UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Software Engineering Curriculum

Lino Moises Mediavilla Ponce

Discovery of Improvement Opportunities in Knock-out Checks of Business Processes

Master's Thesis (30 ECTS)

Supervisor(s): Fredrik Milani, PhD
Manuel Camargo, PhD
Katsiaryna Lashkevich, PhD (c)

Tartu 2022

Discovery of Improvement Opportunities in Knock-out Checks of Business Processes

Abstract:

Business processes such as loan applications or university admissions usually contain a set of activities known as knock-out checks, which classify cases into two groups: accepted and rejected. When a knock-out check rejects a case, all the work previously performed on it is considered a waste. This waste can be reduced by modifying how the checks are performed. Previous studies have provided heuristics and tools for identifying such improvement opportunities from process models at design-time, while others have proposed predictive, *black-box* models for reordering knock-out checks at run-time. However, they have not provided a method for obtaining knock-out check insights and improvement opportunities directly from existing processes' data. Here, we show a data-driven, decision-rules-based approach for discovering improvement opportunities related to the knock-out checks of business processes. Experiments on synthetic and real-world event logs show that the approach successfully identifies improvement opportunities while attaining a performance comparable to *black-box* approaches. Moreover, by leveraging interpretable Machine Learning techniques, our approach provides further insights into the knock-out checks of business processes that *black-box* approaches do not.

Keywords: Process mining, Improvement opportunities identification, Knock-out checks, Decision Rules

CERCS: P170 - Computer science, numerical analysis, systems, control

Parandamisvõimaluste avastamine äriprotsesside Knock-out kontrollides

Lühikokkuvõte:

Sellised äriprotsessid nagu laenutaotlused või ülikoolide vastuvõtmine sisaldavad tavaliselt tegevusi, mida tuntakse kui "knock-out kontrolle", mis liigitavad juhtumid kahte gruppi: vastuvõetud ja tagasi lükatud. Kui knock-out-kontroll lükkab juhtumi tagasi, loetakse kogu varem sellega tehtud töö raisku. Seda raiskamist saab vähendada, kui muuta kontrollide läbiviimise viisi. Varasemad uuringud on pakkunud heuristikat ja vahendeid selliste parandamisvõimaluste tuvastamiseks protsessimudelite abil projekteerimise ajal, samas kui teised on pakkunud prognoosivaid, *musta kasti* mudeleid väljalangevuskontrollide ümberjärjestamiseks tööajal. Kuid nad ei ole pakkunud meetodit, kuidas saada välja lülitatud kontrollide kohta teavet ja parendusvõimalusi otse olemasolevate protsesside andmetest. Siinkohal näitame andmepõhist, otsustusreeglitel põhinevat lähenemisviisi äriprotsesside knock-out kontrollidega seotud parendusvõimaluste leidmiseks. Katsed sünteetiliste ja reaalsete sündmuste logide põhjal näitavad, et lähenemisviis tuvastab edukalt parendusvõimalusi, saavutades samas tulemuslikkuse, mis on võrreldav *musta kasti* lähenemisviisidega. Lisaks sellele annab meie lähenemisviis tõlgendatavaid masinõppe meetodeid kasutades täiendavat teavet äriprotsesside väljalangemise kontrollide kohta, mida *musta kasti* lähenemisviisid ei võimalda.

Võtmesõnad: Protsesside kaevandamine, parendusvõimaluste tuvastamine, Knock-out kontrollid, otsustusreeglid

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

List of Figures

List of Tables

1 Introduction

Every organization has processes. These are collections of events, activities and decisions triggered by needs and leading to outcomes that are of value to the organization's customers [7]. For instance, vendors perform *order-to-cash* processes, which start when a customer submits a purchase order, and end when the requested product or service has been delivered and the customer has made the corresponding payment. Another example is that of *application-to-approval* processes such as loan applications or municipal permits, which start when someone applies for a benefit/privilege, and end when said benefit/privilege is either granted or denied [7].

Having better processes, and executing them better, can help an organization outperform its competition [7]. Thus, to remain effective and competitive, organizations need to understand, manage and continuously improve their processes [9].

A commonly addressed issue in process improvement is minimizing process wastes [14]. Overprocessing waste is one of these, and it occurs in business processes when some effort is spent but no value is provided to the customer or the business [31].

There is a recurrent overprocessing pattern in processes that contain *knock-out checks* [31], which are activities that classify cases as "accepted" or "rejected". When a case is rejected, all the work previously performed on it is considered a waste [31]. Knock-out checks are commonly found in *application-to-approval* processes such as those in banks, insurance companies, governmental departments, and administrations of multinationals [29] [31].

Nowadays, many organizations use process-aware information systems (PAISs) which record information about the execution of their processes and enable the extraction of *event logs* [8]. Process Mining techniques allow organizations to obtain valuable insights into their processes from these event logs [28]. But even with the available Process Mining tools, the discovery of improvement opportunities is mostly driven by the experience and intuition of analysts. In these conditions, large portions of the opportunity solution spaces can go unexplored [9] [8]. To address this issue, research efforts are being made toward developing frameworks, algorithms and tools for assisted and automated discovery of business process improvement opportunities.

This thesis focuses on the problem of overprocessing waste in business processes due to sub-optimal execution of knock-out checks, with a vision toward automated discovery of process improvement opportunities.

Previous studies provided valuable heuristics for the redesign of knock-out checks in business processes [29] [26] but they did not focus on identifying improvement opportunities directly from from event logs. Others have proposed techniques and tools for identifying improvement opportunities in assisted and even automated fashions [9][21] [22] [27], but their approaches take *as-is* process models in graphical notations as the starting point instead of existing process data. Furthermore, Verenich et. al. [31] proposed run-time knock-out checks reordering with predictive models. However,

such techniques provide limited insights due to their *black-box* nature. In other words, their approach provides optimal knock-out check configurations on a case-by-case basis, but does not provide explanations that could be, for instance, exploited by analysts for redesigning processes and systematically addressing the inefficiencies.

Therefore, we have identified a gap in data-driven improvement opportunity discovery for knock-out checks. To address this gap, we ask (RQ1) *how can improvement opportunities related to knock-out checks be identified and analyzed from event logs?*

To answer this question, we propose a data-driven, decision-rules-based approach for discovering improvement opportunities in the execution of knock-out checks within business processes. This approach and its implementation are the contribution of this thesis. It consists of three steps:

1. Identifying knock-out checks from event logs.
2. Analyzing the impact of the knock-out checks on overprocessing waste.
3. Identifying improvement opportunities.

Experiments on synthetic and real-world event logs show that the approach successfully identifies knock-out check improvement opportunities while attaining a performance comparable to *black-box* approaches. Moreover, the decision-rules aspect of the approach provides further insights into the knock-out checks of business processes that *black-box* approaches do not.

The structure of the thesis is as follows: Sections 2 and 3 provide the theoretical background and related work. Section 4 presents the chosen research method. Section 5 describes the approach for discovering knock-out check improvement opportunities. Section 6 covers the evaluation of the approach, and Section 7 contains the conclusion and future work.

2 Background

This section introduces the concepts used in this thesis, namely: business process management, knock-out checks in business processes, process mining, and machine learning.

2.1 Business Process Management (BPM)

Business Process Management (BPM) is a discipline which encompasses principles, methods and tools to analyze, redesign, implement and monitor business processes [7]. Some potential benefits for organizations adopting BPM include performance improvement, cost reductions, and greater customer satisfaction [5]. The applicability

of BPM in an organization depends on the degree of *process thinking* that is practiced within it [7].

Process Thinking. Up until the 1980s, most organizations focused their optimization efforts on particular business functions rather than on their processes as a whole. Then came a shift in thinking and an initial rise to popularity of the discipline then called Business Process Redesign (BPR). This initial enthusiasm faded away due to reasons such as concept misuse, over-radicalism instead of gradual adoption, and immaturity of the supporting IT tools. However, empirical studies carried out over the years reported that process-oriented organizations showed overall better performance and less internal conflicts. This, coupled with technological developments and the adoption of Process-Aware Information Systems (PAISs), gave rise to a renewed interest in process thinking, BPR, and ultimately BPM [7].

Business Process Redesign (BPR). Nowadays, BPR identifies a set of techniques used in the context of BPM for planning and organizing processes to address previously identified process-related issues and to improve process performance [7] [9]. It is considered the most value-adding stage in the BPM Lifecycle [9], an overview of which is presented in Figure 1. For a more in-depth discussion of the BPM Lifecycle, the reader may to refer to [7].

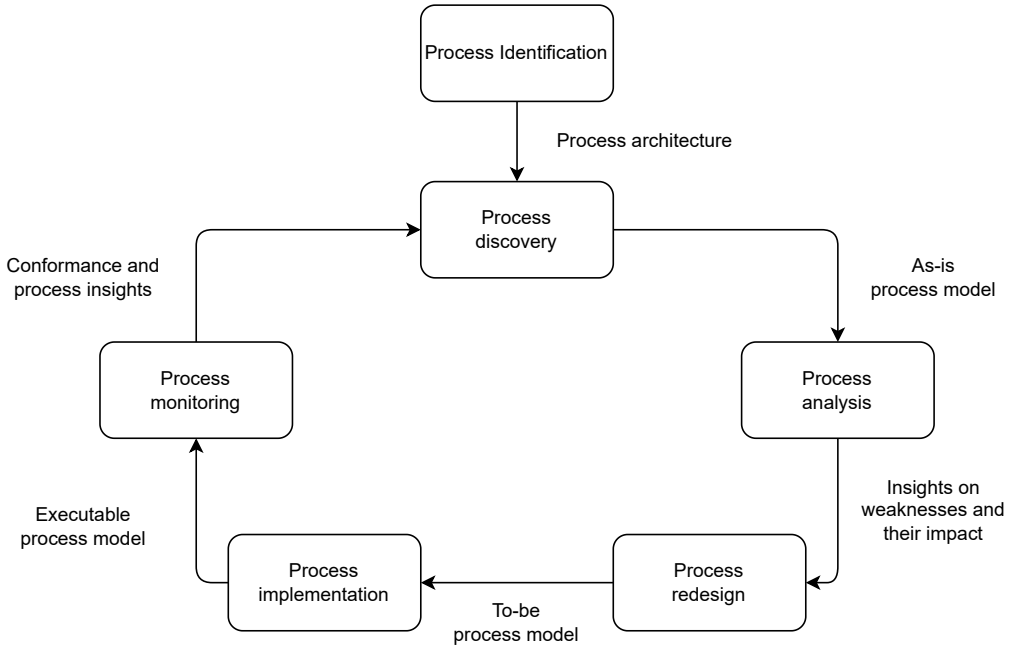

Figure 1. Overview of the BPM Lifecycle [7]

Process performance metrics. BPM activities require clear process performance metrics. There are four dimensions along which specific performance measures can be

defined: time, cost, quality and flexibility. The first three are related to the desire of organizations to make their processes faster, cheaper and better, whereas the fourth one corresponds to the flexibility to adapt to sudden changes in the operating environment [7]. This thesis is primarily focused on the time dimension. Two particularly relevant performance measures in this context, defined from the perspective of a *case*, are:

- *Processing time*: the time during which process participants (e.g. human resources, software or machinery) actually perform work to advance the processing of the case [7].
- *Waiting time*: the time that the case spends *idle*. That is, when its processing remains paused. This can happen due to various factors such as resource unavailability, synchronization, waiting for external input, etc. [7].

2.2 Knock-out checks in Business Processes

Application-to-Approval processes can be found in government agencies, banks, insurance companies, universities and administrations of multinationals [29]. For instance, the process of obtaining building permits at a government agency and the admissions process at a university are both *application-to-approval* processes [7].

These processes start when a customer applies for a benefit or privilege, and end when it is granted or denied [7]. The classification of individual cases as "accepted" or "rejected" is usually achieved through activities known as *knock-out checks* [31]. When one of these checks rejects a case, the work previously performed on it is considered a waste [31]. Lohrmann and Reichert [16] use the term *knock-out principle* to denote an idea deriving from heuristics proposed by Van der Aalst [29] for reducing overprocessing waste by resequencing knock-out checks. The principle suggests to order the checks by "least effort to reject" [31].

2.3 Process Mining

Process Mining denotes a collection of techniques that allow to extract valuable process-related information from *event logs* that contain execution data of business processes [28]. These techniques can be used in most phases of the *BPM lifecycle* to support the discovery, monitoring and improvement of processes directly from what is reflected in the data instead of being based on assumptions [13] [30].

Event Logs. Technological advancements in recent decades have enabled an enormous growth in the amount of data stored and exchanged electronically. Thanks to this, the digital world has become more and more aligned with the real world, and in particular with processes in organizations. This alignment makes it possible to record and analyze *events*. Process mining techniques assume that process *events* can be recorded

sequentially such that each one refers to an *activity* (well-defined step of a process), and is related to a particular case (an instance of a process) [30]. Further insights can be extracted when the logs contain additional information such as the responsible *resource*, *timestamps* and *data elements* (event or case attributes), for instance the loan amount in the case of a loan application process.

Process Mining perspectives. Process Mining may cover different perspectives. The *control flow perspective* has the goal of finding representations of the activity ordering and all possible paths in processes [30]. The *organizational perspective* instead has the goal of discovering organizational structures and classifying resources in terms of roles and organizational units [30]. Moreover, the *time perspective* uses event timestamps to discover bottlenecks, monitor resource utilization and predict remaining processing times [30]. Finally, the *case perspective* is useful for characterizing cases by the values of their *data elements* or *attributes* [30]. The approach presented in this thesis utilizes the *control flow*, *time* and *case* perspectives, as will be discussed in Section 5.

2.4 Machine Learning

Machine Learning is a set of methods for automatically detecting patterns in data, and using them to predict future data or to support decision making under uncertainty [20]. Machine Learning is divided into three categories: ***Supervised Learning*** (learning a function from inputs $\boldsymbol{x}$ to output y, where $\boldsymbol{x} = (x_1, ..., x_p)$ is a p-dimensional *feature vector* and y is the *target variable*), ***Unsupervised Learning*** (learning unknown patterns from the data), and ***Reinforcement Learning*** (learning *optimal policies* to maximize a *reward* in a given *environment*).

A knock-out check can be modelled with a binary classifier (using *Supervised Learning* techniques) where, for a given *case*, the feature vector $\boldsymbol{x}$ is composed of *case attributes* and the target variable $y \in \{negative, positive\}$ indicates if the case successfully passes the check or not [28]. Therefore, the rest of this section focuses on Supervised Learning algorithms for addressing binary classification.

2.4.1 Classification Algorithms

Decision Trees. The Decision Tree (DT) learning algorithm recursively splits the input space, aiming for high-purity subsets in terms of class labels [28]. A model obtained with the DT algorithm can be represented as a tree, where each node represents a splitting condition that determines the branch that a given training instance belongs to. This algorithm supports binary and multi-class classification, and it is widely used due to its simplicity and interpretability. However, DTs are prone to overfitting, and the number of nodes can grow exponentially if the tree depth hyperparameter is not properly tuned [28].

Random Forests. The Random Forest (RF) algorithm aims to mitigate the problem of overfitting usually present in a single Decision Tree by building an *ensemble* of

them. During training, the following procedure is done iteratively: 1) The training set is randomly sampled with replacement, 2) a decision tree is built on the sample. For prediction, a given input passes through all the constructed decision trees and their outputs are averaged to get the final prediction [28]. Random forest models, however, are not considered human-interpretable [6].

Gradient boosted trees. Like random forests, the Gradient Boosted Trees (GBT) algorithm constructs an ensemble of decision trees and takes the average of their outputs to make a prediction. But GBT differs from RF in that GBT constructs the trees sequentially via *boosting* instead of *bagging*. At each new step the decision tree to be constructed aims at correcting the mistakes of the tree at the previous step. In general, Gradient Boosting is currently considered a powerful and state-of-the-art machine learning technique well-suited for problems with heterogeneous features, noisy data and complex dependencies [25]. Popular, open-source implementations of GBT are XGBoost[1] (eXtreme Gradient Boosting) and CatBoost[2] (Categorical Boosting), which improves upon the classic algorithm and other GBT implementations by introducing *ordered boosting* and a novel technique for processing categorical features [25]. Nonetheless, being a tree-based ensemble technique, GBT models are typically inscrutable to humans [6].

Decision Rules. A decision rule is a simple IF-THEN statement that consists of a condition and a prediction [18]. For example: "IF it rains today AND it is April (condition), THEN it will rain tomorrow (prediction)". Their resemblance to natural language makes them a human-interpretable prediction model [18]. Some techniques for obtaining Decision Rules are Incremental Reduced Error Pruning (IREP) [10], Repeated Incremental Pruning to Produce Error Reduction (RIPPER) [3] and the Simple Learner with Iterative Pruning to Produce Error Reduction (SLIPPER) [4]. They are all based on a *sequential covering* strategy, which consists of 1) finding a rule that *covers* a portion of the training instances, 2) saving the rule and removing the covered instances from the training set, 3) repeating the process until there are no more training instances left or a stopping condition is met. In principle, features have to be categorical, so numerical features are usually split in discrete *bins*. Moreover, these type of algorithms are mostly suited for classification tasks, not for regression [18]. There is an open-source implementation of IREP and RIPPER written in Python called wittgenstein[3], which offers out-of-the-box support for numerical & categorical features (performing *binning* for numerical ones), as well as *surrogate model extraction* (for obtaining decision rule models out of black-box models) and an interface compatible with scikit-learn[4], a popular open-source Python library for traditional machine learning tasks.

Many other classification algorithms exist, with varying tradeoffs between complexity and performance, for instance, Logistic Regression, Neural Networks and Support Vector

[1] `https://github.com/dmlc/xgboost`

[2] `https://github.com/catboost/catboost`

[3] `https://github.com/imoscovitz/wittgenstein`

[4] `https://github.com/scikit-learn/scikit-learn`

Machines. For a more complete discussion, the reader can refer to [28] and [20].

2.4.2 Evaluation Measures

Some of the most common metrics for evaluating the performance of binary classification models are computed from the so-called *Confusion Matrix* [28] (see Figure 2) . For a binary classification task, this matrix contains 4 values: the True Positives (TP), which are the test examples correctly classified by the model as positive, the False Positives (FP) where the *ground truth* is negative but the model classifies as positive, the True Negatives (TN) where the model correctly classifies the examples as negative, and False Negatives (FN) where the *ground truth* is positive but the model classifies as negative [28].

Figure 2. Confusion Matrix of a hypothetical model that correctly classifies 3 instances as *positive* (TP = 3), and 4 as *negative* (TN = 4). It does not classify any positive instances incorrectly (FP = 0), but it classifies 1 instance as *negative*, while the ground truth is *positive* (FN = 1).

Using the *Confusion Matrix* values, the following metrics can be defined:

- ***Accuracy:*** the most simple and widely used metric, that measures the overall proportion of correctly classified instances [28]. However, for very imbalanced datasets, e.g. when there are many more instances of one class than of the other, the accuracy value becomes heavily biased and does not provide a complete picture [28]. It is computed in the following way:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

- ***Precision:*** "the ability of the classifier not to label as positive a sample that is negative" [23]. It is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

- ***Recall:*** "the ability of the classifier to find all the positive samples" [23]. It is equivalent to the True Positive Rate (TPR) [28] and it is defined as:

$$Recall = TPR = \frac{TP}{TP + FN} \tag{3}$$

- ***F_1 Score:*** "weighted harmonic mean of the precision and recall" [23]. In fact, it is a particular case of the more general F_β score with $\beta = 1$, which means it gives equal importance to precision an recall [23]. It is defined as:

$$F_1 = \frac{2 * Precision * Recall}{Precision + Recall} \tag{4}$$

- ***False Positive Rate:*** measures the proportion of instances of the *negative* class that are incorrectly predicted as *positive* [28]. It is complementary to the TPR and is defined as:

$$FPR = \frac{FP}{TN + FP} \tag{5}$$

- ***Receiver Operating Characteristic:*** this technique consists of constructing a curve (see Figure 3) and is applicable to classifiers that output real-valued prediction scores (representing the probability of each class) [28]. The Receiver Operating Characteristic (ROC) curve is built with the values of TPR and FPR over all possible decision thresholds (values against which the classifier's prediction scores are compared to determine if instances should be classified as *positive* or *negative*) [28].

- ***Area under the ROC Curve:*** the area under the ROC curve (AUC) condenses information of the ROC curve into a single, threshold-independent measure of a given classifier's performance. Unlike accuracy and the F_1 score, it remains unbiased even for imbalanced datasets. A random classifier would yield a value of $AUC = 0.5$, while a perfect classifier would yield $AUC = 1$ (see Figure 3).

There are two other metrics that are relevant in the context of Decision Rules. For a particular rule, they are defined as:

- ***Support:*** the percentage of instances to which the rule applies. It is necessary just that the rule condition holds, not that the prediction is correct with respect to the ground-truth [18].

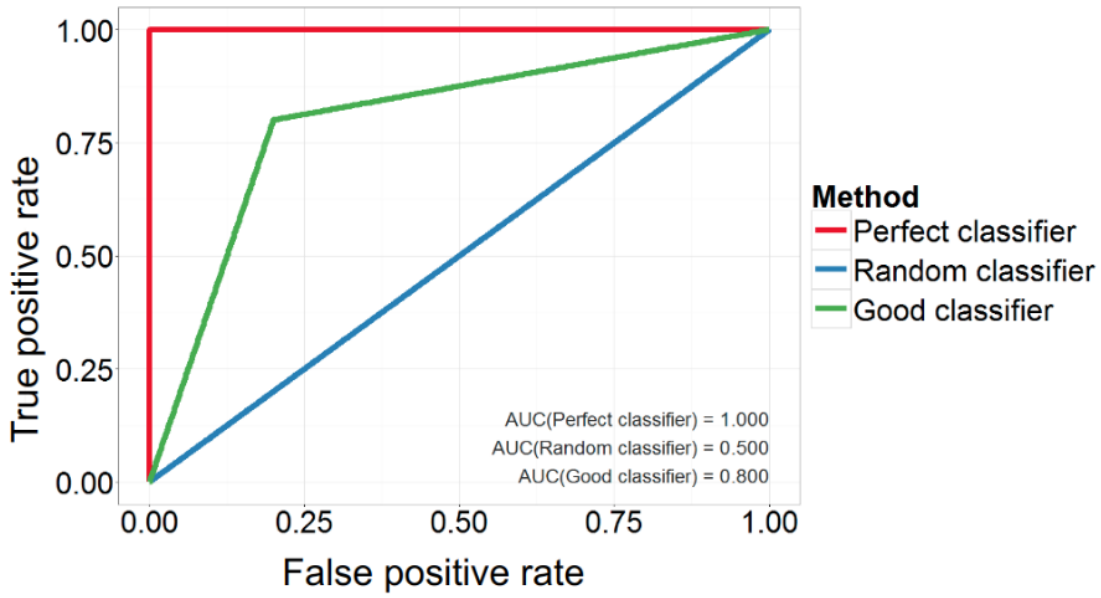

Figure 3. Example ROC curves and AUC values [28].

- ***Confidence:*** the percentage of instances to which the rule applies and the prediction matches the ground-truth [18]. In fact, this corresponds to the *accuracy* metric defined previously.

2.4.3 Interpretability

A system that does not reveal its internal mechanisms is referred to as *black-box*, and in machine learning this describes models that "cannot be understood by inspecting their parameters" [18].

In the context of Business Processes, it is valuable for process workers and analysts that interact with predictive models to understand the reasons behind particular predictions [28]. This can potentially help them make better-founded decisions [28], increase their trust and promote overall adoption of machine learning systems [6].

Interpretable machine learning is a discipline that deals with "providing machine learning models the ability to explain or present their behaviors in terms understandable to humans" [6]. This is particularly challenging because evidence shows there is often a tradeoff between the accuracy and explainability of a model. For instance, recurrent neural network models and ensemble models (e.g. Random Forests or Gradient-Boosted Trees) are considered *black-box* but achieve higher accuracy than simpler models like logistic regression or decision trees on certain problems [28].

There are two approaches for machine learning model interpretability [6]: *intrinsic* interpretability and *post-hoc* interpretability. Intrinsic Interpretability techniques are based on constructing self-explanatory models, i.e. models that incorporate interpretability directly into their structures [6]. This category includes decision trees, decision rules, linear models and attention models [6]. Usually these type of models can provide transparent, undistorted explanations but sacrifice prediction performance to some extent [6] [28]. On the other hand, Post-hoc Interpretability techniques aim at exploiting knowledge from the parameters of an opaque model to present it in human-understandable terms [15]. Usually, this is achieved by creating a second model to provide explanations for an existing model [6]. The former type of techniques are usually better at providing high *explanation fidelity* while the latter aim at preserving the accuracy of the original black-box model [6].

3 Related Work

This section presents an overview of related works. It covers existing approaches for identification of improvement opportunities in business processes, knock-out checks redesign and interpretable machine learning in the context of process mining.

In a 2010 paper, Netjes, Reijers and Van der Aalst [21] presented an approach for "Process Improvement by Creating and Evaluating process alternatives" (abbreviated as PriCE), which consisted of the following steps: 1) finding applicable redesign operations, 2) selecting suitable process parts, 3) creating alternative models and 4) evaluating performance of alternatives. Their goal was to provide a tool to support the full BPM lifecycle, since BPM tools of the time were unable to do so [21]. In fact, this work touched upon several topics that are still actively researched to date, namely: assisted process redesign [9], visualization of process improvements [13] and discovery of simulation models [2].

Similarly, Niedermann and Schwarz [22] introduced a "Deep Business Optimization Platform" that given a process model and process optimization goals, it could compute and recommend improvements using a catalogue of redesign patterns. Souza et. al. [27] proposed heuristics to automate the process redesign pattern selection. They aimed to identify parts of processes where a set of chosen redesign patterns could be applied and provided the algorithms for achieving it. The redesign patterns they addressed were: task composition, activity resequencing (based on data dependencies between activities) and empowering.

More recently, Fehrer et. al. [9], recognizing the limitations of current methods and tools for business process redesign, provided a conceptualization of assisted Business Process Redesign (aBPR) and a classification of redesign recommendation types by level of automation. They also designed a detailed reference architecture intended for state-of-the-art assisted process redesign tools and instantiated it as a software prototype

that supports all the redesigns listed in [26] at varying levels of automation, including knock-out checks.

However, all the implementations of the approaches mentioned so far ([21], [22] and [9]) and the methods proposed in [27] take *as-is* process models as their starting point. Moreover, the work of Souza et. al. [27] considers activity resequencing but it does not specifically address knock-out checks. The approach presented in this thesis is intended for identifying improvement opportunities specifically for knock-out checks and directly from event logs instead of process models. Such data-driven approach could better reflect business processes in practice [13].

Knock-out checks and, more generally, the *knock-out problem* were formally defined in a paper from 2001 by Van der Aalst [29], taking into account information such as resources, resource classes, set-up time of the knock-out checks and synchronization time of knock-out checks performed in parallel. This work also presented a set of heuristics for redesigning business processes containing knock-out checks. Reijers & Limanmansar [26] presented an overview and evaluation of best practices in business process redesign, reiterating the importance of one of the heuristics in [29]. However, it was not the main focus of neither of these works to identify improvement opportunities for knock-out checks directly from event logs. Even though [29] provided an implementation of the proposed approach, it required a process model in Petri-Nets notation as input. This thesis builds upon these definitions and process redesign practices to define and implement an approach for identifying knock-out check improvement opportunities directly from event logs. This could also allow to capture data dependencies, which is an important applicability constraint for redesign patterns mentioned in both [29] and [26].

Verenich et. al. [31] proposed run-time knock-out checks reordering with predictive models, as opposed to the traditional heuristic [29] which suggests reordering based on mean effort and rejection rates. Their approach takes as input an event log and information about the knock-out checks of the process, namely: the knock-out activity names and the disallowed permutations, if any. The presence of start and end timestamps in the input event log allow for a richer analysis (in terms of overprocessing estimation) but the approach can work without them, as shown in their experiments. They tested their approach on two real-world datasets and took as a baseline the traditional heuristic for reordering knock-out checks [29]. The results were a 2.62% overprocessing minimization improvement on one of the datasets and a marginal improvement with the other one, with respect to the baseline. Their approach based on black-box predictive models provides optimal knock-out check orderings on a case-by-case basis, but no further insights into the knock-out checks that could be used for redesigning the process to systematically address the identified inefficiencies. This thesis aims for a more transparent approach, providing also the decision rules of each knock-out check in terms of case attributes and data dependencies detected between activities. This additional information could be exploited for design-time process improvement.

Regarding Interpretable Machine Learning in the context of Process Mining, recently Lee [15] proposed an approach for interpretable prediction of business process outcomes directly from event logs. This work showcased an *intrinsically interpretable* model built using association-rule mining which achieved results comparable to *black-box* models such as those obtained with XGBoost and Random Forests. Association rules were chosen due to their capability of "providing a monotonic representation of relations in a dataset that delivers intuitive comprehension to users" [15]. However, it must be noted that the association rules obtained as part of the approach required significant post-processing before being used for predictions and presented to a user. Moreover, the focus of [15] was on predicting business process outcomes on a case-by-case basis at run-time. Instead, this thesis aims at analyzing process executions from a broader perspective and provide interpretable explanations of knock-out check behaviors over larger periods of time to be used for business process redesigns, not for run-time optimization on a case-by-case basis.

Another recent work involving interpretable machine learning in process mining was [14], which dealt with discovery and analysis of inefficiencies due to batch processing. They too obtained *intrinsically interpretable* models, in this case based on Decision Rules extracted with the RIPPER algorithm[5]. These rules represented the activation rules of *batches* and required notably less post-processing before being presented to the end user, compared to [15]. Mehdiyev and Fettke [17] instead approached the interpretability aspect in predictive process monitoring in a *post-hoc locally interpretable* fashion. Specifically, their approach consisted of training a *black-box* neural network classifier, then, performing k-means clustering to define local regions within the intermediate latent space of the representations obtained in the last hidden layer of the network and finally fitting *surrogate* decision trees to each of these local regions. The decision trees could then be formatted as if-then rules and presented to end users. Even though these two works did not specifically address knock-out checks, the techniques proposed in them are relevant for this thesis because they demonstrate knowledge extraction directly from event logs and how to express it in human-interpretable formats.

4 Methodology

This section presents a description of the Design Science Research Methodology, the justification for having chosen it, and an overview of how its steps are fulfilled in this thesis.

Natural Science research focuses on understanding phenomena and finding "new truths", i.e. why certain things work the way they do [11]. On the other hand, Design Science research focuses on the creation of artifacts for solving problems. These artifacts

[5] `https://github.com/imoscovitz/wittgenstein`

can be can be concepts, models, methods and instantiations [11]. Hevner, et. al. [12] suggested that Design Science research should solve *relevant* problems, and that it should address either *unsolved* problems in unique and innovative ways, or *solved* problems in more efficient or effective ways. Hence, it is popular in fields such as engineering and architecture [11].

A work closely related to this thesis, that of Fehrer et. al. [9], followed the Design Science Research (DSR) method to address the problem of improving Business Process Redesign with assistive tools and proposed a reference architecture and its prototypical implementation as artifacts. Similarly, we consider DSR suitable for this thesis because we aim at *solving a problem* (identification of improvement opportunities in knock-out checks of business processes), and we propose an approach together with its implementation as the resulting *artifacts*.

The DSR methodology described by Peffers, et. al. [24] includes six phases:

1. **Problem identification:** defining the research problem and the value of a solution [11].

   In this thesis, the problem identification phase was performed in Section 1, where we justified the importance of reducing overprocessing waste due to knock-out checks, identified a gap in literature and posed the research question.

2. **Definition of Design Objectives:** specifying the criteria that a solution to the problem should meet [11].

   The gap identified in Section 1 calls for a data-driven approach for improvement opportunity discovery in knock-out checks of business processes. Therefore, we define the following Design Objectives (DOs):

   - **DO 1 (Event log support).** The approach should take event logs instead of process models as input (in this context, this is what we mean by *data-driven*).
   - **DO 2 (Interpretability).** The approach should not be limited to modelling knock-out checks with *black-box* classification techniques but also provide their decision rules.
   - **DO 3 (Identification of improvement opportunities).** The approach should compute improvement opportunities by applying redesign patterns to the knock-out checks of a given process, detecting and taking into account data dependencies between knock-out checks and other activities whenever possible.

3. **Design and development:** creation of artifact(s) to solve the problem [11].

   Section 5 describes all the concepts and steps of our proposed approach, as well as its implementation.

4. **Demonstration:** proving that the artifact works by solving one or more instances of the problem [11].

   We cover this phase in Section 6, in which we design a deliberately inefficient business process, generate a synthetic event log, and apply our approach to find improvement opportunities.

5. **Evaluation:** observing and measuring how well the artifact supports a solution to the problem [11].

   This phase is also covered in Section 6, in which we measure the performance of our approach on a synthetic event log. We also apply the approach on a real-world event log to compare it to an existing baseline, using relevant metrics.

6. **Communication:** communicating the problem, its solution and the utility and effectiveness of the solution to researchers and relevant audiences [11].

   This phase is covered by this public access manuscript[6] and the source code of the implementation of our approach, which is available as a public github repository[7] including installation and usage instructions.

5 Knock-out checks Discovery, Analysis and Improvement Opportunities

This section presents an overview of the approach proposed in this thesis, then it discusses each of its steps in detail, and finally, their implementation.

To fulfill the Design Objectives of Section 4, we propose an approach that consists of three steps:

1. **Knock-out Checks Discovery**. We identify the knock-out checks from an event log (DO 1) and their *knock-out rules* in terms of case attributes (DO 2). This corresponds to step 1 in Figure 4.

   - **Input:** Event Log
   - **Outputs:** Identified Knock-out checks and their knock-out rules

2. **Knock-out Checks Analysis**. For each knock-out check, we compute time waste metrics, and effort-per-rejection. This corresponds to step 2 in Figure 4.

   - **Input:** Identified Knock-out checks and their knock-out rules (from step 1)

[6] `https://comserv.cs.ut.ee/ati_thesis/index.php?language=en`
[7] `https://github.com/AutomatedProcessImprovement/knockouts-redesign`

- **Outputs:** Time waste metrics, effort-per-rejection of each knock-out check and a Knock-out checks analysis report.

3. **Improvement Opportunity Identification**. We compute improvement opportunities by applying redesign patterns (DO 3). This corresponds to step 3 in Figure 4.

    - **Input:** Time waste metrics and effort-per-rejection of each knock-out check (from step 2).
    - **Output:** Improvement opportunities for the knock-out checks.

    When computing improvement opportunities, the approach takes into account data dependencies between knock-out checks and other activities whenever possible. Data dependency detection is done on the basis of the knock-out rules in terms of case attributes obtained in step 1.

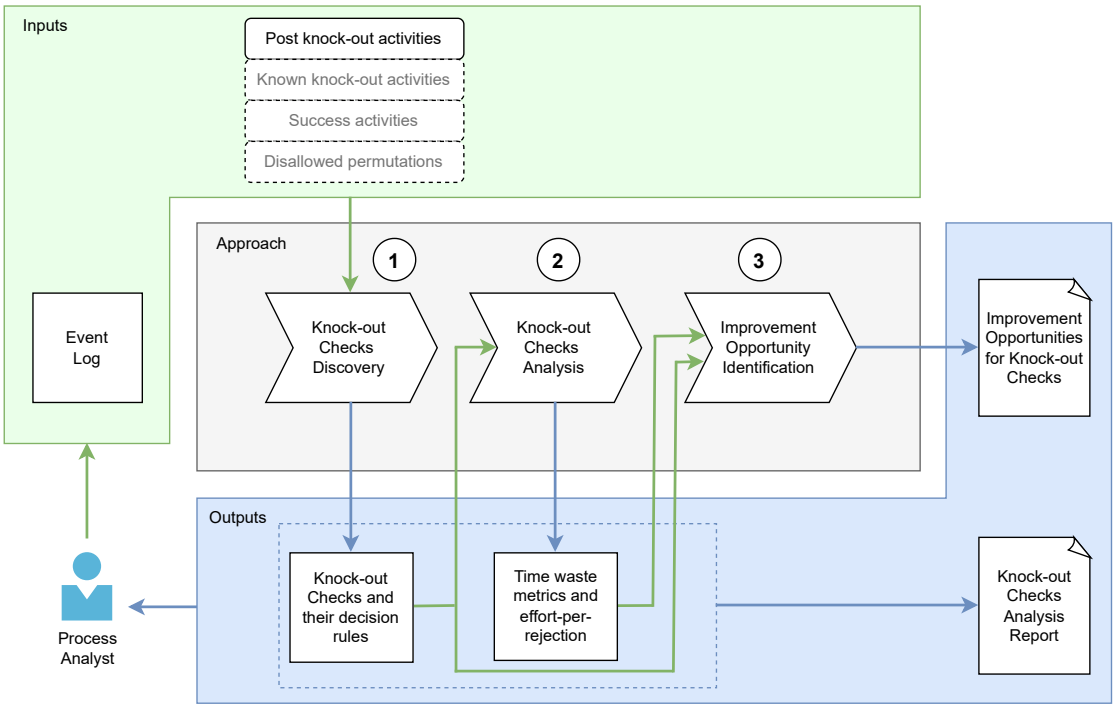

Figure 4. Overview of the proposed approach

5.1 Knock-out check discovery

This step takes an event log of a business process as its starting point. The strictly required fields that the log must contain are: at least one timestamp, case ID and activity name. Technically, the approach could work with event logs containing only these fields, but event logs richer in case attributes are more likely to yield meaningful results.

Since no existing techniques were found in the literature for obtaining the names of the knock-out checks of a process from a given event log, we propose Algorithms 1, 2 and 3. Optionally, users can directly indicate the names of the knock-out activities if they are known. Other information that can be provided to refine the results are the *post knock-out* activities (one or more activities performed immediately after cases are knocked out), *success* activities (one or more activities performed only for cases that are not knocked out) and the *disallowed permutations*, i.e. explicitly prohibited orderings of the knock-out checks (useful for cases when dependencies cannot be discovered from the available data).

We chose Decision Rules (see Section 2.4.1) as the approach for modelling and extracting knowledge out of the knock-out checks due to their intrinsic interpretability, resemblance to natural language and potentially lower complexity of the obtained rules compared to Decision Trees [18].

Algorithm 1: Semi-Automatic knock-out check discovery (variant 1)

Input: Event Log and a list with the names of *post knock-out* activities
Result: A list with the names of the knock-out checks in the process

1 $activities \leftarrow$ `getAllActivities`(log);
2 $relations \leftarrow$ `getAllCombinations`$(activities, postKnockoutActivities)$;
3 $log \leftarrow$ `filterDirectlyFollowsRelation`$(log, relations)$;
4 $log \leftarrow$ `sortByTimestampAscending`(log);
5 $log \leftarrow$ `addColumnWithNextActivity`(log);
6 $log \leftarrow$ `keepRowsIfNextActivityIsIn`$(log, postKnockoutActivities)$;
7 $knockoutChecks \leftarrow$ `getAllActivities`(log);

Algorithm 2: Semi-automatic knock-out check discovery (variant 2)

Input: Event log and a list with the names of *success* activities
Result: A list with the names of the knock-out checks in the process

```
1 startActivity ← getStartActivity(log);
2 relations ← getAllCombinations(startActivity, successActivities);
3 filteredLog ← filterOutEventuallyFollowsRelation(log, relations);
4 knockoutChecks ← AutomaticKnockoutDiscovery(filteredLog);
```

Algorithm 3: Automatic knock-out check discovery

Input: Event Log
Result: A list with the names of the knock-out checks in the process

```
1  variants ← getVariantsSortedByPrefixLengthAscending(log);
2  transitions ← emptyList();
3  for variant in variants do
4      differentiatingTransitions ← emptyList();
5      for transition in getActivityTransitions(variant) do
6          if isNotPresentInOtherVariants(transition) then
7              differentiatingTransitions.append(transition)
8      mostFreq ← findMostFrequentTuple(differentiatingTransitions)
9      transitions.append (mostFreq)
10 knockoutChecks ← getActivitiesFromTransitionTuples(transitions);
```

5.2 Knock-out check analysis

This step takes as input the knock-out checks identified in step 1 and finds answers to the questions: (1) *how much waste is associated to cases rejected by each knock-out check?* and (2) *what is the mean effort associated to each knock-out check?*

To address question (1), we propose the *overprocessing waste*, *processing time waste* and *waiting time waste* metrics, whereas for question (2) we use the *effort-per-rejection* of each knock-out check.

Given an *application-to-approval* process P, a set of resources $R \in P$, a set of activities $A \in P$, a set of knock-out checks $K \subset A$, a set of cases C, a particular case C_i, and a particular knock-out check K_i that **rejects** C_i,

Definition 1. *We define **Processing Time Waste** on case C_i due to knock-out check K_i as*

the sum of the processing times of all activities performed on case C_i, excluding waiting times and the processing time of K_i itself (see Figure 5).

Definition 2. *We define **Overprocessing Waste** on case C_i due to knock-out check K_i as the time elapsed since the case started until it finished, including processing and waiting times but excluding the processing time of K_i itself (see Figure 5).*

Definition 3. *Given the set of cases C_R rejected by K_i and the set of cases C_{NR} not rejected by K_i, we define the **Waiting Time Waste** associated to K_i as the sum of the duration of the intervals (exluding the processing time of K_i) during which C_{NR} cases are held on standby because the resources responsible for performing activities required to advance them are busy performing work on C_R cases (see Figure 6).*

Definition 4. *In this study, we define the **effort-per-rejection** of a knock-out check as the ratio between its average processing time and its rejection rate.*

In our approach, for every knocked-out case, we consider the knock-out check that rejected it as a value-adding activity. Therefore, in such situations we do not count the processing time of that activity toward the time waste metrics.

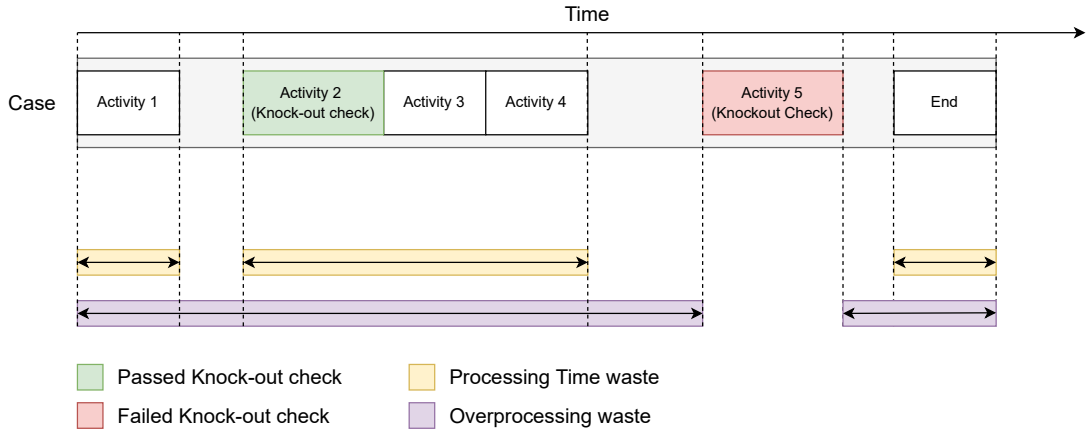

Figure 5. Processing Time Waste and Overprocessing Waste.

In the case of event logs with only one timestamp, such as those used for evaluating the approach in [31], the time waste metrics are omitted, and a constant value of processing time is assumed. That is, the effort-per-rejection of knock-out checks in these situations becomes simply the inverse of its rejection rate, similarly to what was done in [31].

After computing the time waste and effort-per-rejection metrics, we build a *Knock-out Checks Analysis Report* containing the following columns:

- **Knock-out check:** the name of the knock-out check.

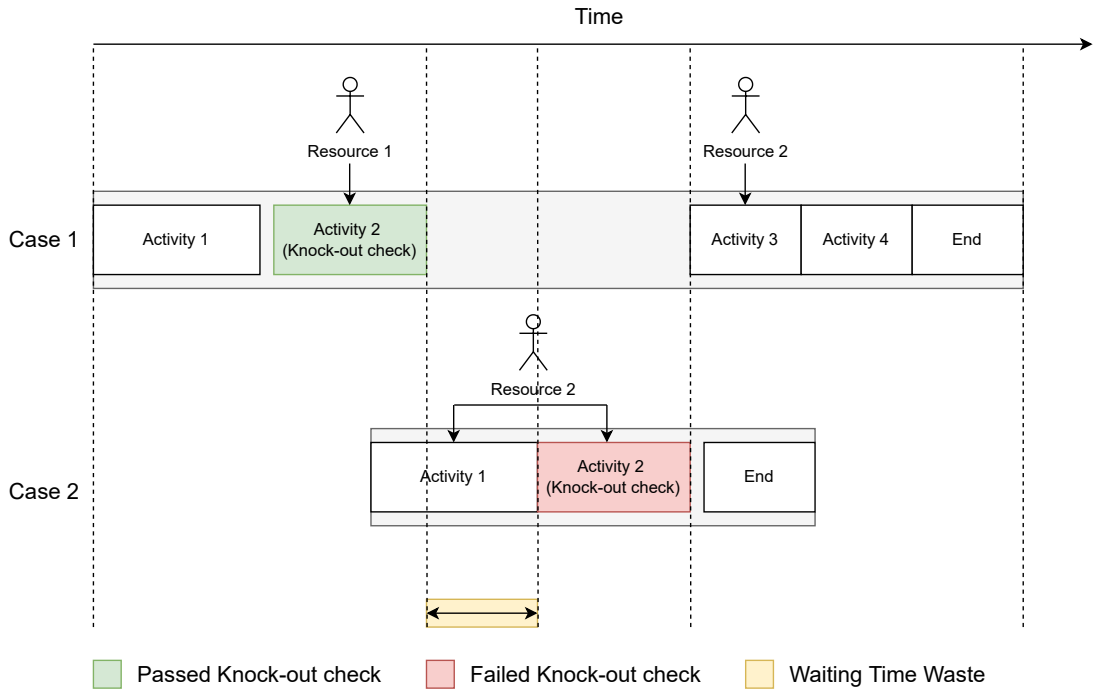

Figure 6. Instance of Waiting Time Waste. *Case 1* (a non-knocked out case) needs *Resource 2* for advancing, but it has to wait because *Resource 2* is busy on *Case 2* (a case to be knocked out).

- **Total frequency:** the total number of cases in which the knock-out check is performed.
- **Case Frequency:** the proportion of cases in which the knock-out check is performed.
- **Rejection Rate:** the rate of cases in which the knock-out check was performed and the result was a rejection.
- **Rejection Rule:** the decision rules obtained for the knock-out check in *disjunctive normal form*, such that if a case satisfies any of them, it is rejected.
- **Effort per rejection**: the effort-per-rejection of the knock-out check, computed according to definition 4.
- **Mean Duration:** the average duration (including processing and waiting time) of the given knock-out check across all the cases where it has been performed.

- **Total Overprocessing Waste**: the total overprocessing waste associated to the knock-out check across all cases where it was performed, computed according to definition 2.
- **Total Processing Time Waste**: the total processing time waste associated to the knock-out check across all cases where it was performed, computed according to definition 1.
- **Mean Waiting Time Waste**: the average waiting time waste associated to the knock-out check, computed according to definition 3. This quantity gives a notion of how much time the cases that are not knocked-out by the given knock-out check have to wait due to cases that are actually knocked-out, in average.
- **Total Waiting Time Waste**: the total waiting time waste associated to the knock-out check. It is similar to the mean waiting time waste, but it is the total sum instead of the average.

5.3 Improvement Opportunities

This step takes as input the identified knock-out checks, their decision rules and the effort-per-rejection of the previous steps to compute improvement opportunities by applying the following redesign patterns:

- ***Knock-out reordering**:* ordering the knock-out checks by least effort to reject [31].
- ***Knock-out relocation**:* moving the knock-out checks as early in the process as the data attributes required by their knock-out rules are available (based on the *Resequencing* pattern [26]).
- ***Knock-out rule change*:** changing the value (or range) of numerical attributes of knock-out rules based on the actual distribution of the values observed in the event log.

A key issue regarding the reordering and relocation (or resequencing) of activities is observed in [26], [29] and [31]. It refers to the fact that usually, some activities depend on data produced by other activities.

We handle this issue by using the data attributes appearing in the discovered decision rules of the knock-out checks (their *knock-out* rules) to perform a search in the log and identify which activity *produces* that value. Specifically, if the knock-out rule of a knock-out check K involves a case attribute that is available (or stops changing) in the log only after another activity A, it is considered that the knock-out check K *depends on* A.

For example, suppose that in a loan application process there is an activity "Assign Risk Score" and a knock-out check "Check Risk Score" with the following knock-out rule: "`Risk Score > 0.5`". Given a log such as that in Table 1, it is possible to identify that `Risk Score` is *produced* by "Assign Risk Score", and that "Check Risk Score" *depends* on "Assign Risk Score".

Table 1. Example Event Log showcasing data dependency

Case ID	Activity	Start	End	Amount	Risk Score
1	Check Documents	07/09/2022 16:36	07/09/2022 16:46	35000	—
1	Assess Application	07/09/2022 16:50	07/09/2022 17:30	35000	—
2	Check Documents	07/09/2022 16:55	07/09/2022 17:50	3000	—
2	Assess Application	07/09/2022 17:55	07/09/2022 18:30	3000	—
1	Assign Risk Score	07/09/2022 18:10	07/09/2022 18:45	35000	0.56
1	Check Risk Score	07/09/2022 18:45	07/09/2022 18:55	35000	0.56
1	Notify Rejection	07/09/2022 19:00	07/09/2022 19:01	35000	0.56

Aditionally, for situations in which it is not possible to detect the data dependencies from the event log with our proposed technique, we allow the user to specify a list of explicitly disallowed knock-out check permutations, as in [31].

5.4 Implementation

As a means for interacting with the implementation of our approach more easily, we provide a basic user interface built with the Streamlit[8] open-source tool (see Figure 7). However, the development of a graphical environment is not the main focus of this thesis and our implementation also exposes a command-line interface (for future integration with other tools). Alternatively, it can be imported into Python scripts as a module.

For event log importing and manipulation we use Pm4py[9], an open-source process mining library written in Python that has been developed by BPM researchers and is intended for use in academia and industry [1].

[8] `https://streamlit.io/`

[9] `https://pm4py.fit.fraunhofer.de/`

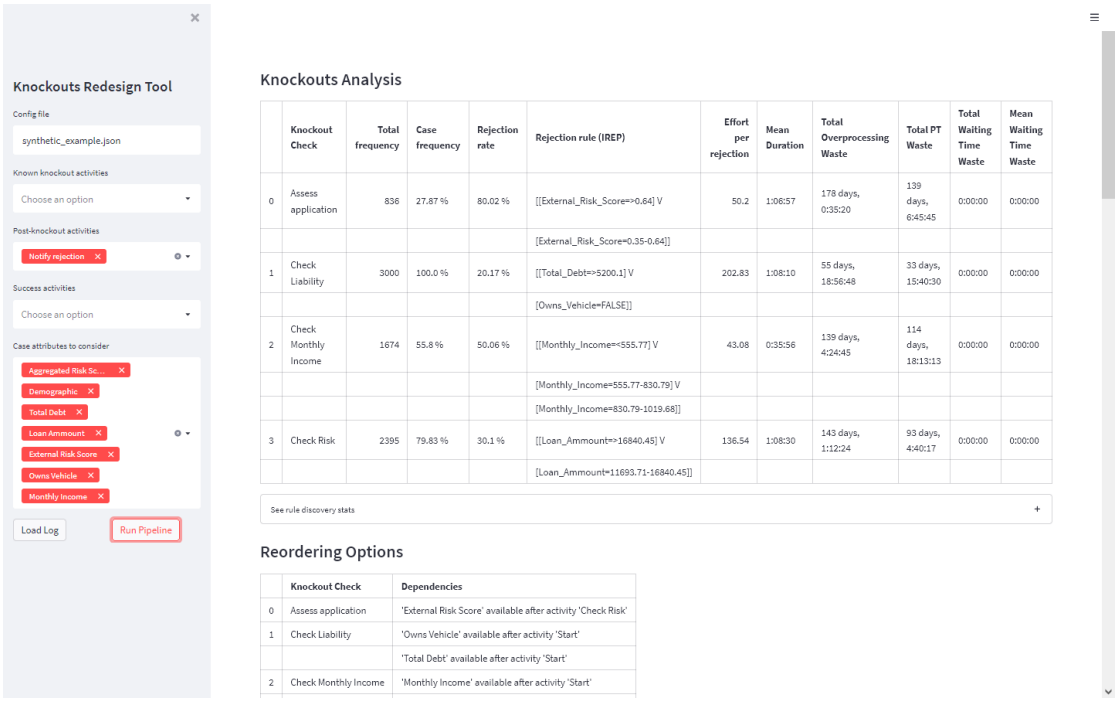

Figure 7. Graphical Interface of the approach implementation

Regarding the identification of knock-out checks in a given process, we support three modes of operation: known knock-out checks (no discovery is performed), semi-automatic discovery with known *post knock-out* activities or *success* activities (Algorithms 1 and 2), and automatic discovery (Algorithm 3).

These modes of operation are aimed at providing maximum flexibility to the user and allowing to integrate available knowledge about the process, since in some cases the knock-out checks may be known beforehand and the user may wish to directly focus on those. In other cases, only the *post-knockout* checks may be known, but providing them as input can improve the quality of the results.

For extracting Knock-out rules, we use an open-source Python library that implements the IREP and RIPPER algorithms called wittgenstein[10], which has been shown to obtain results comparable to the widely used machine learning library scikit-learn[11] in binary classification [19].

We create feature vectors by sorting the events in ascending order by timestamp and aggregating event data at the case level, taking the last available value of the attributes of each case. Afterwards, we train a different Decision Rule model for every knock-out

[10] https://github.com/imoscovitz/wittgenstein
[11] https://github.com/scikit-learn/scikit-learn

check, and obtain a set of rules in *disjunctive normal form* [19] such that, if it evaluates to *True* on a given case encoded as a feature vector, it means the case is labeled as rejected. For example, if a discovered set of rules for a knock-out check called "Check Liability" is "`(Monthly Income < 800) V (Owns Vehicle = False)`" and a certain case has attributes `{Monthly Income: 1200, Owns Vehicle: False, ...}`, the prediction of the decision rule model of "Check Liability" is that this case will be rejected (knocked out).

By obtaining a different Decision Rule model for every knock-out check, we try to answer the question *"given this case, will the knock-out check reject it?"*, thus for every knock-out check we solve a *binary classification* problem. An alternative point of view would be to answer the question *"given this case, which knock-out check will reject it?"*, which may be seen as a *multiclass classification* problem.

Hyperparameter search is done automatically for each Decision Rule model through the grid search functionality provided by scikit-learn [23]. This technique performs an exhaustive search over a pre-defined grid of hyperparameter values searching for the combination that maximizes the performance of a machine learning model [28]. We embedded this in our approach to enable the Decision Rule models to automatically find the best-performing hyperparameter combination (that which maximizes the F_1 score) instead of requiring the users to manually calibrate them for every event log.

Before computing improvement opportunities, the discovered knock-out checks are pre-filtered based on the *confidence* of their decision rule: if it is lower than a user-specified threshold, they are not taken into consideration for the rest of the analysis (users can also choose to keep them in the analysis and receive just relevant warnings instead).

The knock-out check reordering options are obtained by computing the optimal and *dependency-aware* ordering of the checks by applying the *knock-out principle* described in Section 2.2, that is, in ascending order by their effort-per-rejection value, but taking into account any dependencies detected between knock-out checks and disallowed permutations specified by the user, if any.

We then use the detected data dependencies to relocate the knock-out activities throughout the process, i.e. as soon as the case attributes required by their knock-out rules are available. We apply the pattern to a subset of case variants that cover at least a certain percentage of the total cases (specified by the user) and we show the *as-is* and *to-be* order of activities of each of those variants.

Finally, for every numerical attribute appearing in the knock-out rules of the knock-out checks, we compute and show the distribution of the attribute in the cases captured by the log, with an overlay of the range of values covered by the corresponding knock-out rule. This information allows the user to consider adjusting the ranges or values of the knock-out checks in a given process, if appropriate.

6 Evaluation

This section covers phases 4 and 5 of the Design Science Research Methodology introduced in Section 4, namely: demonstration and evaluation. First, we introduce the event logs and the metrics to be used in the experiments, then we demonstrate that the approach meets the Design Objectives on a synthetic event log (Experiment 1). After this, we compare the approach against the baseline (Experiment 2). Finally, we apply the approach to a real-world event log (Experiment 3).

Table 2. Summary of the Event Logs used for evaluation

Log Name	**Type of Process**	**Timestamps**	**Cases**	**Knock-out checks**	
				Name	**Rejection Rate**
Synthetic event log	Credit Application (hypothetical)	Start & End	3000	*Assess application*	80%
				Check Liability	20%
				Check Monthly Income	50%
				Check Risk	30%
Envpermit	Environmental Permit Application	Only End	1230	*T02*	0.4%
				T10	66.2%
				T06	1.2%

6.1 Datasets and Features

6.1.1 Synthetic event log

We built a hypothetical Credit Application process (see Figure 8). It contains four knock-out checks: *Check Liability*, *Check Risk*, *Check Monthly Income* and *Assess application* (see Table 2). Cases that successfully pass all these checks move on to *Make Credit Offer*, those that fail any of these checks pass on to *Notify Rejection*. We modelled the process in Apromore[12] with its BPMN[13] editor functionality, and then we obtained an event log with 3000 cases in XES[14] format by performing a simulation using the parameters described in Figure 9 and Table 3.

At this point the log contained cases with only case id, activity, timestamps and resource information. Some of these cases successfully passed all checks and ended in a Credit Offer, and others were rejected after executing particular knock-out checks,

[12] `https://apromore.com/`

[13] Business Process Modelling Notation, released as a standard by the Object Management Group [7]

[14] eXtensible Event Stream, IEEE Standard for Achieving Interoperability in Event Logs and Event Streams [7]

according to the gateway probabilities configured for the simulation (which correspond to the rejection rates in Table 2). The next step was to "inject" the knock-out rules behavior into the log. We post-processed the log to add case attributes with values that reflected the rules and rejection rates shown in Figure 8 and Table 2.

Additionally, to test our implementation's ability to identify activity data dependencies, we replicated the situation described in Table 1 by selectively removing from all cases in the log the value of the "External Risk Score" attribute such that it becomes available only after "Check Risk" is executed.

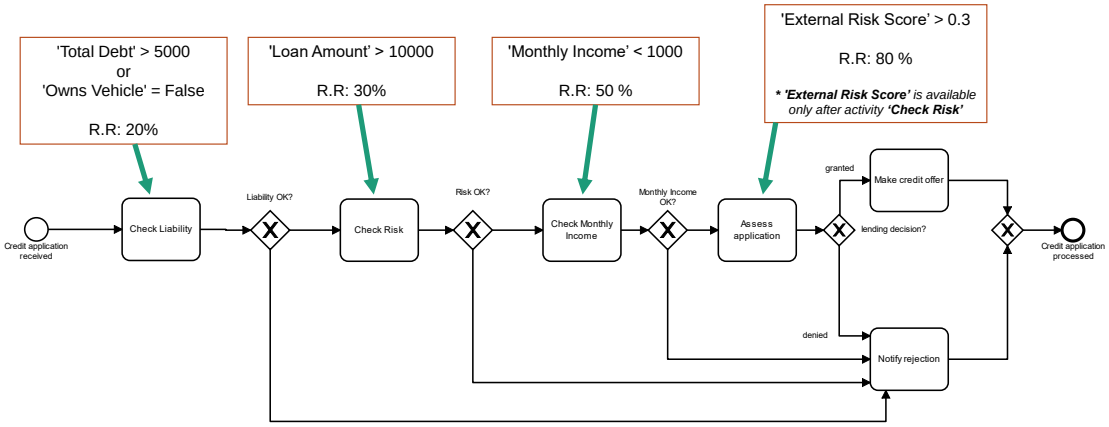

Figure 8. The hypothetical Credit Application process used for evaluation

Table 3. Activity parameters for generating the synthetic event log

Activity	**Resource**	**Distribution**	**Mean**	**Std. Deviation**	**Time unit**
Check Risk	Clerk	Normal	10	5	Minutes
Check Monthly Income	Clerk	Normal	10	5	Minutes
Notify rejection	Credit Office	Normal	10	2	Minutes
Assess application	Clerk	Exponential	20	-	Minutes
Make credit offer	Credit Officer	Normal	10	2	Minutes
Check Liability	Clerk	Normal	15	5	Minutes

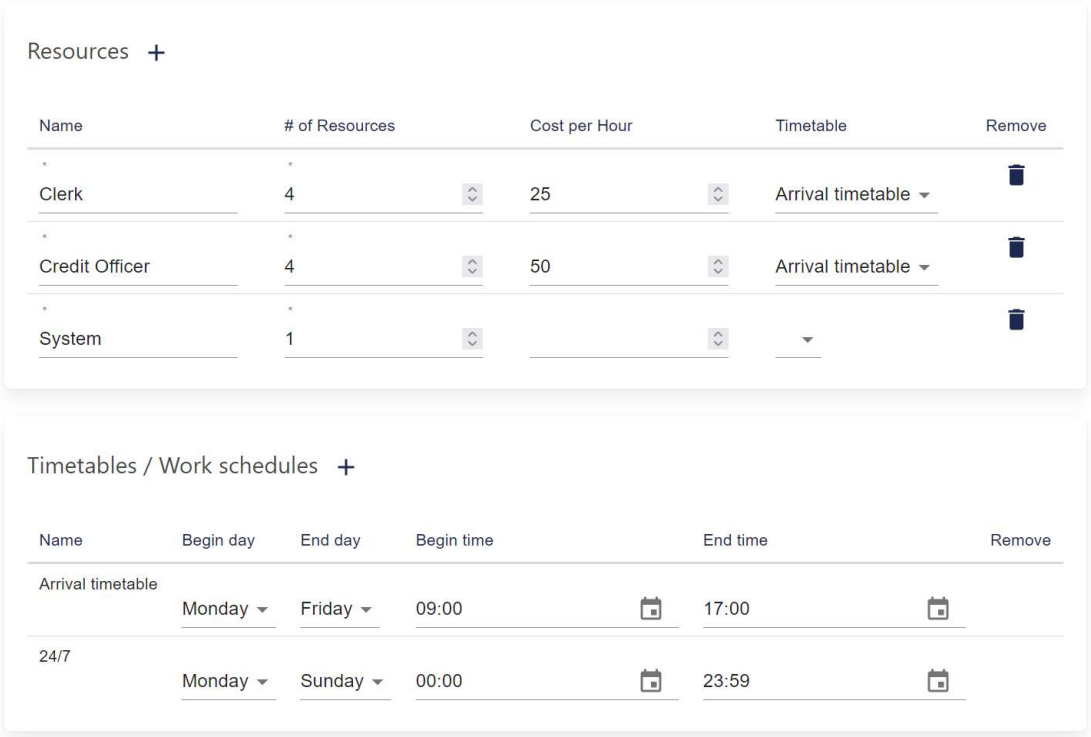

Figure 9. Resource and timetable configuration for generating the synthetic event log

6.1.2 Real-world event log

The second event log we used for evaluating our approach is the same used by Verenich et. al. [31], which we consider as the baseline. It corresponds to an environmental permit application process in a Dutch municipality (see Figure 10).

This process features three knock-out activities: *T02-check confirmation of receipt*, *T06-determine necessity of stop advice*, and *T10-determine necessity to stop indication* (see Table 2). As indicated in [31], these checks are not completely independent; *T10* can only be done after either *T02* or *T06* has been performed. But any other permutation that respects this constraint is allowed.

Cases in this log contain the following data attributes: channel by which the case has been lodged, department that is responsible for the case, responsible resource and its group [31]. Additionally, the log contains only event completion timestamps.

6.2 Metrics

To quantify the performance of the Decision Rule models obtained for the knock-out checks, we use standard metrics for binary classification performance (as defined in

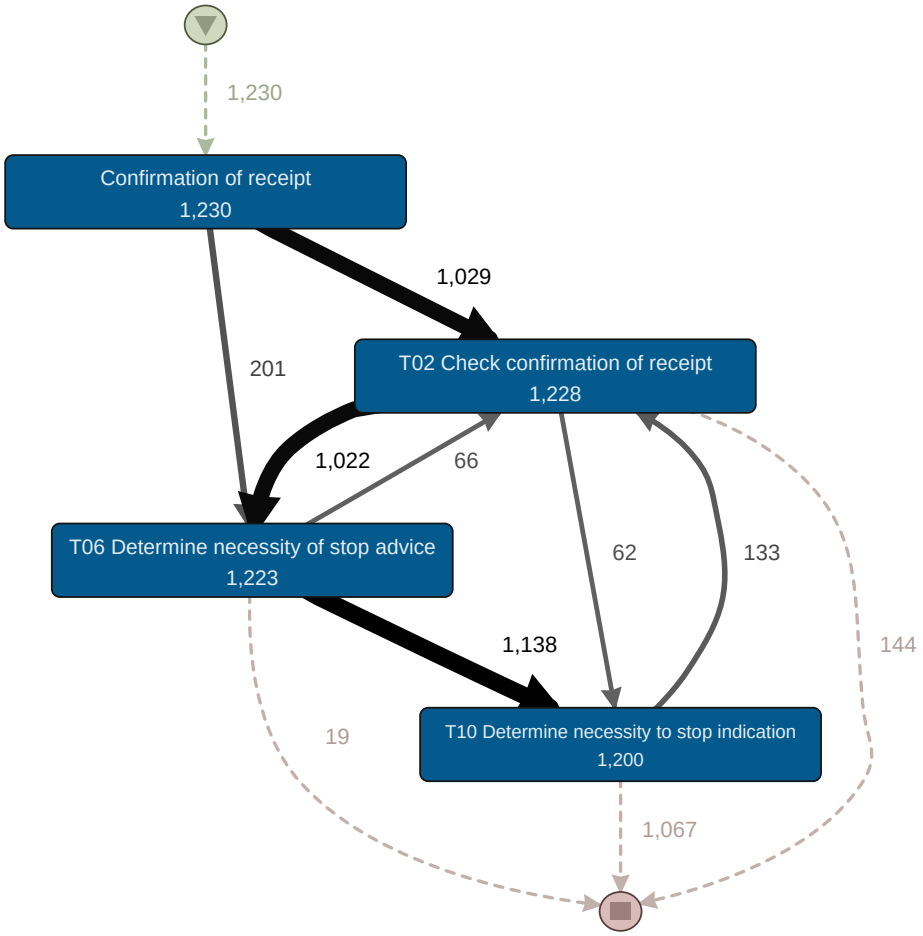

Figure 10. Process map extracted from the Environmental permits log

Section 2.4.2), specifically: the ROC curve and the area under it (AUC). We also report the confidence and support of the learned decision rules.

The correctness of the calculation of time-waste metrics is verified by a set of unit tests included in the code repository of the implementation of our approach[15]. In the following experiments we report these metrics as part of the knock-out analysis results but we do not count them toward any specific performance evaluation.

[15] `https://github.com/AutomatedProcessImprovement/knockouts-redesign`

6.3 Experiment 1: Demonstration of the approach with the synthetic event log

The objective of this experiment is to confirm the compliance with the Design Objectives introduced in section 4 and to verify that the approach is able to capture the patterns we deliberately introduced into the synthetic event log.

6.3.1 Experimental setup

As indicated when introducing the synthetic event log, we *injected* the rules and case attributes shown in Table 4 into it, with the intention to verify how well these patterns are captured by our approach.

Table 4. Knock-out rules injected in the synthetic event log

Knock-out check	**Knock-out rule**
Assess application	External Risk Score > 0.3
Check Liability	(Total Debt > 5000) V (Owns Vehicle = False)
Check Monthly Income	Loan Amount > 10000
Check Risk	Monthly Income < 1000

We split the dataset using 80% of the cases for training and 20% for testing, applying Temporal holdout splitting as recommended for time-series data [28].

The classification performance metrics were obtained through cross-validation but keeping the Temporal holdout paradigm and leveraging the scikit-learn functionality for Time series cross validation[16].

6.3.2 Results

We report the confidence and support of the discovered rules in Table 5, the resulting ROC curves and AUC values averaged over five cross-validation folds (Figure 11), and the outputs of the phases of our approach (Figures 12, 13, 14 and 15).

[16] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.TimeSeriesSplit.html`

Table 5. Confidence and support of the rules discovered from the synthetic event log

Knock-out check	Rule	Confidence	Support
Assess application	[[External_Risk_Score=0.36-0.64] V [External_Risk_Score=>0.64]]	1.000	0.732
Check Liability	[[Total_Debt=>5219.85] V [Owns_Vehicle=FALSE]]	1.000	0.193
Check Monthly Income	[[Monthly_Income=<564.21] V [Monthly_Income=564.21-830.79] V [Monthly_Income=830.79-1020.15]]	0.933	0.536
Check Risk	[[Loan_Ammount=11647.32-16709.71] V [Loan_Ammount=>16709.71]]	1.000	0.252

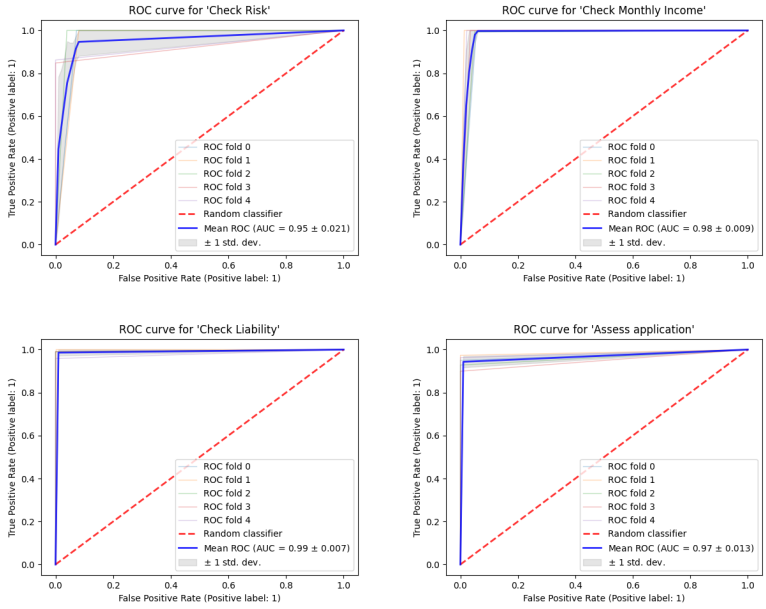

Figure 11. ROC curves of the Decision Rule models obtained for the knock-out checks of the synthetic event log

Knockouts Analysis

	Knockout Check	Total frequency	Case frequency	Rejection rate	Rejection rule (IREP)	Effort per rejection	Mean Duration	Total Overprocessing Waste	Total PT Waste	Total Waiting Time Waste	Mean Waiting Time Waste
0	Assess application	836	27.87 %	80.02 %	[[External_Risk_Score=0.35-0.64] V	50.2	1:06:57	178 days, 0:35:20	139 days, 6:45:45	0:00:00	0:00:00
					[External_Risk_Score=>0.64]]						
1	Check Liability	3000	100.0 %	20.17 %	[[Owns_Vehicle=FALSE] V	202.83	1:08:10	55 days, 18:56:48	33 days, 15:40:30	0:00:00	0:00:00
					[Total_Debt=>5200.1]]						
2	Check Monthly Income	1674	55.8 %	50.06 %	[[Monthly_Income=555.77-830.79] V	43.08	0:35:56	139 days, 4:24:45	114 days, 18:13:13	0:00:00	0:00:00
					[Monthly_Income=<555.77] V						
					[Monthly_Income=830.79-1019.68]]						
3	Check Risk	2395	79.83 %	30.1 %	[[Loan_Ammount=11693.71-16840.45] V	136.54	1:08:30	143 days, 1:12:24	93 days, 4:40:17	0:00:00	0:00:00
					[Loan_Ammount=>16840.45]]						

Figure 12. Knock-out analysis report for the synthetic event log

Reordering Options

	Knockout Check	Dependencies
0	Assess application	'External Risk Score' available after activity 'Check Risk'
1	Check Liability	'Owns Vehicle' available after activity 'Start'
		'Total Debt' available after activity 'Start'
2	Check Monthly Income	'Monthly Income' available after activity 'Start'
3	Check Risk	'Loan Ammount' available after activity 'Start'

Optimal Order of Knock-out checks (taking into account attribute dependencies):

1. Check Monthly Income
2. Check Risk
3. Assess application
4. Check Liability

0/167 non-knocked-out case(s) follow it.

Figure 13. Knock-out reordering options for the synthetic event log

Relocation Options

	Cases Covered	As-is / To-be
0	27.9 %	Start -> Credit application received -> Check Liability -> Check Risk -> Check Monthly Income -> Notify rejection -> Credit application processed -> End
		Start -> Credit application received -> Check Monthly Income -> Check Risk -> Check Liability -> Notify rejection -> Credit application processed -> End
1	24.0 %	Start -> Credit application received -> Check Liability -> Check Risk -> Notify rejection -> Credit application processed -> End
		Start -> Credit application received -> Check Risk -> Check Liability -> Notify rejection -> Credit application processed -> End
2	22.3 %	Start -> Credit application received -> Check Liability -> Check Risk -> Check Monthly Income -> Assess application -> Notify rejection -> Credit application processed -> End
		Start -> Credit application received -> Check Monthly Income -> Check Risk -> Assess application -> Check Liability -> Notify rejection -> Credit application processed -> End
3	20.2 %	Start -> Credit application received -> Check Liability -> Notify rejection -> Credit application processed -> End
		Start -> Credit application received -> Check Liability -> Notify rejection -> Credit application processed -> End
4	5.6 %	Start -> Credit application received -> Check Liability -> Check Risk -> Check Monthly Income -> Assess application -> Make credit offer -> Credit application processed -> End
		Start -> Credit application received -> Check Monthly Income -> Check Risk -> Assess application -> Check Liability -> Make credit offer -> Credit application processed -> End

Figure 14. Knock-out relocation options for the synthetic event log

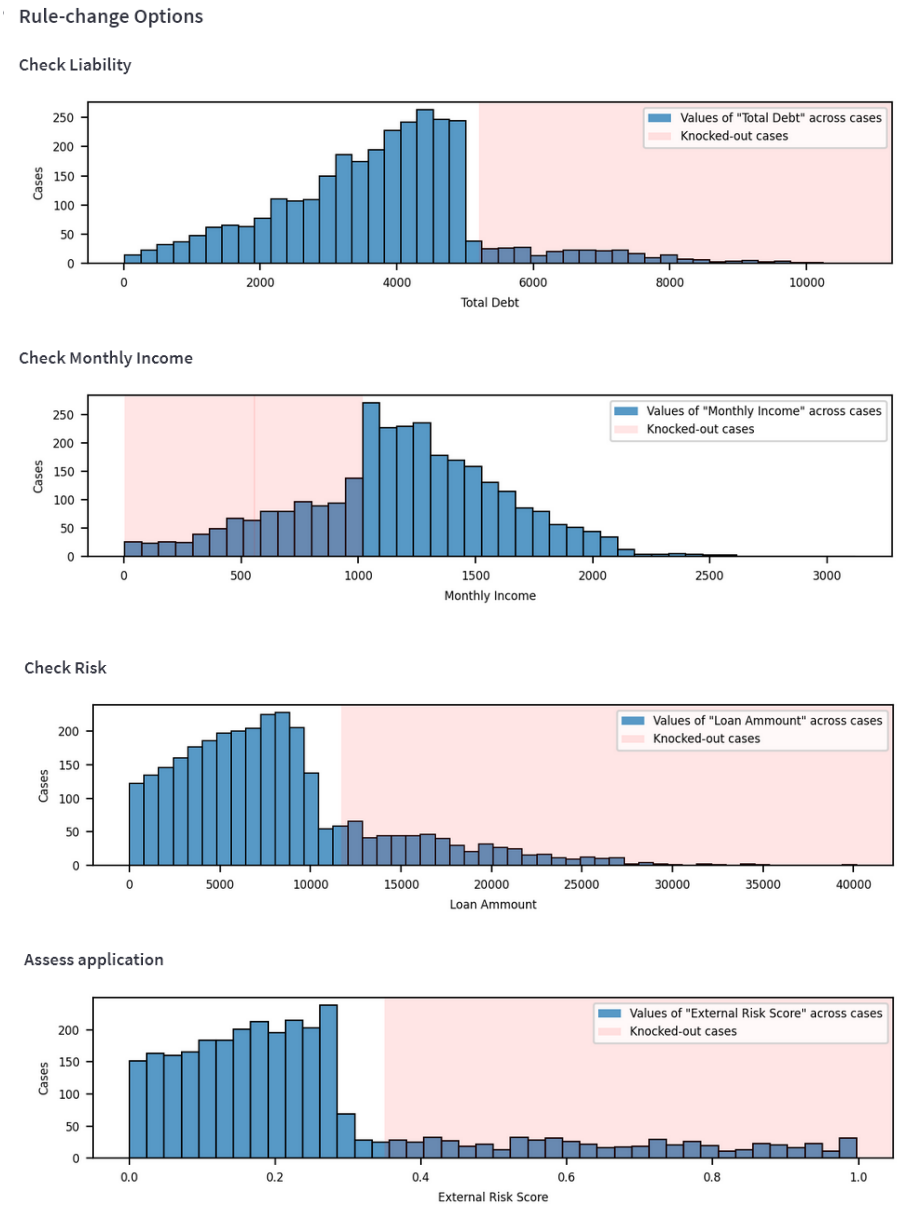

Figure 15. Knock-out rule change options for the synthetic event log

6.3.3 Discussion

From Figure 11, we observe particularly good ROC curves and AUC values near 1.00 (perfect classifier). This, together with the high values of confidence (Table 5) and the fact that the discovered rules shown in the *Knockouts Analysis* output (Figure 12) are coherent with the patterns injected in the log (Table 4), confirms that the classification models perform as expected.

Moreover, from the *Reordering options* output (Figure 13) we observe that the approach was able to capture the data dependency that we inserted, and took it into account for computing the reordering and relocation suggestions. Specifically, the fact that "Assess application" depends on an attribute *produced* by "Check Risk" imposes a constraint on the reordering. If it were not for this fact, "Assess Application" should be pushed earlier in the process, due to having a smaller effort-per-rejection than "Check Risk" and "Check Liability".

This captured data dependency extends to the *Relocation options* output (Figure 14), in which we observe the *as-is* and *to-be* configurations of the most representative process variants. In all cases, the re-configurations respect the data dependencies.

In the output of the *Rule Change* options (Figure 15), we observe the plots of the distributions of numerical case attributes appearing in the knock-out rules of each knock-out check, with an overlay of the value ranges of knocked-out cases. These plots are coherent with the rules we injected in Table 4.

In this dataset, the positive and negative examples were relatively abundant for all knock-out checks and the rules to discover were of relatively low complexity. Nonetheless, this simple example serves its purpose of demonstrating that the approach captures the patterns we expect it to, and the improvement opportunities it computes are correct.

Thus, with this experiment we have verified that the approach works with an event log (DO1), provides the decision rules of the knock-out checks of a process (DO2) and uses that information to compute improvement opportunities, taking into account data dependencies (DO3).

6.4 Experiment 2: Comparison with baseline using the real-world event log

The objective of this experiment is to compare our approach to that of Verenich et. al. [31]. In their approach, they place an emphasis on prediction accuracy and use *black-box* models. Instead, we place an emphasis on interpretability (connected to our Design Objective #2). Here we compare the classification models of both approaches to determine if there is a performance loss or not.

6.4.1 Experimental setup

For this experiment we use the Environmental Permit log introduced previously.

To replicate the experimental conditions of the baseline as much as possible, we use the same dataset splitting proportions as the baseline, namely 80% of the cases for training and 20% testing.

Verenich et. al. [31] observed that the Environmental permits log is highly imbalanced regarding the proportions of accepted and rejected cases, so they performed *class balancing* by undersampling the accepted (non knocked-out) cases. We followed the same strategy.

As previously stated, we use the ROC curve and AUC metric for comparing the performance of our models with the baseline. Verenich et. al. [31] computed the ROC curves of their resulting classifiers with *5-fold* cross-validation. We note that when working with temporal series data (such as event logs), it is important to ensure that "future data is not used for predicting the past" [28]. So the recommended splitting technique for such cases is *Temporal holdout splitting* [28]. However, to ensure the same experimental conditions, for this particular experiment we performed the same *5-fold* cross validation procedure using just *stratified* random-sampling based splitting (to keep the classes balanced at all times).

We do not perform an explicit comparison of overprocessing reduction because our approach uses the traditional reordering strategy based on the *knock-out principle* [26], against which the baseline already reported a marginal improvement in the case of the Environmental Permit log (see Section 3).

6.4.2 Results

Here we show the resulting ROC curves and AUC values averaged over the five cross-validation folds (Figure 16). We observe AUC values overall ranging from 0.571 (*T10*) to 0.744 (*T02*) but with relatively high standard deviations.

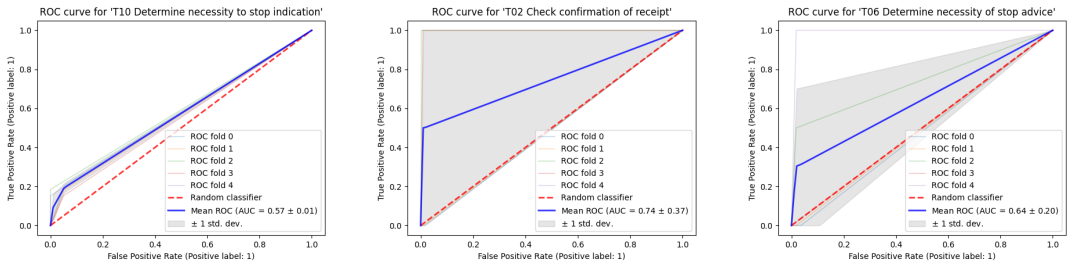

Figure 16. ROC curves of the Decision Rule models obtained for the knock-out checks of the Environmental Permit log

6.4.3 Discussion

The observed AUC values in Figure 16, ranging from 0.571 (*T10*) to 0.744 (*T02*) seem encouraging to a certain extent, specially considering that the AUC values obtained in the baseline study for the Environmental permits log were instead in the range of 0.527 (*T06*) to 0.645 (*T10*) [31].

However, we also observe relatively high values of standard deviation. For instance, the classifier for *T02* obtained the highest mean AUC value: 0.744, but a standard deviation of 0.37. This can be due to the fact that the positive examples available for training the rule model of this knock-out check are relatively very few (*T02* rejected only the 0.4% of all cases, as opposed to *T10* which rejected 64.6%).

Nonetheless, the performance of our approach is comparable to that of the baseline despite the peculiarities of this dataset, and the evidence does not show a significant performance loss by using an interpretable decision-rule-based model instead of black-box models.

6.5 Experiment 3: Demonstration of the approach with the real-world event log

After confirming that the approach works with a synthetic event log (Experiment 1), and that there is no significant loss of performance by using an interpretable decision-rule-based model (Experiment 2), we apply our approach to the Environmental Permit event log with the objective of confirming the compliance with the Design Objectives introduced in section 4, this time on a real-world log.

6.5.1 Experimental setup

We kept the same dataset splitting proportions as in Experiment 2, but this time we applied Temporal holdout splitting as recommended in [28]. The classification performance metrics were obtained through cross-validation but keeping the Temporal holdout paradigm.

Due to the presence of only one timestamp per event in the Environmental Permit log, we do not compute the time waste metrics defined in Section 2.2.

We specified a confidence threshold of 0.5 and chose the option to generate warnings for low-confidence rules instead of automatically excluding the associated knock-out checks from the analysis (see Section 5.4). Aditionally, we specified the disallowed permutations as in [31].

6.5.2 Results

We report the discovered rules with their confidence and support in Table 6 and the outputs from the tool for the phases of the knock-outs improvement opportunity discovery approach in Figures 17, 18 and 19.

Table 6. Confidence and support of the discovered rules for the Environmental Permit log

Knock-out check	**Rule**	**Confidence**	**Support**
T02	[[org:group=EMPTY] V [org:group=Group4]]	0.286	0.011
T06	[[(case)_responsible=Resource01] V [(case)_responsible=Resource03]]	0.167	0.025
T10	[[org:group=Group1]]	0.983	0.392

Knockouts Analysis

	Knockout Check	Total frequency	Case frequency	Rejection rate	Rejection rule (RIPPER)	Effort per rejection
0	T02 Check confirmation of receipt	1228	99.84 %	0.41 %	[[org:group=EMPTY] V	2.46
					[org:group=Group4]]	
1	T06 Determine necessity of stop advice	1223	99.43 %	1.23 %	[[(case)_responsible=Resource01] V	0.82
					[(case)_responsible=Resource03]]	
2	T10 Determine necessity to stop indication	1200	97.56 %	66.25 %	[[org:group=Group1]]	0.02

Warning: "T06 Determine necessity of stop advice" knock-out rule confidence is below threshold (0.167 < 0.5)

Warning: "T02 Check confirmation of receipt" knock-out rule confidence is below threshold (0.286 < 0.5)

Figure 17. Knock-out analysis report for the Environmental Permit log

Reordering Options

	Knockout Check	Dependencies
0	T02 Check confirmation of receipt	'org:group' available after activity 'T06 Determine necessity of stop advice'
1	T06 Determine necessity of stop advice	'(case) responsible' available after activity 'Start'
2	T10 Determine necessity to stop indication	'org:group' available after activity 'T05 Print and send confirmation of receipt'

Optimal Order of Knock-out checks (taking into account attribute dependencies):

1. T06 Determine necessity of stop advice
2. T10 Determine necessity to stop indication
3. T02 Check confirmation of receipt

132/415 non-knocked-out case(s) follow it.

Figure 18. Knock-out reordering options for the Environmental Permit log

Relocation Options

	Cases Covered	As-is / To-be
0	58.0 %	Start -> Confirmation of receipt -> T02 Check confirmation of receipt -> T04 Determine confirmation of receipt -> T05 Print and send confirmation of receipt -> T06 Determine necessity of stop advice -> T10 Determine necessity to stop indication -> End
		Start -> Confirmation of receipt -> T06 Determine necessity of stop advice -> T04 Determine confirmation of receipt -> T05 Print and send confirmation of receipt -> T10 Determine necessity to stop indication -> T02 Check confirmation of receipt -> End
1	10.0 %	Start -> Confirmation of receipt -> T06 Determine necessity of stop advice -> T10 Determine necessity to stop indication -> T02 Check confirmation of receipt -> T04 Determine confirmation of receipt -> T05 Print and send confirmation of receipt -> End
		Start -> Confirmation of receipt -> T06 Determine necessity of stop advice -> T04 Determine confirmation of receipt -> T05 Print and send confirmation of receipt -> T10 Determine necessity to stop indication -> T02 Check confirmation of receipt -> End
2	9.3 %	Start -> Confirmation of receipt -> T02 Check confirmation of receipt -> T06 Determine necessity of stop advice -> T10 Determine necessity to stop indication -> T04 Determine confirmation of receipt -> T05 Print and send confirmation of receipt -> End
		Start -> Confirmation of receipt -> T06 Determine necessity of stop advice -> T04 Determine confirmation of receipt -> T05 Print and send confirmation of receipt -> T10 Determine necessity to stop indication -> T02 Check confirmation of receipt -> End
3	6.1 %	Start -> Confirmation of receipt -> T02 Check confirmation of receipt -> T04 Determine confirmation of receipt -> T06 Determine necessity of stop advice -> T10 Determine necessity to stop indication -> T05 Print and send confirmation of receipt -> End
		Start -> Confirmation of receipt -> T06 Determine necessity of stop advice -> T04 Determine confirmation of receipt -> T05 Print and send confirmation of receipt -> T10 Determine necessity to stop indication -> T02 Check confirmation of receipt -> End

Figure 19. Knock-out relocation options for the Environmental Permit log

6.5.3 Discussion

From Table 6, we observe that the rule discovered for the knock-out check *T10* yielded a confidence of 0.983, which indicates that the rule may indeed be significant or useful. The other rules however, scored very low on confidence, and the tool generated appropriate warnings, as seen in the *Knockouts Analysis* output (Figure 17).

From the *Reordering options* output (Figure 18), we observe that certain data dependencies were found in the log, in particular one between *T02* and *T06*. However, the discovered dependencies were not enough to infer that either *T02* or *T06* needed to be executed before *T10* in a given process, so we provided the tool with a list of disallowed permutations as in [31].

Given the disallowed permutations, the tool correctly computes the most optimal reordering of the knock-out checks that is compatible with the constraints (Figure 18). If *T10* did not require either *T06* or *T02* to be performed before, it would actually be suggested as the first knock-out check to perform, given its very high rejection rate and consequently low effort-per-rejection value compared to the other checks.

Moreover, in the *Relocation options* output (Figure 19), we observe that the knock-out checks have been placed as early as possible in the process, respecting the data dependencies that the approach was able to detect. In principle there was nothing stopping the approach from suggesting, for instance, activity *T06* to be placed before *Confirmation of Receipt*, since the detected dependency indicated only that *T06* should go after the activity *Start*. However, from the process map of the Environmental Permit log (Figure 10) we observed that *Confirmation of Receipt* is an activity performed in all the cases, and always before the *knock-out* section of the process. Therefore, we indicated the tool to consider *Confirmation of Receipt* as the starting point of the process, and this is why we observe that the re-configurations only affect the activities after *Confirmation of Receipt*.

This showed that our approach for dependency detection between activities is limited by the availability and granularity of data.

Nonetheless, with this experiment we have verified that the approach supports a real-world event log as input (DO1), provides the decision rules of the knock-out checks of a process (DO2) and, once again, uses that information to compute improvement opportunities (D03), taking into account data dependencies (to the extent that they are reflected in the log).

7 Conclusion and Future Work

This thesis addressed the problem of overprocessing waste in business processes due to sub-optimal execution of knock-out checks. Specifically, it aimed at researching how to identify improvement opportunities for the execution of knock-out checks in a data-driven manner.

We observed that some of the current approaches and tools for identification of improvement opportunities related to knock-out checks take *as-is* process models instead of event logs as input, while others that work directly with event logs, place more emphasis on predictive model accuracy than interpretability, providing limited insights to the users into the knock-out checks of their processes.

To fill this gap, we proposed an approach that works directly with event logs and is based on interpretable machine learning techniques. It was shown through experiments on synthetic and real-world event logs that the approach successfully identifies improvement opportunities and provides insights into the knock-out checks of business processes. Moreover, the evidence did not show a significant loss of performance due to using a decision rule model, as opposed to the baseline which is based on *black-box* models.

We followed the Design Science Research Methodology [24] and formulated three Design Objectives: event log support (DO1), interpretability (DO2) and identification of improvement opportunities (DO3), and we proposed our approach and its implementation as the resulting artifacts. The fulfillment of the Design Objectives was verified through the experiments in Section 6.

We also showed a technique for detecting knock-out check dependencies directly from event logs that is based on analyzing the case attributes appearing in the discovered knock-out rules. While it was confirmed to work with a synthetic event log, the effectiveness of this technique on real-world logs is limited by the availability and granularity of event log data.

As future work, our approach could be extended by integrating more redesign patterns for computing additional improvement opportunities, such as eliminating or introducing knock-out checks. The approach could also be extended with a simulation phase for evaluating redesigns based on the obtained improvement opportunities. This could enable a *what-if* operation mode, in which the user could, for instance, interactively alter the decision rules of the checks, and then assess the impact through simulations. To preserve the data-driven paradigm of our approach, this would have to be done with state-of-the-art techniques for discovering simulation models directly from event logs such as the one presented in [2].

References

[1] BERTI, A., ZELST, S. J. V., AND AALST, W. M. P. V. D. Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science. *CoRR abs/1905.06169* (2019). arXiv: 1905.06169.

[2] CAMARGO, M., DUMAS, M., AND GONZÁLEZ-ROJAS, O. Automated discovery of business process simulation models from event logs. *Decision Support Systems 134* (July 2020), 113284.

[3] COHEN, W. W. Fast Effective Rule Induction. In *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 115–123.

[4] COHEN, W. W., AND SINGER, Y. A Simple, Fast, and Effictive Rule Learner. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA* (1999), J. Hendler and D. Subramanian, Eds., AAAI Press / The MIT Press, pp. 335–342.

[5] COTA, A. F. V. A BPM approach in a Small Azorean Company. Master's thesis, NOVA Information Management School, Jan. 2022.

[6] DU, M., LIU, N., AND HU, X. Techniques for interpretable machine learning. *Communications of the ACM 63*, 1 (Dec. 2019), 68–77.

[7] DUMAS, M., LA ROSA, M., MENDLING, J., AND REIJERS, H. A. *Fundamentals of Business Process Management*, 2nd ed. 2018 ed. Springer Berlin Heidelberg : Imprint: Springer, Berlin, Heidelberg, 2018.

[8] DUMAS, M., MILANI, F., NOLTE, A., LÓPEZ-PINTADO, O., AND CHAPELA-CAMPA, D. The Process Improvement Explorer (PIX): Automated Discovery and Assessment of Business Process Improvement Opportunities, 2019.

[9] FEHRER, T., FISCHER, D. A., LEEMANS, S. J., RÖGLINGER, M., AND WYNN, M. T. An assisted approach to business process redesign. *Decision Support Systems 156* (May 2022), 113749.

[10] FÜRNKRANZ, J., AND WIDMER, G. Incremental Reduced Error Pruning. In *Machine Learning Proceedings 1994*. Elsevier, 1994, pp. 70–77.

[11] GEERTS, G. L. A design science research methodology and its application to accounting information systems research. *International Journal of Accounting Information Systems 12*, 2 (June 2011), 142–151.

[12] HEVNER, MARCH, PARK, AND RAM. Design Science in Information Systems Research. *MIS Quarterly 28*, 1 (2004), 75.

[13] KUBRAK, K. Visualizing Business Process Improvements. Master's thesis, University of Tartu, Tartu, 2021.

[14] LASHKEVICH, K., MILANI, F., CHAPELA-CAMPA, D., AND DUMAS, M. Data-Driven Analysis of Batch Processing Inefficiencies in Business Processes. In *Research Challenges in Information Science*, R. Guizzardi, J. Ralyté, and X. Franch, Eds., vol. 446. Springer International Publishing, Cham, 2022, pp. 231–247. Series Title: Lecture Notes in Business Information Processing.

[15] LEE, S. A rule-based framework for interpretable predictions of business process outcomes using event logs. Master's thesis, Ulsan National Institute of Science and Technology, Ulsan, Dec. 2021.

[16] LOHRMANN, M., AND REICHERT, M. Effective application of process improvement patterns to business processes. *Software & Systems Modeling 15*, 2 (May 2016), 353–375.

[17] MEHDIYEV, N., AND FETTKE, P. Explainable Artificial Intelligence for Process Mining: A General Overview and Application of a Novel Local Explanation Approach for Predictive Process Monitoring. *CoRR abs/2009.02098* (2020). arXiv: 2009.02098.

[18] MOLNAR, C. *Interpretable machine learning: a guide for making Black Box Models interpretable*. Lulu, Morisville, North Carolina, 2019.

[19] MOSCOVITZ, I. How to Perform Explainable Machine Learning Classification — Without Any Trees, Mar. 2019.

[20] MURPHY, K. P. *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2012.

[21] NETJES, M., REIJERS, H. A., AND AALST, W. M. P. V. D. The PrICE Tool Kit: Tool Support for Process Improvement. In *Proceedings of the Business Process Management 2010 Demonstration Track, Hoboken, NJ, USA, September 14-16, 2010* (2010), M. L. Rosa, Ed., vol. 615 of *CEUR Workshop Proceedings*, CEUR-WS.org.

[22] NIEDERMANN, F., AND SCHWARZ, H. Deep Business Optimization: Making Business Process Optimization Theory Work in Practice. In *Enterprise, Business-Process and Information Systems Modeling*, T. Halpin, S. Nurcan, J. Krogstie, P. Soffer, E. Proper, R. Schmidt, and I. Bider, Eds., vol. 81. Springer Berlin

Heidelberg, Berlin, Heidelberg, 2011, pp. 88–102. Series Title: Lecture Notes in Business Information Processing.

[23] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[24] PEFFERS, K., TUUNANEN, T., ROTHENBERGER, M. A., AND CHATTERJEE, S. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems 24*, 3 (Dec. 2007), 45–77.

[25] PROKHORENKOVA, L. O., GUSEV, G., VOROBEV, A., DOROGUSH, A. V., AND GULIN, A. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada* (2018), S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., pp. 6639–6649.

[26] REIJERS, H., AND LIMANMANSAR, S. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega 33*, 4 (Aug. 2005), 283–306.

[27] SOUZA, A., AZEVEDO, L. G., AND SANTORO, F. M. Automating the identification of opportunities for business process improvement patterns application. *Int. J. Bus. Process. Integr. Manag. 8*, 4 (2017), 252–272.

[28] TEINEMAA, I. *Predictive and prescriptive monitoring of business process outcomes*. Doctoral Thesis, University of Tartu, Tartu, Estonia, Mar. 2019.

[29] VAN DER AALST, W. Re-engineering knock-out processes. *Decision Support Systems 30*, 4 (Mar. 2001), 451–468.

[30] VAN DER AALST, W., ADRIANSYAH, A., DE MEDEIROS, A. K. A., ARCIERI, F., BAIER, T., BLICKLE, T., BOSE, J. C., VAN DEN BRAND, P., BRANDTJEN, R., BUIJS, J., BURATTIN, A., CARMONA, J., CASTELLANOS, M., CLAES, J., COOK, J., COSTANTINI, N., CURBERA, F., DAMIANI, E., DE LEONI, M., DELIAS, P., VAN DONGEN, B. F., DUMAS, M., DUSTDAR, S., FAHLAND, D., FERREIRA, D. R., GAALOUL, W., VAN GEFFEN, F., GOEL, S., GÜNTHER, C., GUZZO, A., HARMON, P., TER HOFSTEDE, A., HOOGLAND, J., INGVALDSEN, J. E., KATO, K., KUHN, R., KUMAR, A., LA ROSA, M., MAGGI, F., MALERBA, D., MANS, R. S., MANUEL, A., MCCREESH, M., MELLO, P., MENDLING,

J., MONTALI, M., MOTAHARI-NEZHAD, H. R., ZUR MUEHLEN, M., MUNOZ-GAMA, J., PONTIERI, L., RIBEIRO, J., ROZINAT, A., SEGUEL PÉREZ, H., SEGUEL PÉREZ, R., SEPÚLVEDA, M., SINUR, J., SOFFER, P., SONG, M., SPERDUTI, A., STILO, G., STOEL, C., SWENSON, K., TALAMO, M., TAN, W., TURNER, C., VANTHIENEN, J., VARVARESSOS, G., VERBEEK, E., VERDONK, M., VIGO, R., WANG, J., WEBER, B., WEIDLICH, M., WEIJTERS, T., WEN, L., WESTERGAARD, M., AND WYNN, M. Process Mining Manifesto. In *Business Process Management Workshops*, F. Daniel, K. Barkaoui, and S. Dustdar, Eds., vol. 99. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 169–194. Series Title: Lecture Notes in Business Information Processing.

[31] VERENICH, I., DUMAS, M., LA ROSA, M., MAGGI, F. M., AND DI FRANCESCO-MARINO, C. Minimizing Overprocessing Waste in Business Processes via Predictive Activity Ordering. In *Advanced Information Systems Engineering*, S. Nurcan, P. Soffer, M. Bajec, and J. Eder, Eds., vol. 9694. Springer International Publishing, Cham, 2016, pp. 186–202. Series Title: Lecture Notes in Computer Science.

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Lino Moises Mediavilla Ponce**,
(author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Discovery of Improvement Opportunities in knock-out checks of Business Processes,
   (title of thesis)

   supervised by Fredrik Milani, Manuel Camargo and Katsiaryna Lashkevich.
   (supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Lino Moises Mediavilla Ponce
05/08/2022