APCCM*, vol. 67 of APCCM '07, ACS, 2007, pp. 71–80.
- [11] A.K. Elmagarmid, P.G. Ipeirotis, V.S. Verykios, Duplicate record detection: a survey, *Trans. Knowl. Data Eng. 19 (1)* (2007) 1–16.
- [12] I.P. Fellegi, A.B. Sunter, A theory for record linkage, *J. Am. Stat. Assoc. 64 (328)* (1969) 1183–1210.
- [13] M. Fowler, *Analysis Patterns: Reusable Object Models*, Addison-Wesley, Reading, Massachusetts, 1997.
- [14] M. Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2002.
- [15] E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns: Elements of Object-Oriented Software*, Addison-Wesley, 1994.
- [16] L. Ghionna, G. Greco, A. Guzzo, L. Pontieri, Outlier detection techniques for process mining, in *ISMIS*, volume 4994 of LNCS, Springer, 2008, pp. 150–159.
- [17] T. Gschwandtner, J. Gärtner, W. Aigner, S. Miksch, A taxonomy of dirty time-oriented data, in *Multidisciplinary Research and Practice for Information Systems*, volume 7465 of LNCS, Springer, 2012, pp. 58–72.
- [18] C.W. Günther, *Process Mining in Flexible Environments (Ph.D thesis)*, Technische Universiteit Eindhoven, 2009.
- [19] G. Hohpe, B. Woolf, *Enterprise Integration Patterns: Designing, Building and Deploying Messaging Solutions*, Addison-Wesley, 2003.
- [20] W. Kim, B. Choi, E. Hong, S. Kim, D. Lee, A taxonomy of dirty data, *Data Min. Knowl. Discov. 7 (1)* (2003) 81–99.
- [21] C. Klinkmüller, I. Weber, J. Mendling, H. Leopold, A. Ludwig, Increasing recall of process model matching by improved activity label matching, in: *Proceedings of BPM*, volume 8094 of LNCS, Springer, 2013, pp. 211–218.
- [22] M. Laverdiere, A. Mourad, A. Hanna, M. Debbabi, Security design patterns: survey and evaluation, in: *IEEE Canadian Conference on Electrical and Computer Engineering*, 2006, pp. 1605–1608.
- [23] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, *Dokl. Akad. Nauk SSSR 163 (4)* (1965) 845–848.
- [24] R.S. Mans, W.M.P. van der Aalst, R.J.B. Vanwersch, Process mining in healthcare – opportunities beyond the ordinary, Technical Report BPM-13-26, Eindhoven University of Technology, (<http://bpmcenter.org/wp-content/uploads/reports/2013/BPM-13-26.pdf>), 2013.
- [25] R.S. Mans, W.M.P. van der Aalst, R.J.B. Vanwersch, A.J. Moleman, Process mining in healthcare: Data challenges when answering frequently posed questions, in: *Process Support and Knowledge Representation in Health Care*, volume 7738 of LNCS, Springer, 2012, pp. 140–153.
- [26] M.A. Marsan, G. Conte, G. Balbo, A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems, *ACM Trans. Comput. Syst. 2 (2)* (1984) 93–122.
- [27] L. Maruster, A.J.M.M. Weijters, W.M.P. van der Aalst, A. van den Bosch, A rule-based approach for process discovery: dealing with noise and imbalance in process logs, *Data Min. Knowl. Discov. 13 (2006)* 67–87.
- [28] A. Moore, M. Collins, D. Mundie, R. Ruele, D. McIntire, Pattern-based design of insider threat programs, Technical Report CMU/SEI-2014-TN-024, Software Engineering Institute, 2014.
- [29] A. Moore, D. McIntire, D. Mundie, The justification of a pattern for detecting intellectual property theft by departing insiders, Technical Report 732, Software Engineering Institute, 2013.
- [30] A. Moore, M. Hanley, D. Mundie, A pattern for increased monitoring for intellectual property theft by departing insiders, in: *Proceedings of the Conference on Pattern Languages of Programs*, 2011, pp. 1–17.
- [31] H. Müller, Problems, methods, and challenges in comprehensive data cleansing, *Informatik-Berichte*, Professoren des Inst. Für Informatik, 2005.
- [32] R.E. Neapolitan, *Learning Bayesian Networks*, Prentice-Hall Inc., Upper Saddle River, NJ, USA, 2003.
- [33] H.B. Newcombe, J.M. Kennedy, S.J. Axford, A.P. James, Automatic linkage of vital records, *Science 130 (3381)* (1959) 954–959.
- [34] P. Oliveira, F. Rodrigues, P. Henriques, A formal definition of data quality problems, in: *Proceedings of the MIT IQ Conference*, MIT, 2005.
- [35] A. Partington, M.T. Wynn, S. Suriadi, C. Ouyang, J. Karnon, Process mining for clinical processes: a comparative analysis of four Australian hospitals, *ACM Trans. Manag. Inf. Syst. 5 (4)* (2015) 19:1–19:18.
- [36] M.V.S. Prasad, C.K. Prasad, B. Rambabu, Appraisal of efficient techniques for online record linkage and deduplication using q-gram based indexing, *Int. J. Comput. Sci. Mob. Comput. 3 (5)* (2014) 404–414.
- [37] E. Rahm, H. Do, Data cleaning: problems and current approaches, *IEEE Bull. Tech. Comm. Data Eng. 23 (4)* (2000) 3–13.
- [38] D. Riehle, H. Züllighoven, Understanding and using patterns in software development, *Theory Pract. Object Syst. 2 (1)* (1996) 3–13.
- [39] A. Rogge-Solti, R.S. Mans, W.M.P. van der Aalst, M. Weske, Repairing event logs using timed process models, in: *OTM 2013 Workshops*, volume 8186 of LNCS, Springer, 2013, pp. 705–708.
- [40] S. Suriadi, R.S. Mans, M.T. Wynn, A. Partington, J. Karnon, Measuring patient flow variations: A cross-organisational process mining approach, in: *AP-BPM, LNBIP, Springer*, 2014, pp. 43–58.
- [41] S. Suriadi, M.T. Wynn, C. Ouyang, A.H.M. ter Hofstede, N. van Dijk, Understanding process behaviours in a large insurance company in Australia: A case study, in: *CAiSE*, volume 7908 of LNCS, 2013, pp. 449–464.
- [42] T. Taibi, D. Ngo, Formal specification of design patterns – a balanced approach, *J. Object Technol. 2 (4)* (2003) 127–140.
- [43] P. van Bommel, A.H.M. ter Hofstede, T.P. van der Weide, Semantics and verification of object-role models, *Information Systems*, 16(5), 1991, pp. 471–495.
- [44] W.M.P. van der Aalst, *Process Mining – Discovery, Conformance and Enhancement of Business Processes*, Springer, 2011.
- [45] W.M.P. van der Aalst et al. Process mining manifesto, in: *Florian Daniel et al. (ed.), BPM 2011 Workshops (1)*, volume 99 of LNBIP, Springer-Verlag, 2011, pp. 169–194.
- [46] W.M.P. van der Aalst, A.H.M. ter Hofstede, Workflow patterns put into context, *Softw. Syst. Model. 11* (2012) 319–323.
- [47] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, A.P. Barros, Workflow patterns, *Distrib. Parallel Databases 14 (3)* (2003) 5–51.
- [48] Y. Wand, R. Wang, Anchoring data quality dimensions in ontological foundations, *Commun. ACM 39 (11)* (1996) 86–95.
- [49] R. Wang, V.C. Storey, C.P. Firth, A framework for analysis of data quality research, *IEEE Trans. Knowl. Data Eng. 7 (4)* (1995) 623–640.
- [50] R.Y. Wang, D.M. Strong, Beyond accuracy: what data quality means to data consumers, *J. Manag. Inf. Syst. 12 (4)* (1996) 5–33.
- [51] A.J.M.M. Weijters, W.M.P. van der Aalst, Rediscovering workflow models from event-based data using little thumb, *Integr. Comput.-Aided Eng. 10 (2)* (2003) 151–162.
- [52] S. Yacoub, H. Ammar, *Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems*, Addison-Wesley Profession, United States, 2004.