DETECTING WORKAROUNDS IN BUSINESS PROCESSES – A DEEP LEARNING METHOD FOR ANALYZING EVENT LOGS

Conference Paper · May 2020

CITATION
1

READS
458

5 authors, including:

Sven Weinzierl
Friedrich-Alexander-University of Erlangen-Nürnberg
21 PUBLICATIONS **39** CITATIONS
SEE PROFILE

Verena Wolf
Universität Paderborn
18 PUBLICATIONS **83** CITATIONS
SEE PROFILE

Tobias Pauli
Friedrich-Alexander-University of Erlangen-Nürnberg
10 PUBLICATIONS **12** CITATIONS
SEE PROFILE

6-15-2020

DETECTING WORKAROUNDS IN BUSINESS PROCESSES—A DEEP LEARNING METHOD FOR ANALYZING EVENT LOGS

Sven Weinzierl
Friedrich-Alexander-Universität Erlangen-Nürnberg, sven.weinzierl@fau.de

Verena Wolf
Paderborn University, verena.wolf@uni-paderborn.de

Tobias Pauli
Friedrich-Alexander-Universität Erlangen-Nürnberg, tobias.pauli@fau.de

Daniel Beverungen
Paderborn University, daniel.beverungen@upb.de

Martin Matzner
Friedrich-Alexander-Universität Erlangen-Nürnberg, martin.matzner@fau.de

DETECTING WORKAROUNDS IN BUSINESS PROCESSES—A DEEP LEARNING METHOD FOR ANALYZING EVENT LOGS

Research paper

Sven, Weinzierl, Chair for Digitial Industrial Service Systems, Institute of Information Systems, Friedrich-Alexander-Universität Erlangen-Nürnberg, Nuremberg, Germany, sven.weinzierl@fau.de

Verena, Wolf, Chair of Business Information Systems, Faculty of Business Administration and Economics, Paderborn University, Paderborn, Germany, verena.wolf@upb.de

Tobias, Pauli, Chair for Digitial Industrial Service Systems, Institute of Information Systems, Friedrich-Alexander-Universität Erlangen-Nürnberg, Nuremberg, Germany, tobias.pauli@fau.de

Daniel, Beverungen, Chair of Business Information Systems, Faculty of Business Administration and Economics, Paderborn University, Paderborn, Germany, daniel.beverungen@upb.de

Martin, Matzner, Chair for Digitial Industrial Service Systems, Institute of Information Systems, Friedrich-Alexander-Universität Erlangen-Nürnberg, Nuremberg, Germany, martin.matzner@fau.de

Abstract

Business processes performed in organizations often deviate from the abstract process models issued by designers. Workarounds that are carried out by process participants to increase the effectiveness or efficiency of their tasks are often viewed as negative deviations from prescribed business processes, interfering with their efficiency and quality requirements. But workarounds might also play an important role in identifying and re-structuring inefficient, dysfunctional, or obsolete processes. While ethnography or critical incident techniques can serve to identify how and why workarounds emerge, we need automated methods to detect workarounds in large data sets. We set out to design a method that implements a deep-learning-based approach for detecting workarounds in event logs. An evaluation with three public real-life event logs exhibits that the method can identify workarounds best in standardized business processes that contain fewer variations and a higher number of different activities. Our method is one of the first IT artifacts to bridge boundaries between the complementing research disciplines of organizational routines and business processes management.

Keywords: Workarounds, Business Processes, Deep Learning, Process Mining.

1 Introduction

Aligning information systems (IS) and business processes is challenging, since processes frequently change, for example, by adopting new digital technologies (Beerepoot and Weerd, 2018). Previous research has focused on investigating the implementing phase of IS (Ignatiadis and Nandhakumar, 2009). However, analyzing the post-implementation phase of IS is equally important, since employees frequently perform workarounds to overcome, bypass, or minimize the impact of obstacles or structural constraints (Alter, 2015). Workarounds are "adaptions of work activities that are not expected or specified to be changed in this manner" (Laumer, Maier, and Weitzel, 2017, p.335). Workarounds play an important

role in almost all organizations (Alter, 2015; Röder et al., 2015b). For instance, nurses spend about twelve percent of their time working around operational failures and constraints (Tucker, Heisler, and Janisse, 2014).

Workarounds can disrupt the efficiency and quality of a business process, while they can also facilitate process innovation. The disruption might manifest as cumbersome (Alter, 2015) and inefficient processes that deviate from standard operating procedures (Röder et al., 2014). If workarounds remain undetected, they may become institutionalized in an organization and become difficult to change (Tucker, Heisler, and Janisse, 2014). At the same time, workarounds can also lead to performance improvements (Röder et al., 2014) and, thus, provide profound opportunities for analyzing and re-designing a business process (Alter, 2015; Beerepoot, Weerd, and Reijers, 2019).

Workarounds are complex phenomena, which prime business processes in a digitized world. Understanding them and their impact on business processes and technology enables organizations to create and maintain a competitive advantage. Previous research has centered around categorizing types of workarounds and on explaining how and why employees perform workarounds (Beerepoot and Weerd, 2018). Workarounds are often detected with qualitative empirical approaches, such as semi-structured interviews, direct observations, or document analysis. While these methods can foster rich empirical inquiries to identify and explain the emergence of workarounds, research often pre-supposes that informants are aware of their workarounds and are willing to expose the deviation from standard business processes (Beerepoot, Weerd, and Reijers, 2018). Also, manual detection methods appear as too labor-intensive for identifying workarounds on a large-scale.

In digitized business processes, workarounds leave data traces in event logs that can be identified and analyzed automatically. In business process management (BPM), a plethora of methods has been developed to detect anomalies in business processes (Nolle et al., 2018, 2019) but not for detecting workarounds. While first research indicates that process mining, especially compliance checking, can be used for detecting workarounds (Beerepoot and Weerd, 2018; Outmazgin and Soffer, 2013, 2016), these methods still have a limited predictive quality in detecting them (Outmazgin, 2012). Hence, we set out to design a new method for detecting workarounds automatically, based on analyzing real-life event log data. We apply deep neural networks to detect workarounds in high-dimensional data through multi-representation learning (LeCun, Bengio, and G. Hinton, 2015).

The contribution of this paper is twofold. First, following the design science research (DSR) paradigm by Gregor and Hevner (2013) and Hevner and March (2004), we design a deep-learning-based method that researchers and practitioners can use to detect and analyze workarounds performed in a digitized business process. Thereby, we provide a novel application on how data can be analyzed in organizations, driving the exploration and exploitation of innovation through workarounds. Second, by utilizing a BPM approach to identify performances of workarounds in organizations, we span boundaries between organizational routines and process mining.

The paper unfolds as follows. In Section 2, we present related research on workarounds and process mining. In Section 3, we describe and justify our research method. In Section 4, we present our method for identifying workarounds from event logs. In Section 5, we evaluate our method with three public data sets. In Section 6, we discuss implications for research before concluding the paper in Section 7.

2 Theoretical Background

2.1 Emergence of Workarounds

The concept of workarounds emerged in IS research in the mid-1980s, to describe the apparent mismatch between work requirements and design (Gasser, 1986). Workarounds are omnipresent in organizations and frequently occur when employees use IS to perform their day-to-day work (Röder et al., 2015b). Workarounds constitute "deviations from defined business processes that are carried out in the employees' performances of routines in a work system" (Wolf and Beverungen, 2019, p.1). Workarounds are performed

for various reasons, including troubleshooting, avoiding rules or limitations, mitigating poor IS' design, or simply to cut time (Röder et al., 2016).
Previous research considers workarounds as temporary deviations that are enacted by individuals (Azad and King, 2012). However, workarounds sometimes become institutionalized in an employee's daily routines, even if they conflict with the rules inscribed in IS (Azad and King, 2012). As they become institutionalized, workarounds can affect routines performed by other employees in an organization, too (Röder et al., 2016). If organizations manage to identify and resolve root causes behind workarounds, they can understand, improve, adapt, or redesign their business processes as continual improvement (Röder et al., 2015b).
Workarounds are directed towards changing IS or business processes (Wolf and Beverungen, 2019; Zainuddin and Staples, 2016) (see Figure 1). On the one hand, workarounds can constitute an anomalous use of IS (Azad and King, 2012; Ignatiadis and Nandhakumar, 2009). These workarounds are invoked through human actions that center around IT artifacts, i.e., using an IS in a way that is inconsistent with the artifact's original purpose (Azad and King, 2012). With the workaround, employees try to adapt an IS to their needs, to compensate for a system's perceived deficiencies, or to offset their ignorance of a system's features (Boudreau and Robey, 2005). Even though deviance in using an IS may conflict with the system's intended design, it is a viable form of enacting it (Azad and King, 2012). Workarounds directed towards an IS may be expressed by a violation of one's responsibilities if they contradict pre-defined roles (Alter, 2014; Ignatiadis and Nandhakumar, 2009; Outmazgin, 2012; Zainuddin and Staples, 2016) or manifest in the generation of manipulated data (Alter, 2015; Ignatiadis and Nandhakumar, 2009; Outmazgin and Soffer, 2013; Zainuddin and Staples, 2016).
On the other hand, employees may work around improperly designed business processes to complete tasks they deem essential for their day-to-day work (Outmazgin, 2012). In this regard, workarounds manifest in handling exceptions, addressing gray spaces, or deviating from routines, processes, or methods (Alter, 2014). Employees may repeat activities (Alter, 2014), substitute activities (Alter, 2015; Laumer, Maier, and Weitzel, 2017), interchange steps in a sequence of activities (Zainuddin and Staples, 2016), bypass activities (Alter, 2014; Outmazgin and Soffer, 2013, 2016; Röder et al., 2014), or add activities to a business process (Alter, 2015; Koopman and Hoffman, 2003; Laumer, Maier, and Weitzel, 2017).
If they remain undetected, workarounds can entail negative impacts, such as a loss of control (Lapointe and Rivard, 2005), compliance issues (Da Cunha and Carugati, 2009), or inferior process quality (Boudreau and Robey, 2005). However, recent research emphasizes that workarounds can also be a source of innovation as they might remedy misfits and inefficiencies between IS and business processes (Wolf and Beverungen, 2019). For example, workarounds have been argued to be vital in software engineering (Safadi and Faraj, 2010). Workarounds may diffuse through an organization if they affect the work of other employees (Leonardi, 2011; Pentland, Recker, and Wyner, 2017; Wolf and Beverungen, 2019), which amplifies their potential to change organizations for the better or worse. Either way, it is important to discover workarounds early to take action as needed (Outmazgin and Soffer, 2013), taking counter-measures to prevent undesirable consequences or guiding the emergence of workarounds as desirable process innovation. Thus, detecting workarounds can leverage process redesign, system improvement, training of employees, or taking disciplinary actions (Outmazgin and Soffer, 2013).

2.2 Process Mining for Detecting Workarounds

Process mining is a prevalent process analysis method in BPM that deals with the data-driven discovery, monitoring, and enhancement of processes (van der Aalst et al., 2011). While the detection of non-compliant or anomalous behavior in event log data has gained a lot of attention, only Outmazgin and Soffer (2016) have addressed the automated detection of workarounds as intentional non-compliant behavior in event log data. However, workarounds can also emerge as unintentional actions (Beerepoot and Weerd, 2018). Outmazgin and Soffer (2016) use a backward compliance checking method represented by six static rules for detecting six different workaround types. Nevertheless, their method has a deficit in

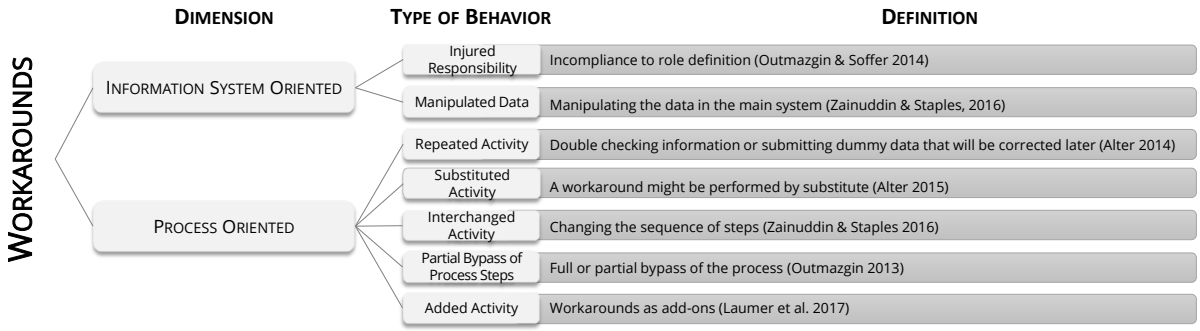

Figure 1. Taxonomy of Workarounds (adapted from Zainuddin and Staples (2016)).

detecting workarounds in (sequential) event log data, since the rules are static and workarounds belonging to a specific type can vary considerably (Bishop, 2006). For example, an employee may bypass not only a single step but up to ten or even more process steps when performing a business process, which highlights the need to detect workarounds that vary in parts of their properties.

Approaches to outcome-oriented predictive business process monitoring—a recent sub-field of process mining—typically build on machine learning algorithms to predict a process's categorical outcomes (Di Francescomarino et al., 2018). Categorical outcomes could be types of process states (e.g., the fulfillment of a compliance rule) or events in the process's environment (e.g., achievement of a business goal) at different time steps of a running (incomplete) process instance or based on a completed process instance (Teinemaa et al., 2019). Concerning anomaly detection, the categorical outcomes represent types of anomalous behavior. Current research has proposed using deep neural networks such as autoencoders (Nolle et al., 2018) and recurrent neural networks (Nolle et al., 2019) for detecting if a finished process instance represents an anomaly or not. In addition to binary classification, Nolle et al. (2019) use static rules to classify each process instance by a concrete anomaly type.

For two reasons, existing outcome-oriented, machine-learning-based predictive process monitoring techniques are insufficient for our purpose. First, these techniques do only refer to the subject of anomalies. An anomaly or abnormality always has a negative effect (Pimentel et al., 2014). However, a workaround is a broader and more complex phenomenon that can have positive or negative effects on the subsequent process; thereby, workarounds differ conceptually from anomalies. In other words, anomalies and workarounds can be considered as a group of behavior, respectively, where the intersection represents a subset of behavior types of both groups. Second, existing methods are rather geared to analyze synthetic data, especially for classifying process instances, distinguishing between different anomaly types (Nolle et al., 2019), and therefore, neglect noisy data, which naturally occur in real-life event log data.

3 Research Method

Our research follows the DSR paradigm for IS as proposed by Hevner and March (2004) and Gregor and Hevner (2013). In line with DSR, we address and solve a real-world problem by designing a socio-technical artifact and generate new knowledge (Gregor and Hevner, 2013). The DSR paradigm provides guidance for the design of an IT artifact, which are structured as constructs, models, methods, and/or instantiations (March and Smith, 1995).

We follow the seven design guidelines for DSR by Hevner and March (2004) for developing an IT artifact. Thereby, we also adhere to the schema proposed by Gregor and Hevner (2013). As DSR is inherently a problem-solving process, we review the descriptive and prescriptive knowledge in this area (Hevner and March, 2004). As no research has suggested a method for detecting workarounds automatically so far, our goal is to design a method for detecting workarounds in digitized business processes. We position our

contribution as exaptation as proposed by Gregor and Hevner (2013), by adopting a deep-learning-based method from BPM to a new problem context, i.e., for identifying workarounds in event logs. After defining the problem, we identify seven types of workarounds from the literature. Further, we examine existing approaches in process mining to detect workarounds. Subsequently, we begin with the design cycle in which we develop our method for workaround detection based on deep learning in a way that allows for detecting all seven classes of workarounds in event log data. By rooting our work in the existing theoretical foundations and methodologies, we ensure rigor and relevance (Hevner and March, 2004). Then, the artifact is instantiated, evaluated, and demonstrated. In the evaluation phase, we assess its ability to solve the previously identified problem of automated workaround detection (Hevner and March, 2004). We evaluate our method with applying it to three public data sets supplied for the business process intelligence (BPI) challenges 2012 (van Dongen, 2012), 2013 (Steeman, 2013), and 2019 (van Dongen, 2019) to assess our method's efficacy (Peffers et al., 2012). In the last step, by communicating our results, we offer prescriptive knowledge on how methods for identifying workarounds from even logs ought to be designed.

4 Deep Learning for Detecting Workarounds in Event Logs

We design a deep-learning-based method for detecting workarounds. The method detects automatically for each process instance of a real-life event log if a workaround of a particular type is included or not. The method comprises five phases (see Figure 2).

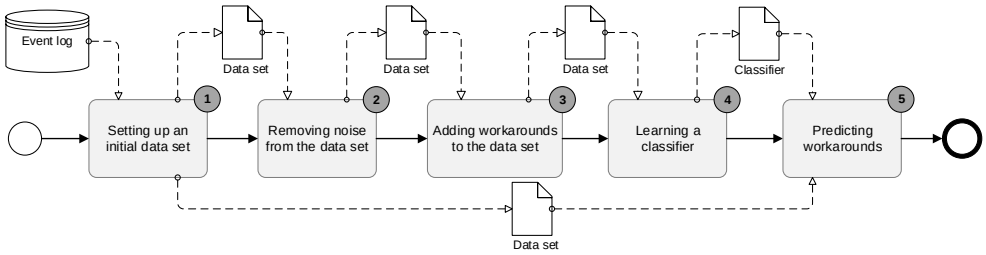

Figure 2. Phases of our Deep-Learning-Based Method for Detecting Workarounds.

First, an initial vector-oriented data set is created, which is based on an event log in order to apply deep neural networks. In the second phase, existing noisy data (like a workaround type and simply typing errors caused by inattentive process participants) are removed from the data set by using an autoencoder, which is a specific type of deep neural network. In the third phase, workaround variations from a defined set of workaround types are randomly added to the existing process instances that are contained in the data set. In the next phase, a classifier is learned by using a convolutional neural network (CNN). The classifier shall map a process instance to a workaround class. Finally, the learned classifier is applied on a left-out subset of the initial data set, in which the noisy data has not been removed. In the following sub-sections, each phase of the proposed method is presented, explained, and justified.

4.1 Setting Up an Initial Data Set

The first phase of the method transforms the event log's data representation and creates an initial data set that is geared towards its later use by deep neural networks. Thereby, we follow a well-established creation process consisting of the three steps (e.g., used in the work of Weinzierl, Stierle, et al. (2020)): load the event log, transform the event log to a data set, and encode the attributes of the data set.
In the first step, our method loads an event log L that we define as multi-set of process instance $L = \{pi_1, pi_2, \ldots, pi_l\}$, where $l = |L|$. A process instance represents a concrete execution of a business process and can be depicted as a sequence of events. Each event is described by an event type (activity) and

one or more additional context attributes. Context attributes characterize the environment of a process instance (Cunha Mattos et al., 2014). A process instance pi_i of an event log L can thus be formalized as:

$$pi_i = \langle \langle act_{0,0}^{(i)}, attr_{0,1}^{(i)}, \ldots, attr_{0,m_0}^{(i)} \rangle, \langle act_{1,0}^{(i)}, attr_{1,1}^{(i)}, \ldots, attr_{1,m_1}^{(i)} \rangle, \ldots, \langle act_{n,0}^{(i)}, attr_{n,1}^{(i)}, \ldots, attr_{n,m_n}^{(i)} \rangle \rangle, \quad (1)$$

where $act_j^{(i)}$ for $1 \leq j \leq n$ represents the type of the j^{th} event of process instance pi_i and $attr_{j,m_j}^{(i)}$ the m^{th} attribute associated to an event j. The number of attributes associated with an event and the number of events per process instance can vary. Besides, we assume that events have an explicit reference to a process instance; this is an assumption that is often made in process mining, and that has been reported to be typically satisfied in real-world event logs (van der Aalst, 2016). We use the event log $L_1 = \{pi_1, pi_2, pi_3\}$, depicted in (2), subsequently as an example to demonstrate our artifact. The events within each of the three process instances are ordered by their associated timestamp.

$$\begin{aligned} L_1 = \{ &\langle \langle (\text{"Create PO"}), (\text{"Hu"}), (\text{"PO"}) \rangle, \langle (\text{"Create Inv."}), (\text{"An"}), (\text{"Inv."}) \rangle, \langle (\text{"Pay"}), (\text{"Al"}), (\text{"PD"}) \rangle \rangle, \\ &\langle \langle (\text{"Create PO"}), (\text{"Al"}), (\text{"Order"}) \rangle, \langle (\text{"Pay"}), (\text{"Al"}), (\text{"PD"}) \rangle \rangle, \\ &\langle \langle (\text{"Create PO"}), (\text{"Hu"}), (\text{"PO"}) \rangle, \langle (\text{"Create Inv."}), (\text{"Yi"}), (\text{"Inv."}) \rangle, \langle (\text{"Pay"}), (\text{"Al"}), (\text{"PD"}) \rangle \rangle \}. \end{aligned} \quad (2)$$

In the second step, the event log L_1 with a log-oriented representation is transformed into a data set D_1 with a vector-oriented representation (Weinzierl, Revoredo, and Matzner, 2019). In most cases, event logs are encoded using a log-oriented representation, like the popular XES format does (Verbeek et al., 2010). However, a deep neural network requires a data set with a fixed-sized structure as input (LeCun, Bengio, and G. Hinton, 2015) like the two-dimensional data structure shown in Table 1, in which each row has the same number of columns.

act_0	$attr_{0,0}$	$attr_{0,1}$	act_1	$attr_{1,0}$	$attr_{1,1}$	act_2	$attr_{2,0}$	$attr_{2,1}$
"Create PO"	"Hu"	"PO"	"Create Inv."	"An"	"Inv."	"Pay"	"Al"	"PD"
"Create PO"	"Al"	"Order"	"Pay"	"Al"	"PD"			
"Create PO"	"Hu"	"PO"	"Create Inv."	"Yi"	"Inv."	"Pay"	"Al	"PD"

Table 1. Data Set D_1 with a Fixed-Size, Vector-Oriented Representation.

In the last step of this phase, discrete values of activities and context attributes of the data set are transformed into numeric ones. A deep neural network that is fully differential requires numerical data in order to perform a backpropagation algorithm (e.g., stochastic gradient descent). In general, a deep neural network's architecture consists of one input layer, several hidden layers, and one output layer. A backpropagation algorithm is typically the way in which the multi-layered architecture of a deep neural network is trained (LeCun, Bengio, and G. Hinton, 2015).

First, we address missing values. On the one hand, missing values of the activity attribute and each categorically scaled context attribute is replaced by the global constant "0" (Han, Pei, and Kamber, 2011). This global constant does not affect the deep neural network's result since this numerical value differs from the existing values of a categorical attribute. On the other hand, we replace missing values of each numerically scaled context attribute by the median of the existing values (Han, Pei, and Kamber, 2011). Again, the median does not affect the deep neural network's result because it indicates the "middle" of data distribution. However, this step is necessary since a deep neural network cannot deal with missing values. Further, for each numerically scaled context attribute with a value range that is not equal to $[0, 1]$, a min-max normalization is applied. Finally, we apply binary encoding for the activity attribute as well as for each of the categorically scaled context attributes. Table 2 depicts the resulting data set D_1. Each categorial context attribute now has two different values[1].

[1] Note, from a technical perspective, D_1 has to be reshaped for the numerical transformation so that the values of the same attributes (e.g., act_0, act_1 and act_2) are in the same column (e.g., act). Additionally, each column (e.g., act_0) in Table 2 represents two separate columns in a second-order tensor.

act_0		$attr_{0,0}$		$attr_{0,1}$		act_1		$attr_{1,0}$		$attr_{1,1}$		act_2		$attr_{2,0}$		$attr_{2,1}$	
0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1
0	1	1	0	0	1	1	1	1	0	1	1	0	0	0	0	0	0
0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1

Table 2. Initial Data Set D_1.

Finally, the initial data set D_1 is divided into two subsets. One data set is transferred to the subsequent step of the method while the other data set is left out for the last step of the method in which we predict workarounds.

4.2 Removing Noise from the Data Set

We assume that not only workarounds but also other forms of deviance (like fraud, sabotage, or interpretive flexibility) of (un-)intentional deviating behavior exist. All are a source of what data analysts call "noise" or "noisy data" (Han, Pei, and Kamber, 2011). Thus, our method suggests first to remove the entire noisy data (i.e., workarounds and other deviations) using an autoencoder, which is another type of neural network. In a subsequent step, we create and reintegrate noisy data that can be explained by workarounds. In general, a complete autoencoder is a neural network that is trained to attempt copying its input to its output (Goodfellow, Bengio, and Courville, 2016). It comprises two components. The first component is the encoder function f that maps the input x to an internal representation h. The second component is the decoder function g that maps the internal representation h to the reconstruction r. Given the data set D_1 consisting of process instances, the reconstruction error E as an element of the loss function $L(D_1, g(f(D_1))$ is minimized during the learning process (Goodfellow, Bengio, and Courville, 2016). After the learning process is finished, the function o, as depicted in (3), is applied to each process instance $pi_i \in D_1$ to determine the data set D_{1,no_noise} without noise.

$$o(pi_i) = \begin{cases} pi_i \in D_{1,no\_noise}, & \text{if } e(pi_i) \leq t. \\ pi_i \notin D_{1,no\_noise}, & \text{otherwise.} \end{cases} \tag{3}$$

In (3), represents $e(pi_i)$ the reconstruction error of p_i, while t constitutes the threshold determined by $median(E) + sd(E)$. After applying $o(pi_i)$, the resulting data set D_{1,no_noise} is depicted in Table 3, where the second process instance of D_1 is removed since the resource "Al" bypassed the event "Create Inv.".

act_0		$attr_{0,0}$		$attr_{0,1}$		act_1		$attr_{1,0}$		$attr_{1,1}$		act_2		$attr_{2,0}$		$attr_{2,1}$	
0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1
0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1

Table 3. Data Set D_{1,no_noise} without Noise.

4.3 Adding Workarounds to the Data Set

In this phase, the noisy data related to workarounds are integrated into D_{1,no_noise}. We follow the examples of Nolle et al. (2019), Böhmer and Rinderle-Ma (2016), and Bezerra and Wainer (2013) who detect anomalies in event logs, and add noisy data (representing anomalies) to 30% of the process instances randomly selected from the data set D_{1,no_noise}. Thus, we leave 70% of the process instances unchanged and assign them to a class *No workaround (0)* that does not represent a workaround type. We integrate

one of the seven workaround types into each of the other 30% of the process instances and assign these to a class that corresponds to the workaround type. Based on the workaround taxonomy presented in Figure 1 and inspired by the work of Nolle et al. (2019), we define the seven workaround types, as shown in Table 4.

Id	Type	Description
1	Violated responsibility	Randomly select any other value of a resource attribute.
2	Manipulated data	Randomly select any other value of a data attribute.
3	Repeated activity	Repeat a randomly selected activity (and its context attributes) of *pi* three up to ten times.
4	Substituted activity	Replace a randomly selected activity of *pi* by an activity that is not yet included. Given this activity, the values of its context attributes are retrieved from another randomly selected process instance, where this activity is part of.
5	Interchanged activity	Change a randomly selected activity (and its context attributes) of *pi* (except the last one) pairwisely with its subsequent activity (and its context attributes).
6	Bypassed activity	Skip one or more activities of *pi*, where the start and the bypassed steps are randomly selected depending on each other.
7	Added activity	Add an activity that is not part of *pi*. Given this randomly selected activity, values of its context attributes are retrieved from another randomly selected process instance, where this activity is part of.

Table 4. Technical Specification of the Seven Workaround Types.

As shown in Table 5, a workaround of the type *bypassed activity* is integrated into the second process instance of the data set D_{1,no_noise}. Additionally, the last column represents the label specifying the eight different workaround classes.

act_0		$attr_{0,0}$		$attr_{0,1}$		act_1		$attr_{1,0}$		$attr_{1,1}$		act_2		$attr_{2,0}$		$attr_{2,1}$		Workaround
0	1	0	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6

Table 5. Data Set D_{1,wa_noise} after Adding Noisy Data and the Label Specifying the Workaround Class.

4.4 Learning a Classifier

To learn a classifier represented by a function $f(D_{1,wa\_noise})$ that maps each process instance of D_{1,wa_noise} to one workaround type, we use a CNN (LeCun, 1989). CNNs are neural networks that include at least one convolutional layer, consisting of three stages: convolutional stage, detection stage, and pooling stage (Goodfellow, Bengio, and Courville, 2016).

In the first stage, several convolutions are performed in parallel to produce a set of linear activations based on the input, e.g., a sequence. A convolution is a filtering operation performed by a feature map. All units in a feature map share the same filter bank, in which the weights for this filter are stored (this principle is called parameter sharing). For each feature map, only one linear activation is calculated by element-wise multiplication between the filter-sized batch of the input and the filter bank, which is then summed. In the second stage, each linear activation is run through a nonlinear activation function. In the third stage, a pooling function is used to merge semantically similar features (LeCun, Bengio, and G. Hinton, 2015) and to remove irrelevant features.

We decided to use a CNN for two reasons. First, a CNN can process one-dimensional, sequential input data (LeCun, Bengio, and G. Hinton, 2015). Second, a CNN typically provides the capability of sparse interactions accomplished by making the kernel (called filter) smaller than the input (Goodfellow, Bengio, and Courville, 2016). For example, if we consider a process instance of D_{1,wa_noise} featuring low-level encoded activities and context attributes, the process instance might have one thousand or more elements.

However, we can detect small, meaningful features such as workaround types that occupy only ten or one hundred of these elements.

4.5 Predicting Workarounds

In the last phase, the learned CNN model is applied to the subset $D_{1,pred}$ of the initial data set D_1, created by the first phase. Concerning $D_{1,pred}$, noisy data that are presented in the data set remain unchanged. For each process instance pi_i of $D_{1,pred}$ the CNN model calculates a probability distribution d_i, where $1 \leq i \leq |D_{1,pred}|$. The most likely workaround type for pi_i is determined by $max(d_i)$.

5 Evaluation

5.1 Event Logs

We evaluate our method based on three event logs that store human behavior, i.e., only activities performed by humans. This selection criterion is essential because workarounds represent intended or unintended human behavior (Beerepoot and Weerd, 2018). The main characteristics of the three event logs are summarized in Table 6.

Event log	# process instances	# process instance variants	# events	# activities	# context attributes	Dimension of con. attributes	Instance length [min; max; mean]
bpi2019	8,790	1,944	74,745	38	3	[275; 3; 2]	[1; 278; 8.50]
bpi2013i	7,554	2,278	65,533	13	3	[1,440; 25; 24]	[2; 123; 8.68]
bpi2012w	9,658	2,263	72,413	6	2	[60; 530]	[1; 74; 7.50]

Table 6. Characteristics of the Used Event Logs.

The *bpi2019* (van Dongen, 2019) event log shared at the BPI 2019 comes from a large multinational coating and paints company operating in the Netherlands. The data represent a mass transaction process, i.e., a purchasing process. As the event log had an original size of 1,595,923 events, we use a sample of the first 80,000 events that is comparable in size with the *bpi2012w* and *bpi2012w* event logs. For computational reasons, we remove process instances that include more than 300 events (the longest sequence observed in the original event log is 924), as including longer sequences would have challenged the available resources and computing time. Finally, our experiment considers the context attributes *event org:resource* (human actor), *case Document Type*, and *case GR-Based Inv. Verif.*
The *bpi2013i* (Steeman, 2013) event log provided at the BPI 2013 comes from the Volvo IT department in Belgium and contains data that relate to their incident and problem management process. We take the data of all process instances of this event log. We select the context attributes *Owner First Name* (human actor), *Involved ST Function Div*, and *Involved Org line 3*.
The *bpi2012w* (van Dongen, 2012) event log was published for the BPI challenge 2012, and it contains data from an application process for commercial products of a large financial institution. This process consists of three sub-processes. We just consider the *work item* sub-process, because it is the only sub-process that includes activities performed by humans rather than machines. We filter out all events that are not of the type *complete*. We select the context attributes *org:resource_Event* (human actor) and *AMOUNT_REQ_Trace*.

5.2 Evaluation Procedure

For each of the three real-life event logs, a 10%-subset of the process instances (prediction set) is randomly selected for the last phase of our method. From the other 90% of the process instances, the noisy data are removed. Subsequently, as recommended by Haykin (1998) for neural networks, the remaining process instances are randomly split in an 80%-training-set and a 20%-testing-set.

In phase two of the method, the noisy data are removed from the 90%-subset of the original event log by using an autoencoder. Inspired by the work of G. E. Hinton and Salakhutdinov (2006), the architecture of the applied autoencoder consists of one input layer, five hidden layers, and one output layer. The first three layers of the hidden layers represent the encoder component and the following two layers the decoder component. All hidden layers are dense layers. We assigned the following number of neurons: 128, 64, 32, 64, and 128. Each hidden layer applied a *tanh* activation function. Finally, the number of neurons of the output layer corresponds to the number of neurons of the input layer. In the training phase of the autoencoder, a batch size of 256 is applied, where gradients are updated after each 256th process instance. Further, we perform the *Adam* optimization algorithm (Kingma and Ba, 2014) in each autoencoder model with a *mean squared error loss*. For other parameters of this optimization algorithm (e.g., the learning rate), we use default values.
As described in phase four of the method, a CNN is used to learn the mapping from the process instances of the training set to the created workaround labels. The CNN's architecture consists of eight layers. The first five layers are convolutional layers comprising a convolution stage, a detector stage, and a pooling stage (Goodfellow, Bengio, and Courville, 2016). In line with Al-Jebrni, Cai, and Jiang (2018), we use a deeper architecture with five convolutional layers to reach a higher rate in detecting the different workaround types. For the convolution stage, 128 filters and a kernel size of two, are set. For the detector stage, the activation function *rectifier linear unit (ReLU)* is used. In the pooling stage, the pooling function *max-pooling* is applied. A flatten layer is used to transform the three-dimensional output of the last convolutional layer to a two-dimensional representation for the subsequent dense layer. The dense layer includes 100 neurons, and the activation function *ReLU* is set. The number of neurons of the last layer corresponds to the number of workaround classes. For this layer, the *Sigmoid* activation function is applied. Further, as with the autoencoder, the batch size is set to 256. We use the *Nadam* optimization algorithm (Dozat, 2016) in each CNN model with a *categorical cross-entropy loss*. Again, for other parameters of the optimization algorithm, we use the default values.
After learning the CNN models in phase four, we test their predictive quality based on the test set. Thereby, we use, on the one hand, the four states of the confusion matrix for presenting and discussing the predictive quality of the three classification models, in which we differentiate between the two classes: *workaround* and *no workaround*. The four states of the confusion matrix are: true positives (*TPs*), true negative (*TNs*), false positives (*FPs*), and false negatives (*FNs*). On the other hand, we calculate the average *accuracy*, *precision*, and *recall* of all eight workaround classes (seven workaround types and the class *no workaround)*. These machine-learning metrics are well-established in the process-mining community, and explanations can be found in the IS literature as well, e.g., in Breuker, Matzner, et al. (2016). In phase five, the trained and tested CNN models are applied based on the prediction set that includes noisy data.

5.3 Results

Due to the common existence of noisy data, we remove from the 90%-subsets: 300 of 7,828 process instances from bpi2019; 918 of 6,732 process instances from bpi2013i; 1,527 of 6,715 process instances from bpi2012w. Next, we train a classifier based on each training set, and these classifiers are subsequently tested using the corresponding test sets. Table 7 presents quality metrics for each classifier.

Event log	# test instances	# TPs	# TNs	# FPs	# FNs	Accuracy	Precision	Recall	F1-score
bpi2019	1,504	365	1,004	58	77	0.865	0.739	0.700	0.716
bpi2013i	1,163	135	808	25	195	0.759	0.359	0.310	0.321
bpi2012w	1,038	89	714	20	215	0.745	0.403	0.286	0.310

Table 7. Result of Detecting Workarounds in three Data Sets.

The method detected 365 process instances correctly as workarounds in the case of bpi2019; it detected fewer workarounds for bpi2013i (135) and bpi2012w (89). The number of false classifications was also the

lowest in the case of bpi2019. Therefore, the highest *accuracy*, *precision*, *recall*, and *f1-score* values were observed for the bpi2019 event log. To convey a deeper understanding of the detection quality, Figure 3 shows the test quality for detecting each workaround class based on the bpi2019 event log.

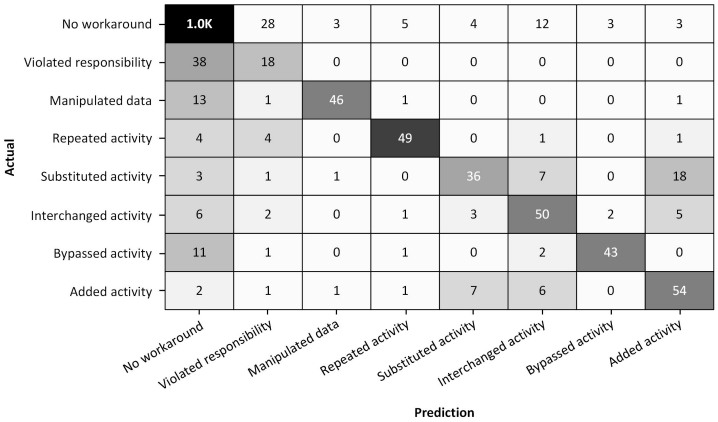

Figure 3. Results for Detecting each Workaround Class Based on the bpi2019 Event Log.

Figure 3 illustrates that we can detect process instances of each workaround type in the bpi2019 event log using our CNN-based approach. Only the type *violated responsibility* is more often detected as the type *no workaround*. However, the CNN wrongly detects 28 processes instances of the class *no workaround* as to belong to the class *violated responsibility*. For the workaround type *substituted activity*, 18 process instances are wrongly detected to be a workaround of the type *added activity*. The classifiers were not only applied to the testing set but also to the prediction set. Table 8 summarizes our prediction results. For the bpi2019 event log, each workaround type could be detected at least once. Workarounds of the type *violated responsibility* are detected correctly in most cases. Regarding the bpi2013i event log, three of the types could not be detected, and the type *repeated activity* shows the highest number of occurrences. Finally, for the bpi2012w event log, four of the types are not detected, and the workaround type *bypassed activity* occurs in 225 process instances.

Event log	No workaround	Violated responsibility	Manipulated data	Repeated activity	Substituted activity	Interchanged activity	Bypassed activity	Added activity	Sum
bpi2019	788	40	2	16	1	9	19	4	879
bpi2013i	670	0	0	47	7	0	12	20	756
bpi2012w	618	0	0	91	0	0	225	32	966

Table 8. Prediction Results for Detecting Workarounds.

6 Discussion

With the designed method, we specifically address the gap identified by Breuker and Matzner (2013), emphasizing that event log data constitute a valuable data source for studying organizational routines. In this regard, we investigate workarounds as deviations from routines and business processes, constituting a promising source for innovation. Most research on workarounds is centered around how and why employees perform them (Beerepoot and Weerd, 2018). However, researchers have already postulated that an automatic detection method for workarounds would be beneficial since it remedies biases and hidden information in empirical research methods (Beerepoot, Weerd, and Reijers, 2018; Röder et al., 2015a). With the successful implementation of our method, we enable a new and highly requested approach for the detection and analysis of workarounds in event logs.

As identified in the evaluation, our method is capable of detecting most workarounds in the event logs we analyzed. Nevertheless, the results also show that *accuracy*, *precision*, *recall*, and *f1-score* differ substantially across all our three event logs. In our evaluation, the highest predictive quality is achieved for the bpi2019 event log. 365 process instances were correctly detected as workarounds. The number of false classifications was also the lowest across all three event logs. As we made a distinction between eight workaround classes, we achieve an *accuracy* of 86.45%, a *precision* of 73.90%, a *recall* of 70.01%, and a *f1-score* of 71.60%. All four metrics highlight that the method performs well in identifying workarounds in this event log.

However, only 135 process instances in the bpi2013i event log and 89 process instances in the bpi2012w event log were detected as workarounds correctly. The number of false classifications in both event logs is much higher compared to the bpi2019 event log. With respect to detecting different workaround types, the highest difference to the bpi2019 event log is exhibited in terms of *precision*, *recall*, and *f1-score*. For bpi2019, *precision* is more than 30% higher than for bpi2013i and bpi2012w. Likewise, bpi2019's *recall* value exceeds the other two logs' value by about 40%. The bpi2019's *f1-score* is also more than 30% higher than for the other two logs since it is the harmonic mean of *precision* and *recall*.

Further, the test results for the bpi2019 event log show that our method sometimes misinterpreted *no workarounds* as *violated responsibilities*. An explanation could be that our sample size was not big enough to sufficiently train the model, as the context attribute *event org:resource* has 275 different values. Moreover, the results show that workarounds of the type *substituted activity* are often mixed-up with workarounds of the type *added activity*. This observation can be explained by the similarity of both workaround types, which makes it hard for an automated method to draw the distinction correctly.

Interestingly, two workaround types appeared to have lower correct detection rates than the others. First, *violated responsibility* might be very difficult to detect. Inferring from real-life scenarios of daily work, it seems reasonable to assume that *violated responsibility* may not always be documented in event logs. For instance, some employees might provide co-workers access to a database, PC workstation, or other resources (e.g., by sharing their password), which the system could not detect. In other processes, activities might be assigned to user roles, and user roles are mapped to a group of employees, making it hard to detect violations without permission to an access control list. Second, the detection rate of *substituted activity* is lower than the average detection rate of other workarounds. We conclude that this workaround type is not likely to occur in processes in which the control flow is defined strictly.

Our results show that workarounds can be detected very well in mass-transaction data that display fewer variations in a process's control flow, such as the purchase-to-pay process that is documented in bpi2019 (cf, Table 6). This observation can be attributed to the fact that employees often have relatively strict instructions and restrictive software tools in this type of process, which leave fewer possibilities to deviate. If processes display greater variability and employees are expected to instantiate processes differently, the comparability of performances and, therefore, the predictive quality of detecting workarounds is reduced. Both event logs, bpi2013i and bpi2012w (cf, Table 6), which we used to evaluate our method, document service processes that, by definition, are conducted cooperatively with customers. Therefore, it seems natural that training deep neural networks requires much more data to enable an algorithm for distinguishing a workaround from other deviations in a process. Further, data richness in an event log can, of course, impact the predictive quality for detecting workarounds. As shown in Table 6, the number of activities exhibited in the bpi2013i and bpi2012w event logs is lower than in bpi2019. Hence, we assume that employees have fewer options to perform workarounds in these processes. Additionally, we suppose that the imbalance regarding the number of instances belonging to the different workaround classes decreases the predictive quality of our method. Especially since the learning process of a CNN, as used in our method, can be negatively affected by a class imbalance (Buda, Maki, and Mazurowski, 2018).

Concerning the identification of particular workaround types, our evaluation shows that some types were detected more accurately than others. For example, *substituted activities* can only be detected if the workaround is executed by using the same IS, i.e., can be found in the same event log data. If an employee substitutes an activity through manual activities or by using a different IS, the workaround

cannot be detected in the data set. If these activities are missing in the event logs, our method would detect the workaround *bypassed activity*. Hence, our method works best for processes that are inscribed in a workflow engine or if event logs are merged before the analysis is performed.

7 Conclusion

Drawing on process mining and machine learning, we designed a deep-learning-based method to detect workarounds in event logs automatically. An evaluation with three public event logs shows that our method is capable of detecting workarounds with high predictive *accuracy*, *precision*, *recall* and, *f1-score* for standardized processes that display few variations and contain a sufficient amount of instances to train the deep neural network. Workarounds are ubiquitous in an organization and frequently occur while using IS. Often, workarounds remain undetected, because employees do not report deviations from a business process or regarding the use of an IS. This can lead to a gradual manifestation in the organizational structures (Tucker, Heisler, and Janisse, 2014), which can have positive or negative consequences. Besides the unawareness of their employee's actions, managers are often missing easy applicable methods and tools to analyze and improve their business. By providing a method for automatically detecting workarounds, we make a significant contribution to research and practice who can easily identify workarounds and derive actions to either counteract or leverage the impact of workarounds.

While our results are encouraging, our study is also subject to limitations. First, our evaluation suggests that some types of workaround might be harder to detect than others with process mining methods (Outmazgin and Soffer, 2016). For instance, if employees use a mix of different IS to perform a task, workarounds might no longer be detected based on event logs from just one system. Second, we cannot be sure if workarounds exist in the removed noisy data of the event logs, if workarounds exist in the remaining noise-free portion of the event log data, or if the noisy data included regular variations of a process that are not workarounds. Third, the implemented specification of the workaround types is subject to design decisions we made when training the neural network; these decisions might reflect the nature of particular workarounds somewhat distorted (cf, Figure 1), as workarounds might manifest differently depending on the specific behavior in an event log. As suggested by Nolle et al. (2019), we reduce this problem by using randomized elements in the specification of the different workaround types. Finally, we focused exclusively on workarounds that can be detected in an event log, whereas manually performed workarounds were not considered.

In summary, our study strengthens the connection between research in organizational routines and process mining, which are astonishingly separated from each other, even if they investigate similar phenomena (Breuker and Matzner, 2014). Additional research is required to elaborate more on the nexus between both disciplines. We plan to investigate further event logs to better understand from which event log characteristics our method's predictive quality depends on. In addition, researchers can specify different workaround types more precisely, investigate the relationship between the autoencoder and the CNN, consider "traditional" machine-learning algorithms (e.g., a "shallow" neural network) or other deep-learning algorithms (e.g., a long short-term memory neural network (Hochreiter and Schmidhuber, 1997)) for detecting workarounds, perform a log-wise hyperparameter optimization or address the class-imbalance problem of CNNs (e.g., via an oversampling method (Buda, Maki, and Mazurowski, 2018)). Finally, researchers can detect multiple workarounds per process instance, even if the types of workarounds are different from each other. A CNN, as used in this paper, can be a promising source because of its shared parameter capability.

Acknowledgements

This project is funded by the German Federal Ministry of Education and Research (BMBF) within the framework program *Software Campus* (www.softwarecampus.de) under the funding code 01IS17045 and

the project *Digivation* (www.digivation.de) under the funding code 02K14A220.

References

Alter, S. (2014). "Theory of Workarounds." *Communications of the Association for Information Systems* 34 (55), 1041–1066.

— (2015). "A Workaround Design System for Anticipating, Designing, and/or Preventing Workarounds." In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, pp. 489–498.

Azad, B. and N. King (2012). "Institutionalized Computer Workaround Practices in a Mediterranean Country: An Examination of Two Organizations." *European Journal of Information Systems* 21 (4), 358–372.

Beerepoot, I. and I. van de Weerd (2018). "Prevent, Redesign, Adopt or Ignore: Improving Healthcare Using Knowledge of Workarounds." In: *Proccedings of the 26th European Conference on Information Systems (ECIS2018)*, pp. 1–15.

Beerepoot, I., I. van de Weerd, and H. A. Reijers (2018). "Detecting Workarounds Using Domain Knowledge-driven Process Mining." In: *Proceedings of the 39th International Conference on Information Systems (ICIS2018)*, pp. 1–1.

— (2019). "Business Process Improvement Activities: Differences in Organizational Size, Culture, and Resources." In: *Proceedings of the 17th International Conference on Business Process Management (BPM2019)*. Springer, pp. 402–418.

Bezerra, F. and J. Wainer (2013). "Algorithms for Anomaly Detection of Traces in Logs of Process Aware Information systems." *Information Systems* 38 (1), 33–44.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. New York, NY: Springer, pp. 1–738.

Böhmer, K. and S. Rinderle-Ma (2016). "Multi-perspective Anomaly Detection in Business Process Execution Events." In: *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, pp. 80–98.

Boudreau, M.-C. and D. Robey (2005). "Enacting Integrated Information Technology: A Human Agency Perspective." *Organization Science* 16 (1), 3–18.

Breuker, D. and M. Matzner (2013). "Statistical Sequence Analysis for Business Process Mining and Organizational routines." In: *Proceedings of the 21st European Conference on Information Systems (ECIS2013*, pp. 1–12.

— (2014). "Performances of Business Processes and Organizational Routines: Similar Research Problems, Different Research Methods-A Literature Review." In: *Proceedings of the 22nd European Conference on Information Systems (ECIS2014)*, pp. 1–13.

Breuker, D., M. Matzner, P. Delfmann, and J. Becker (2016). "Comprehensible Predictive Models for Business Processes." *MIS Quarterly* 40 (4), 1009–1034.

Buda, M., A. Maki, and M. A. Mazurowski (2018). "A systematic study of the class imbalance problem in convolutional neural networks." *Neural Networks* 106, 249–259.

Cunha Mattos, T. da, F. Santoro, K. Revoredo, and V. Nunes (2014). "A Formal Representation for Context-Aware Business Processes." *Computers in Industry* 65, 1193–1214.

Da Cunha, J. V. and A. Carugati (2009). "Information Technology and the First-Line Manager's Dilemma: Lessons from an Ethnographic Study." In: *Preceeding of the 17th European Conference on Information Systems (ECIS2009)*, pp. 2834–2845.

Di Francescomarino, C., C. Ghidini, F. M. Maggi, and F. Milani (2018). "Predictive Process Monitoring Methods: Which One Suits Me Best?" In: *Proceedings of 16th International Conference on Business Process Management (BPM2018)*. Springer, pp. 462–479.

Dozat, T. (2016). "Incorporating Nesterov Momentum into Adam." In: *Proceedings of the 4th International Conference on Learning Representations (ICLR2016)*, pp. 1–4.

Gasser, L. (1986). "The Integration of Computing and Routine Work." *ACM Transactions on Information Systems (TOIS)* 4 (3), 205–225.

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. Cambridge, MA: MIT Press, pp. 1–800.

Gregor, S. and A. Hevner (2013). "Positioning and Presenting Design Science Research for Maximum Impact." *MIS Quarterly*, 337–355.

Han, J., J. Pei, and M. Kamber (2011). *Data Mining: Concepts and Techniques*. 3rd Edition. Waltham, MA: Elsevier, pp. 1–740.

Haykin, S. S. (1998). *Neural Networks: A Comprehensive Foundation*. 2nd Edition. Upper Saddle River, NJ: Pearson Prentice Hall, pp. 1–823.

Hevner, A. and S. T. March (2004). "Design Science Research in Information Systems." *MIS Quarterly* 28 (1), 75–105.

Hinton, G. E. and R. R. Salakhutdinov (2006). "Reducing the Dimensionality of Data with Neural Networks." *Science* 313 (5786), 504–507.

Hochreiter, S. and J. Schmidhuber (1997). "Long short-term memory." *Neural Computation* 9 (8), 1735–1780.

Ignatiadis, I. and J. Nandhakumar (2009). "The Effect of ERP System Workarounds on Organizational Control: An Interpretivist Case Study." *Scandinavian Journal of Information Systems* 21 (2), 59–90.

Al-Jebrni, A., H. Cai, and L. Jiang (2018). "Predicting the Next Process Event Using Convolutional Neural Networks." In: *Proceedings of the International Conference on Progress in Informatics and Computing (PIC2018)*. IEEE, pp. 332–338.

Kingma, D. P. and J. Ba (2014). "Adam: A Method for Stochastic Optimization." In: *Proceedings of the 3rd International Conference on Learning Representations (ICLR2015)*, pp. 1–15.

Koopman, P. and R. R. Hoffman (2003). "Work-arounds, Make-work, and Kludges." *IEEE Intelligent Systems* 18 (6), 70–75.

Lapointe, L. and S. Rivard (2005). "A Multilevel Model of Resistance to Information Technology Implementation." *MIS Quarterly* 29 (3), 461–491.

Laumer, S., C. Maier, and T. Weitzel (2017). "Information Quality, User Satisfaction, and the Manifestation of Workarounds: A Qualitative and Quantitative Study of Enterprise Content Management System Users." *European Journal of Information Systems* 26 (4), 333–360.

LeCun, Y. (1989). "Generalization and Network Design Strategies." In: *Connectionism in Perspective*. Holmdel, NJ: Elsevier.

LeCun, Y., Y. Bengio, and G. Hinton (2015). "Deep Learning." *Nature* 521 (7553), 436–444.

Leonardi, P. M. (2011). "When Flexible Routines Meet Flexible Technologies: Affordance, Constraint, and the Imbrication of Human and Material Agencies." *MIS Quarterly* 35 (1), 147–167.

March, S. T. and G. F. Smith (1995). "Design and Natural Science Research on Information Technology." *Decision Support Systems* 15 (4), 251–266.

Nolle, T., S. Luettgen, A. Seeliger, and M. Mühlhäuser (2018). "Analyzing Business Process Anomalies Using Autoencoders." *Machine Learning* 107 (11), 1875–1893.

— (2019). "BINet: Multi-Perspective Business Process Anomaly Classification." *Information Systems*.

Outmazgin, N. (2012). "Exploring Workaround Situations in Business Processes." In: *Proceedings of the International Conference on Business Process Management (BPM2012)*. Springer, pp. 426–437.

Outmazgin, N. and P. Soffer (2013). "Business Process Workarounds: What Can and Cannot Be Detected by Process Mining." In: *Enterprise, Business-Process and Information Systems Modeling*. Springer, pp. 48–62.

— (2016). "A Process Mining-Based Analysis of Business Process Workarounds." *Software & Systems Modeling* 15 (2), 309–323.

Peffers, K., M. Rothenberger, T. Tuunanen, and R. Vaezi (2012). "Design Science Research Evaluation." In: *Proceedings of the International Conference on Design Science Research in Information Systems (DESRIST2012)*. Springer, pp. 398–410.

Pentland, B. T., J. Recker, and G. Wyner (2017). "Rediscovering Handoffs." *Academy of Management Discoveries* 3 (3), 284–301.

Pimentel, M. A., D. A. Clifton, L. Clifton, and L. Tarassenko (2014). "A Review of Novelty Detection." *Signal Processing* 99, 215–249.

Röder, N., M. Wiesche, M. Schermann, and H. Krcmar (2014). "Why Managers Tolerate Workarounds-The Role of Information Systems." In: *Proceedings of the 20th American Conference on Information Systems (AMCIS2014)*, pp. 1–13.

— (2015a). "Embracing a Relational View of Workarounds: An Affordance Perspective." In: *Proceedings of the Twentieth DIGIT Workshop*.

— (2015b). "Workaround Aware Business Process Modeling." In: *Proceedings of the 12th Conference on Wirtschaftsinformatik (WI2019)*, pp. 482–496.

— (2016). "Toward an Ontology of Workarounds: A Literature Review on Existing Concepts." In: *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS2016)*. IEEE, pp. 5177–5186.

Safadi, H. and S. Faraj (2010). "The Role of Workarounds During an Open Source Electronic Medical Record System Implementation." In: *Proceedings of the 31st International Conference on Information Systems (ICIS2010)*, pp. 1–10.

Steeman, W. (2013). *BPI Challenge 2013*. `https://data.4tu.nl/repository/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07`. [Online; accessed 2019-03-29].

Teinemaa, I., M. Dumas, M. L. Rosa, and F. M. Maggi (2019). "Outcome-Oriented Predictive Process Monitoring: Review and Benchmark." *ACM Transactions on Knowledge Discovery from Data (TKDD)* 13 (2), 1–57.

Tucker, A. L., W. S. Heisler, and L. D. Janisse (2014). "Designed for Workarounds: a Qualitative Study of the Causes of Operational Failures in Hospitals." *The Permanente Journal* 18 (3), 33.

van der Aalst, W. M. P. (2016). *Process Mining: Data Science in Action*. 2nd Edition. Berlin, Heidelberg, Germany: Springer, pp. 1–467.

van der Aalst, W. M. P., A. Adriansyah, A. K. A. De Medeiros, F. Arcieri, T. Baier, T. Blickle, J. C. Bose, P. Van Den Brand, R. Brandtjen, J. Buijs, et al. (2011). "Process Mining Manifesto." In: *Proceedings of the 9th International Conference on Business Process Management (BPM2011)*. Springer, pp. 169–194.

van Dongen, B. F. (2012). *BPI Challenge 2012*. `https://data.4tu.nl/repository/uuid:3926db30-f712-4394-aebc-75976070e91f`. [Online; accessed 2019-03-29].

— (2019). *BPI Challenge 2019*. `https://data.4tu.nl/repository/uuid:a7ce5c55-03a7-4583-b855-98b86e1a2b07`. [Online; accessed 2019-03-29].

Verbeek, H., J. Buijs, B. Van Dongen, and W. van der Aalst (2010). "XES, XESame, and ProM 6." In: *Proceedings of the 22nd International Conference on Advanced Information Systems Engineering (CAiSE2010)*. Springer, pp. 60–75.

Weinzierl, S., K. C. Revoredo, and M. Matzner (2019). "Predictive Business Process Monitoring With Context Information from Documents." In: *Proceedings of the 27th European Conference on Information Systems (ECIS2019)*, pp. 1–10.

Weinzierl, S., M. Stierle, S. Zilker, and M. Matzner (2020). "A Next Click Recommender System for Web-based Service Analytics with Context-aware LSTMs." In: *Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS2020)*. IEEE, pp. 1542–1551.

Wolf, V. and D. Beverungen (2019). "Conceptualizing the Impact of Workarounds-An Organizational Routines'Perspective." In: *Proceedings of the 27th European Conference on Information Systems (ECIS2019)*, pp. 1–11.

Zainuddin, E. and S. Staples (2016). "Developing a Shared Taxonomy of Workaround Behaviors for the Information Systems Field." In: *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS2016)*. IEEE, pp. 5278–5287.