Understanding Business Process Dynamics through System-Level Process Mining

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Alexander Kraus

Mannheim, 2026

Dekan: Prof. Dr. Claus Hertling, Universität Mannheim
Referent: Prof. Dr. Han van der Aa, Universität Wien
Koreferent: Prof. Dr. Jan Mendling, Humboldt-Universität zu Berlin

Tag der mündlichen Prüfung: 20. November 2025

Abstract

Process mining is a discipline that studies how organizational processes are carried out. By analyzing process instances in event logs recorded by information systems during process execution, process mining techniques allow organizations to understand how processes are actually performed, identify inefficiencies, and improve performance. Most existing techniques analyze process instances and their events in isolation, treating the process as static and unchanging over time. However, business processes are dynamic systems whose high-level behavior evolves in response to changing operational environments. This high-level process dynamics is often ignored. As a result, many process mining problems cannot be properly addressed without considering it, while others can benefit greatly from its inclusion.

The goal of this thesis is therefore to demonstrate how business process dynamics can advance process mining through system-level process mining. By focusing on high-level process characteristics that emerge from the execution of many concurrent instances, the thesis makes four contributions: (1) a taxonomy and framework for comprehensive concept drift characterization, (2) a computer vision approach for concept drift detection, (3) a framework for steady-state detection in business processes, and (4) a data-driven approach for process resilience assessment. We demonstrate the effectiveness of the proposed solutions through quantitative evaluations on synthetic data and highlight the practical value of our solutions using real-world data. All approaches were developed and evaluated following established practices in algorithm-engineering research, ensuring sound contributions that advance the body of knowledge in the field of process mining.

Zusammenfassung

Process Mining ist eine Disziplin, die untersucht, wie organisatorische Prozesse tatsächlich ausgeführt werden. Durch die Analyse von Prozessinstanzen, die während der Nutzung von Informationssystemen aufgezeichnet in Ereignisdaten werden, ermöglichen Process-Mining-Techniken Organisationen ein besseres Verständnis ihrer Abläufe, die Identifikation von Ineffizienzen sowie die Verbesserung der Leistungsfähigkeit. Die meisten bestehenden Techniken betrachten Prozessinstanzen und deren Ereignisse isoliert und gehen dabei implizit von einem statischen, unveränderten Prozess über die Zeit aus. Geschäftsprozesse sind jedoch dynamische Systeme, deren Verhalten sich auf Systemebene in Reaktion auf veränderte operative Umgebungen weiterentwickelt. Diese Prozessdynamik wird im Bereich des Process Mining häufig vernachlässigt. Folglich können viele Fragestellungen ohne ihre Berücksichtigung nicht zufriedenstellend gelöst werden, während andere erheblich von ihrer Einbeziehung profitieren.

Das Ziel dieser Dissertation ist es daher aufzuzeigen, wie die Dynamik von Geschäftsprozessen durch systemorientiertes Process Mining die Analyse von Prozessen erweitern und vertiefen kann. Mit Fokus auf Systemeigenschaften von Prozessen, die aus der Analyse vieler gleichzeitiger Prozessinstanzen entstehen, leistet die Arbeit vier zentrale Beiträge: (1) eine Taxonomie und ein Rahmenwerk zur umfassenden Charakterisierung von Concept Drifts, (2) einen Computer-Vision-Ansatz zur Erkennung von Concept Drifts, (3) ein Rahmenwerk zur Identifikation von stationären Zuständen in Geschäftsprozessen sowie (4) einen Ansatz zur Bewertung der Resilienz von Geschäftsprozessen. Wir zeigen die Wirksamkeit der vorgeschlagenen Lösungen anhand quantitativer Auswertungen mit synthetischen Daten und verdeutlichen ihren praktischen Nutzen anhand von Realweltdaten. Alle Ansätze wurden unter Anwendung etablierter Praktiken des Algorithm Engineering entwickelt und evaluiert, wodurch fundierte Beiträge entstehen, die den Wissensstand im Bereich des Process Mining erweitern.

Acknowledgements

Every achievement is made possible through the encouragement and support of many people, without whom it could not be realized. I therefore extend my sincere gratitude to all who contributed to the success of my doctoral journey.

First and foremost, I would like to express my deepest gratitude to my supervisor, Han van der Aa. Dear Han, I am grateful for your constant guidance throughout my doctoral journey and for the time you invested in my development. I have gone through the academic lows and highs of the doctoral journey to an extent I could not have imagined. However, under your guidance, I was able to endure, to grow, and to develop qualities that will continue to guide me successfully through life. And let us be honest, I am now a strong man with a strong CV ;).

I would like to express my sincere gratitude to Jana Rehse. Dear Jana, your valuable support and outstanding mentoring, especially in the moments when it mattered most, were crucial to the success of my PhD journey.

Throughout my doctoral journey, I had the privilege of meeting, collaborating, working, and exchanging ideas with many inspiring individuals. I would like to thank Adrian Rebmann for being an outstanding colleague and a true role model, whose achievements have continuously inspired and motivated me to grow. I am also grateful to Keyvan Amiri Elyasi for our productive collaboration. My sincere thanks go to my incredible colleagues: Robert Blümel, Michael Grohs, Marie-Christin Häge, Lukas Kirchdorfer, and Luka Abb for the valuable discussions and the many enjoyable moments we shared. I am also thankful to my colleagues from the DWS group and to my peers from other universities, with whom I had the opportunity to exchange ideas and build networks during conferences.

I would like to thank two outstanding friends who made my PhD journey truly special through meaningful conversations and many moments of joy: Christopher Dörr, with whom I spent a lot of time in discussions about our academic adventures, and Patrick Szymaniec, with whom I enjoyed valuable time outside academia, especially in sports and leisure activities.

Last, but most importantly, I wish to express my deepest gratitude to Alina for being the most magical person in my life.

Contents

Acronyms

BPM Business Process Management.

BPMN Business Process Model and Notation.

BPS Business Process Simulation.

CDC Concept Drift Characterization.

CV4CDD Computer Vision for Concept Drift Detection.

DES Discrete Event Simulation.

DFG Directly-Follows Graph.

ML Machine Learning.

PPI Process Performance Indicator.

PRA Process Resilience Assessment.

SSD Steady-State Detection.

VAR Vector Autoregression.

XES eXtensible Event Stream.

List of Algorithms

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we study business process dynamics through system-level process mining and demonstrate its usefulness for a variety of process mining tasks. *Business process dynamics* refers to the evolution of high-level process behavior over time. Instead of viewing a process as a static sequence of activities, process dynamics considers the evolution of the entire process as a system. This perspective helps to reveal and better understand many phenomena, such as dynamic bottlenecks, performance fluctuations, and process resilience. *System-level process mining* offers a way to study business process dynamics by analyzing the process as a whole system rather than focusing only on individual cases and their attributes. It captures high-level process characteristics that emerge from the interaction of many concurrent instances and evolve over time. By focusing on these characteristics, system-level process mining provides deeper insights into process dynamics, which can improve the accuracy of existing solutions and enable the handling of tasks that would otherwise remain unaddressed.

In this chapter, we provide an introduction to this doctoral thesis. Section 1.1 presents the motivation for studying system-level process mining. Section 1.2 gives an overview of the contributions of this thesis and the corresponding publications. Section 1.3 describes the research foundations of the conducted projects. Finally, Section 1.4 outlines the remaining chapters of this thesis.

1.1 Motivation

Business Process Management (BPM) is the art and science of overseeing how work is performed in an organization to ensure consistent outcomes and to take advantage of improvement opportunities [1]. The work performed in this context is typically represented by *business processes*, which refer to the core sets of activities that organizations execute to deliver products or services to their customers [1]. Business processes are widely supported by *information systems*, which record data generated during process execution. Such data, captured in the form of *event logs*, forms the basis for *process mining*, a family of techniques that analyze how business processes are truly executed [2]. Using event logs, process mining techniques equip organizations with the ability to gain a data-driven understanding of their operations, identify inefficiencies, and implement improvements.

Since the establishment of process mining as a discipline in the early 2000s, various techniques have been proposed to address different process mining tasks. Most of these techniques are based on *instance-level process mining*, which examines individual process instances and their case and event attributes, before aggregating insights from these instances into a process-level view. This approach has been useful for many well-established process mining tasks, such as process discovery and conformance checking. However, its main limitation is that process instances are analyzed in isolation, assuming that the underlying business process is static and unchanging over time. In reality, business processes are dynamic systems that evolve in response to changing operational environments. For various aspects of process analysis, particularly for tasks that consider the evolution of high-level process behavior over time (i.e., process dynamics), such a case-by-case approach to process mining does not suffice. As a result, many process mining problems, such as dynamic bottleneck detection [3] and high-level event mining [4], cannot be properly addressed without considering process dynamics, while others can benefit significantly from it.

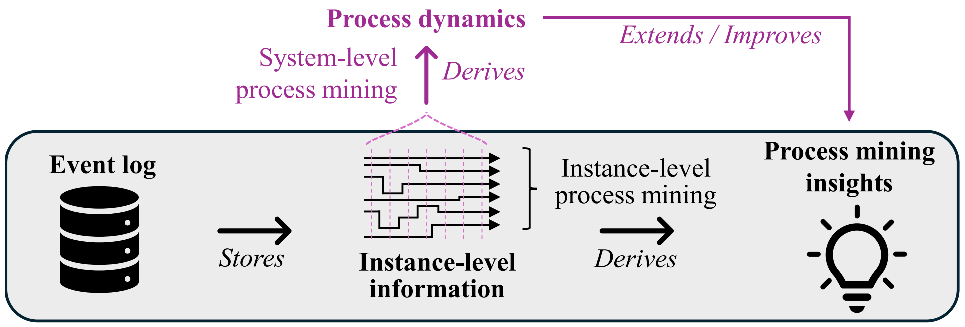


Figure 1.1: The idea of system-level process mining.

To address the limitations of instance-level process mining and to uncover the underlying process dynamics captured in event logs, this thesis considers *system-level process mining*. System-level process mining refers to the analysis that enables the study of process dynamics by capturing the evolution of high-level process behavior and its characteristics that emerge from aggregated data across multiple cases over time. As illustrated in Figure 1.1, this approach shifts the perspective from isolated process instances to the behavior of the business process as a whole system that evolves over time. Rather than assuming a static process, it captures high-level process characteristics that emerge from the interactions among process elements and reflect its dynamic behavior over time. By incorporating time-dependent system-level process characteristics such as the number of active process instances or the rate at which new cases arrive during a given period, system-level process mining has already shown clear benefits in process mining. For instance, it increases the accuracy of remaining time predictions for ongoing process instances [5], supports the automated development of more precise simulation models [6], and is essential for tasks such as detecting dynamic bottlenecks [3] or identifying high-level events [4]. However, significant research potential remains untapped.

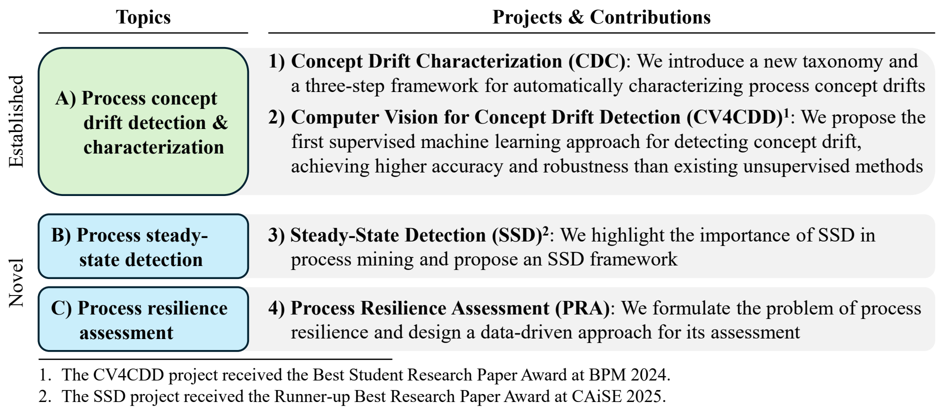
The objective of this thesis is to advance the field of process mining by showing how different forms of system-level process mining can be applied to study business process dynamics from multiple perspectives that help to address specific process mining tasks. Specifically, we show how system-level process mining improves the accuracy of *concept drift detection*, a well-established problem traditionally addressed using instance-level process mining. By examining the process as a whole over time, we enable the detection of subtle behavioral changes in process dynamics that can be used to detect drift patterns that instance-level methods cannot capture. We further illustrate how system-level process mining opens up previously unexplored tasks that require a system-level perspective. In particular, we focus on the detection of *steady states*, which identifies periods of overall stability in a process through the analysis of short- and long-term dynamics, and on the assessment of *business process resilience*, which examines how long-term process dynamics respond to temporary disruptions.

1.2 Contributions

This section presents the main contributions of the thesis. It outlines the addressed process mining topics and the related research projects with their key contributions. It concludes with a list of publications that form the foundation of this work.

1.2.1 Overview

As shown in Figure 1.2, the contributions of this thesis span three research directions in process mining and are realized through four projects. The first research direction focuses on the well-established task of detecting concept drifts in event logs. We study it in two projects: *concept drift characterization* and *concept drift detection using computer vision*. The other two directions focus on novel topics that are examined in separate projects: *steady-state detection* of a business process and business *process resilience assessment*. Notably, two of our projects received the Best Student Research Paper Award at the International Conference on Business Process Management (BPM 2024) and the Runner-up Best Research Paper Award at the International Conference on Advanced Information Systems Engineering (CAiSE 2025), as indicated in Figure 1.2.



Topics | **Projects & Contributions**

Established

A) Process concept drift detection & characterization

1) Concept Drift Characterization (CDC): We introduce a new taxonomy and a three-step framework for automatically characterizing process concept drifts

2) Computer Vision for Concept Drift Detection (CV4CDD)[1]: We propose the first supervised machine learning approach for detecting concept drift, achieving higher accuracy and robustness than existing unsupervised methods

Novel

B) Process steady-state detection

3) Steady-State Detection (SSD)[2]: We highlight the importance of SSD in process mining and propose an SSD framework

C) Process resilience assessment

4) Process Resilience Assessment (PRA): We formulate the problem of process resilience and design a data-driven approach for its assessment

1. The CV4CDD project received the Best Student Research Paper Award at BPM 2024.
2. The SSD project received the Runner-up Best Research Paper Award at CAiSE 2025.

Figure 1.2: Overview of research topics, projects, and key contributions.

The conducted projects share similarities. Specifically, across all projects, we developed automated, data-driven techniques that take event logs as input and apply system-level process mining to uncover relevant process dynamics, which are then used to address the research problem. In the following, we present each project by motivating the addressed problem and outlining its key contributions:

1. *Concept Drift Characterization (CDC)*: Business processes are subject to changes due to the dynamic environments in which they are executed. These process changes can lead to concept drifts, which are situations when the characteristics of a business process undergo significant changes. This results in event logs that contain data from different versions of a process. The accuracy and usefulness of process mining results derived from such event logs may be compromised because they rely on historical data that no longer reflects the current process behavior, or because the results do not distinguish between different process versions. Therefore, concept drift detection in process mining aims to identify drifts recorded in an event log by detect-

ing when they occurred, localizing process modifications, and characterizing how they manifest over time.

This project focuses on the latter task, i.e., drift characterization, which seeks to understand whether changes unfolded suddenly or gradually and if they form complex patterns, such as incremental or recurring drifts. However, current solutions for automatically detecting concept drifts from event logs lack comprehensive characterization capabilities. Instead, they mainly focus on drift detection and characterization of isolated process changes. This leads to an incomplete understanding of more complex concept drifts, like incremental and recurring drifts, when several process changes are interconnected. As a result, the actual evolution of the process remains hidden. To address these limitations, we introduce an improved taxonomy for characterizing concept drifts and a three-step framework that automatically identifies them from event logs. The framework applies two algorithms that leverage process dynamics to relate process changes. We evaluate our framework through extensive experiments conducted using a large collection of synthetic event logs. The results highlight the effectiveness of our proposed framework and show that it outperforms state-of-the-art techniques.

2. *Computer Vision for Concept Drift Detection (CV4CDD)*: This project goes a step further compared to the previous one. In the CDC project, our contribution focused on characterizing concept drifts, whereas in the CV4CDD project, we propose a new approach that also detects when drifts occur. Although many techniques have been proposed for drift detection, they often suffer from reduced accuracy in the presence of noise, diverse drift types, and varying degrees of change severity. Moreover, they are limited in scope, primarily detecting sudden and gradual drifts while neglecting more complex types such as incremental and recurring drifts, or in usability, requiring manual visual inspection to identify complex drifts.

   To address these limitations, we present CV4CDD, a novel approach for automated concept drift detection that can identify sudden, gradual, incremental, and recurring drifts. Our approach follows an entirely different paradigm. Specifically, it employs a supervised Machine Learning (ML) model fine-tuned on a large collection of event logs with known concept drifts, enabling the model to learn how drifts manifest in event logs. The possibility of training such a model has recently emerged through the availability of a tool that generates event logs with known concept drifts. However, applying supervised ML remains challenging due to the complexities of event log encoding. To address this, we propose converting an event log into an image-based representation that captures the evolution of process dynamics, enabling the use of a state-of-the-art computer vision model to detect drifts. Our experiments show that our approach, compared to existing solutions, improves the accuracy and robustness to noise of drift detection while covering a broader range of drift types, highlighting the potential of this new paradigm.

3. *Steady-State Detection (SSD)*: Detection of steady states is a critical task in the analysis of dynamic systems, as it enables the reliable evaluation of system behavior by differentiating between stable and unstable states. While SSD techniques have been developed and tested in domains such as signal processing and industrial systems, their application in the information systems domain, particularly in process mining, has been largely overlooked. Specifically, event logs that record the executed behavior of a business process often contain data from both steady and non-steady states. This can distort process mining results when conducting, e.g., performance analysis and remaining time prediction.

   In this project, we demonstrate the importance of SSD in process mining and introduce a two-step framework for the detection of steady states in business processes. The framework extracts relevant system-level process characteristics from an event log and applies established SSD techniques to identify periods in which a business process operated in a steady state. We evaluate the framework through experiments that assess its accuracy within a controlled environment using simulated event logs and that demonstrate the benefits of SSD for a downstream process mining task, namely remaining time prediction. The findings emphasize the importance and potential of SSD for obtaining more accurate process mining insights.

4. *Process Resilience Assessment (PRA)*: Process resilience represents a core competence for organizations in light of an increasing number of process disruptions, such as sudden increases in case arrivals or absences in the workforce. It reflects an organization's ability to restore a process to its acceptable performance level after a disruption. In this regard, the first key step for organizations towards achieving resilience is to understand how resilient their processes actually are. Although recognized as important, few works focus on such resilience assessment in a data-driven manner, thus limiting the ability of organizations to gain the necessary insights into how much their processes are affected by disruptions and how long it takes them to recover.

   To address this problem, we propose an approach for automated resilience assessment, based on recorded event data. Our approach interprets relevant system-level process characteristics, such as the average lead time or arrival rate, as time series that capture the development of the process execution over time. Based on these time series, it uses statistical modeling, specifically a vector autoregressive model, to determine the interrelations between those characteristics and assesses how the process performance responds to a disruption, i.e., a significant and temporary change in one of the process characteristics. We validate our approach by comparing its accuracy with that of a "what-if" analysis using a simulation model and demonstrate its effectiveness by assessing the resilience of the same process to diverse disruptions across different organizations.

1.2.2 Publications

The research conducted in this doctoral thesis has resulted in five published papers:

A. Process concept drift detection & characterization:

- [7] **Alexander Kraus**, Han van der Aa: Looking for Change: A Computer Vision Approach for Concept Drift Detection in Process Mining. *In: International Conference on Business Process Management (BPM), pp. 273-290, (2024).*

- [8] **Alexander Kraus**, Han van der Aa: Machine Learning-based Detection of Concept Drifts in Business Processes. *Process Science, 2,5 (2025).*

- [9] **Alexander Kraus**, Han van der Aa: Comprehensive Characterization of Concept Drifts in Process Mining. *Information Systems, 135, 102584 (2026).*

  Contribution: In all three publications, the author of this doctoral thesis was responsible for the conceptualization, implementation, initial draft writing, editing, and reviewing. Han van der Aa contributed as a supervisor by providing guidance during the research, as well as reviewing and editing the work.

B. Process steady-state detection:

- [10] **Alexander Kraus**, Keyvan Amiri Elyasi, Han van der Aa: On the Use of Steady-State Detection for Process Mining: Achieving More Accurate Insights. *In: International Conference on Advanced Information Systems Engineering (CAiSE), pp. 204-220 (2025).*

  Contribution: The author of this doctoral thesis was responsible for the conceptualization, implementation, drafting, editing, and reviewing. Keyvan Amiri Elyasi contributed by conducting the runtime prediction experiment, implementing the corresponding setup, and preparing the evaluation results. Han van der Aa contributed as a supervisor by providing guidance during the research, as well as reviewing and editing the work.

C. Process resilience assessment:

- [11] **Alexander Kraus**, Jana-Rebecca Rehse, Han van der Aa: Data-driven Assessment of Business Process Resilience. *Process Science, 1,4 (2024).*

  Contribution: The author of this doctoral thesis was responsible for the conceptualization, implementation, drafting, editing, and reviewing. Jana-Rebecca Rehse and Han van der Aa both contributed through guidance, reviewing, and editing.

Beyond the scope of this thesis, the author contributed to a demonstration paper presenting a tool for automatically generating event logs with known concept drifts, developed to support the CV4CDD project:

- [12] Justus Grimm, **Alexander Kraus**, Han van der Aa: CDLG: A Tool for the Generation of Event Logs with Concept Drifts. *In: Demonstration Proceedings of the International Conference on Business Process Management (BPM), CEUR, Vol. 3216, pp. 92-96 (2022).*

1.3 Research Methodology

This chapter presents the research methodology that underpins the work conducted in this thesis. The thesis proposes several projects that present data-driven algorithms aimed at addressing practical problems in the field of process mining. As such, the research aligns with the domain of *algorithm engineering*, which involves the design, implementation, and experimental evaluation of algorithms [13].

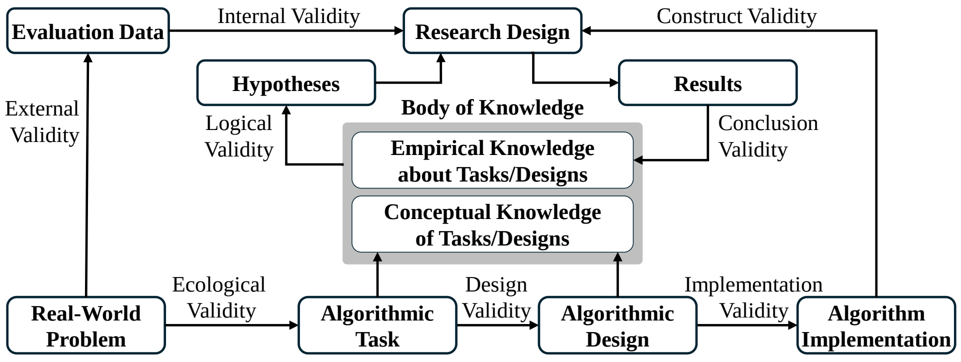


Figure 1.3: Methodology of algorithm engineering (adopted from [13]).

Figure 1.3 presents the overall research methodology and its key components. In the following, we examine three aspects of this methodology: the *ontological perspective*, the *epistemological foundations*, and the *methodological validity*.

1.3.1 Ontological Perspective

The ontology of algorithm engineering refers to the practice of identifying and implementing an algorithm to address a specific real-world problem [13]. From an ontological perspective, four key elements can be distinguished: the real-world problem, the algorithmic task, the algorithmic design, and the algorithmic implementation [13]. A scientific contribution demonstrates *ontological clarity* if it is explicitly aligned with these elements [13].

Table 1.1: Ontological clarity in the conducted research projects.

Project	**Description**
	Real-world problem
CDC	How to comprehensively characterize concept drifts in business processes?
CV4CDD	How to enhance automated concept drift detection in process mining?
PRA	How to assess business process resilience in a data-driven manner?
SSD	How to identify steady states of a business process?
	Algorithmic task
CDC	Input: event log, output: simple and complex drift types
CV4CDD	Input: event log, output: drift moment and type
PRA	Input: event log, output: four resilience measures
SSD	Input: event log, output: steady and non-steady state periods (sublogs)
	Algorithmic design
CDC	Framework: change detection, classification, interrelation analysis
CV4CDD	Approach based on visual encoding and computer vision
PRA	Approach based on statistical modeling and impulse-response analysis
SSD	Framework: process characteristic extraction, steady-state identification
	Algorithmic implementation
CDC	Python-based open-source implementation
CV4CDD	Python-based open-source implementation
PRA	Python-based open-source implementation
SSD	Python-based open-source implementation

Table 1.1 relates the four ontological elements to the conducted projects. In the following, each element is briefly described, and examples are provided to illustrate how the presented research achieves ontological clarity.

Real-world problem. Algorithm engineering is driven by real-world problems that emerge within specific contexts, indicating the need for an algorithm as part of a potential solution [13].

In this thesis, all proposed approaches address concrete real-world problems, such as how to detect process concept drift, assessing business process resilience, or identifying process steady states, as indicated in Table 1.1.

Algorithmic task. An algorithmic task represents an abstraction of a specific real-world problem by capturing its essential structure through a well-defined formulation, such as a set of requirements, while intentionally omitting irrelevant or excessive complexity [13]. It defines what needs to be done and in what context, often through input/output specifications.

For each real-world problem examined in this thesis, we define one or more algorithmic tasks that collectively contribute to its solution. For example, in the PRA project, we define the algorithmic task as the transformation of information recorded in an event log into four specific resilience measures that characterize the resilience of a business process. In addressing the algorithmic task of the CV4CDD and CDC projects, we define the algorithmic task as identifying instances of drifts, where each drift is described by its type and the corresponding change points. Finally, in the SSD project, the algorithmic task is to detect periods when a business process operates in a steady state.

Algorithm design. Algorithm design defines how to produce the desired output that meets the requirements of an algorithmic task, and when properly developed, it directly corresponds to that task [13].

In this thesis, we propose distinct designs for specific algorithmic tasks in each project. For example, in the CV4CDD project, we introduce a two-step design. The first step implements a four-phase algorithm that processes an event log meeting certain assumptions. It applies a sequence of operations—windowing, behavior abstraction, and similarity analysis—grounded in relevant domain knowledge. The result is an image that visualises process evolution over time. In the second step, we apply object detection techniques to these images to identify instances of concept drift. This two-step design produces outputs that fully satisfy the requirements of the overall algorithmic task. In the SSD project, we propose a two-step framework that first extracts system-level process characteristics from an event log as time series, and then applies established SSD techniques to detect periods when a business process operated in a steady or non-steady state. In the PRA project, after extracting system-level process characteristics, we employ a statistical model to perform an impulse–response analysis, which directly produces the resilience measures of interest. Finally, in the CDC project, our design follows a three-step framework, where in steps two and three we propose new algorithms to accomplish the task of detecting inter-related process changes.

Algorithm implementation. Algorithm implementation is a specific realization of the algorithm design [13]. It describes a concrete solution that is based on certain implementation decisions (e.g., selecting a programming language, hardware, etc.) to effectively address the original real-world problem when executed with given input data on a computer [13].

As shown in Table 1.1, all algorithm implementations in this thesis are developed in Python and released as open-source. These implementations are used to gain insights into the designs through evaluation experiments, where performance measures are applied to assess their effectiveness in addressing real-world problems.

1.3.2 Epistemological Foundations

The epistemology of algorithm engineering concerns the question of what can be known about an algorithm [13]. A scientific contribution demonstrates *epistemological precision* when it clearly identifies what is not known, explains how this gap in knowledge constitutes a research problem, and shows how the research extends the existing body of knowledge [13].

As summarized in Table 1.2, the projects presented in this thesis contribute to both *conceptual knowledge* of and *empirical knowledge* about algorithmic tasks and designs. In the following, we examine these types of knowledge in more detail and provide examples that demonstrate epistemological precision by highlighting the contributions made within each knowledge type.

Contribution to conceptual knowledge. Our contribution to conceptual knowledge is twofold: it advances the understanding of algorithmic tasks (knowledge of tasks) and the understanding of algorithm designs (knowledge of designs).

Conceptual knowledge of tasks. Conceptual knowledge of tasks refers to the understanding developed by abstracting real-world problems [13]. Due to their complexity and unpredictability, algorithmic tasks cannot be directly derived from these problems but must be constructed through a process of abstraction. Therefore, knowledge of tasks includes both the identification of relevant tasks and the understanding of their structure. This understanding can be described using mathematical, formal, semi-formal, or informal approaches [13].

In our research projects, we contribute in different ways to the conceptual knowledge of tasks, as shown in Table 1.2. For instance, in the PRA project, we propose a conceptualization of the problem that allows the task to be addressed algorithmically through the derivation of quantitative measures that assess the resilience level of a business process. In the CDC project, we propose a new taxonomy for classifying concept drift. This taxonomy enables a more comprehensive approach to concept drift detection compared to existing classification schemes. The SSD project formalizes the concept of a steady state in a business process. However, our CV4CDD project does not provide conceptual knowledge about tasks. Instead, it contributes to other types of knowledge.

Table 1.2: Epistemological precision in the conducted research projects.

Project	Description
	Conceptual knowledge of tasks
CDC	New concept drift characterization taxonomy
CV4CDD	Not applicable
PRA	Concept of process resilience and the resulting four resilience measures
SSD	Concept of the steady state of a business process
	Conceptual knowledge of designs
CDC	Three-step framework and two algorithms
CV4CDD	Log visualization technique that enables the use of computer vision
PRA	Approach that uses a statistical model to assess resilience
SSD	Two-step framework that uses system-level process characteristics
	Empirical knowledge about tasks
CDC	Not applicable
CV4CDD	Not applicable
PRA	Extraction of system-level process characteristics
SSD	Extraction of system-level process characteristics
	Empirical knowledge about designs
CDC	Comparison with the baselines, sensitivity analysis
CV4CDD	Comparison with the baselines, sensitivity analysis
PRA	Evaluation of approach effectiveness and usefulness
SSD	Evaluation of approach effectiveness and usefulness

Conceptual knowledge of designs. Conceptual knowledge of design refers to what an algorithm design is, how it functions, and how it addresses an algorithmic task [13]. This knowledge includes established designs that have proven effective based on formal or empirical evidence. These designs range from task-specific solutions to general procedures applicable to various tasks and often rely on generic design principles, such as divide-and-conquer [13]. This knowledge can be represented in various forms, depending on its proximity to implementation, including diagrams, pseudocode, flowcharts, or mathematical formulas [13].

The approaches presented in this thesis contribute to the knowledge of design by introducing novel designs in the form of an approach or framework that fulfill algorithmic tasks derived from real-world problems. These designs employ multiple representations to communicate the knowledge effectively. For example, when proposing a novel taxonomy for concept drift characterization, we use a UML Class Diagram to illustrate relationships between different elements related to concept drift and process changes. A flowchart represents the overall framework and its steps, while pseudocode is provided for two algorithms. Similarly, in the

CV4CDD project, we propose a new design for detecting concept drifts recorded in event logs using supervised ML, in contrast to existing state-of-the-art techniques that are unsupervised. The approach consists of two steps, which are represented in a chevron diagram. The PRA project introduces a design that applies a statistical model and its features to enable a data-driven assessment of business process resilience. Finally, the SSD project presents a novel two-step framework for studying steady states in business processes.

Contribution to empirical knowledge. Our contribution to empirical knowledge is also twofold: we advance the understanding of the properties of the addressed algorithmic tasks (knowledge about tasks) and the proposed solution designs (knowledge about designs) through controlled experiments in our evaluations.

Empirical knowledge about tasks. Empirical knowledge about tasks refers to insights into the characteristics of specific algorithmic problems [13]. A central aspect of empirical knowledge about tasks is the identification of recurring patterns in input data that inform the design of effective algorithms.

In our work, we contribute to empirical knowledge about tasks by identifying patterns in typical input data across two projects. Specifically, in the PRA and SSD projects, we examine system-level process characteristics using information recorded in event logs. These analyses demonstrate how the identified problems can be addressed using only limited data obtained from the underlying processes. In the other two projects, this contribution is not applicable.

Empirical knowledge about designs. Empirical knowledge about designs refers to insights into the properties and characteristics of a given algorithmic solution [13]. It includes the analysis of algorithm behavior, performance, and trade-offs, and often considers the relationship between algorithmic tasks, the corresponding designs, and how effectively those designs address the intended problems [13].

Through experimental evaluations of our design implementations, we contribute to both performance knowledge and sensitivity knowledge in the CDC and CV4CDD projects. We evaluate the performance of our designs by applying them to real-world event log collections and measuring their effectiveness using established task-specific performance metrics. Furthermore, we compare our results with baseline methods and state-of-the-art techniques. For example, in evaluating our concept drift detection approach, we employ widely used metrics such as precision, recall, and F1-score, and compare the outcomes with those achieved by existing state-of-the-art methods. To build sensitivity knowledge, we examine the impact of varying parameter settings, such as the number of windows, on accuracy. In the remaining two projects, where real baselines are not available, we conduct accuracy and validity analyses using synthetic data. We also evaluate the usefulness and effectiveness of the proposed designs on real-world data.

Table 1.3: Methodological validity in the conducted research projects.

Project	Description
	Ecological validity
CDC CV4CDD	Tasks aligned with an established problem in process mining
PRA SSD	Tasks aligned with a problem through analogies with other domains
	Design validity
CDC CV4CDD PRA SSD	We provided a detailed description of all relevant design steps, including explanations for key design decisions
	Implementation validity
CDC CV4CDD PRA SSD	The code was thoroughly tested during implementation and made reproducible through the public release of code, datasets, and results
	Internal validity
CDC CV4CDD PRA SSD	Controlled execution environment (Python, same server); randomized data split (if applicable), multiple runs on synthetic and real-world datasets
	External validity
CDC CV4CDD PRA SSD	Evaluation on diverse public datasets and synthetic data, including generated synthetic datasets when needed
	Construct validity
CDC CV4CDD	Use synthetic data with known gold standard and established metrics (precision, recall, F1-score); compare results with existing baselines
PRA SSD	Use synthetic data with known gold standard and established metrics (ϕ-coefficient, accuracy, mean absolute deviation)
	Logical validity
CDC CV4CDD PRA SSD	Derive hypotheses based on prior empirical knowledge and logical deduction to overcome limitations of existing methods
	Conclusion validity
CDC CV4CDD PRA SSD	State evaluation goals, quantitative evaluation, sensitivity analysis; report limitations of the proposed solutions

1.3.3 Methodological Validity

This thesis addresses methodological validity threats that are particularly relevant to the contributions of this doctoral work, as outlined in Figure 1.3. The following sections discuss these threats and the measures taken to ensure the reliability and robustness of the research findings, which are briefly summarized in Table 1.3.

Ecological validity. Ecological validity examines the relationship between real-world problems and an algorithmic task, evaluating how well task specifications represent the real-world problem to ensure result generalizability [14].

In this thesis, ecological validity is addressed in several ways. In the PRA project, we draw comparisons to similar tasks investigated in other research fields. This ensures that the defined task characteristics, such as maximal deviation and recovery time, are consistent with the requirements of real-world problems of a similar nature. In the SSD project, we represent the business process as a system and apply stability principles from systems theory, which have been used in other domains to determine when a system has reached a steady state. In the CDC and CV4CDD projects, we focus on an algorithmic task that is already well-known and established in the process mining domain.

Design validity. Design validity refers to the extent to which the internal structure of an algorithm is consistent, transparent, and explainable [13]. It enables reproduction, critical assessment, and reuse in an incremental scientific process [13].

In this thesis, we ensure design validity by providing a detailed and systematic description of all relevant design steps for each proposed approach. This includes formal definitions of the addressed problem, a clear specification of input and output formats, and a step-by-step explanation of the algorithmic procedures. For example, in the CDC and CV4CDD projects, we describe the conceptual framework, individual processing stages, parameter settings, and decision rules, making it possible for other researchers to replicate or adapt the methods. In the PRA project, we present the statistical foundations of the approach, such as the use of a Vector Autoregression (VAR) model, and explain how resilience measures are derived from its outputs. In the SSD project, we provide the rationale for selecting system-level process characteristics, describe how these characteristics are computed, and outline the logic behind the steady-state detection mechanism.

Implementation validity. Implementation validity: the influence of implementation choices on outcomes and the risk of unfaithful implementation [13].

To ensure implementation validity, we performed thorough testing throughout the development process to detect and correct potential errors early. We further reduced the risk of unfaithful implementation by ensuring full reproducibility. This was achieved by publicly releasing the source code, evaluation datasets, evaluation scripts, gold standards, and raw experimental results.

Internal validity. Internal validity refers to the extent to which a research design accurately establishes a causal relationship between the manipulation and the observed effect, without interference from external factors [13]. It ensures that observed performance differences result from the proposed approach rather than unrelated variables. Threats may arise from uncontrolled experimental conditions, implementation differences, or biases in data selection.

We address internal validity by reducing the influence of confounding variables in different ways. First, all runs of our approaches and baselines are executed in a consistent environment to block environmental factors such as hardware performance or memory usage. Second, randomization is applied, for example, by splitting datasets into training and test sets in the CV4CDD project, where we fine-tune and test a computer vision model, to mitigate hidden biases. Finally, multiple evaluation runs are conducted to account for variance, and both known external synthetic datasets and our generated datasets are used to ensure that results are not dependent on a single data source or configuration.

External validity. External validity concerns the generalizability of the results [15]. It considers to what extent we can assume the results of an experiment to hold "beyond the sample or domain that the researcher observes" [16].

In practice, addressing this validity concern is challenging because the number of publicly available real-world datasets with the required properties is very limited. Even when such data exists, the corresponding gold standard is often unknown. Therefore, in the projects presented in this thesis, we aimed to achieve the best possible evaluation by assessing the proposed solutions using both publicly available synthetic and real-world datasets that are widely recognized in the process mining research community and encompass a diverse range of characteristics that cover various process domains. For example, in the SSD and PRA project, we use several real-world event logs to demonstrate the usefulness of our solutions. However, when existing datasets lacked sufficient complexity or variation, or when no gold standard was available to enable effective evaluation, we generated synthetic data tailored to the requirements of specific experiments, ensuring that it contained the properties relevant to the analysis. For example, in the CV4CDD project, we employed a synthetic dataset that included different drift types, change severities, and noise levels. In the SSD project, we simulated a business process to generate synthetic data with a known gold standard, since no real data was available.

Construct validity. Construct validity assesses whether an evaluation measure accurately reflects the intended property [17]. More specifically, the evaluation measure has to be a valid and reliable operationalization of the intended concept [18].

In our projects, we focus on assessing the effectiveness of the proposed solutions. To ensure construct validity, we employ well-established evaluation metrics that measure the accuracy of our approaches in relation to the tasks they are designed to address. For example, in the CDC and CV4CDD projects, we use recall, precision, and the F1-score, which is the harmonic mean of precision and recall. In this context, precision is defined as the proportion of detected process change points that correspond to actual change points, while recall is the proportion of actual change points correctly detected by our approach. The corresponding gold standard is obtained either from existing synthetic datasets or generated synthetically using an external tool. In the PRA and SSD projects, we generate synthetic data with a known gold standard to assess the accuracy of the proposed algorithms. In the PRA project, we compare the measures produced by our approach with those obtained from simulation to determine accuracy. In the SSD project, we compare the detected steady-state periods with the actual steady-state periods obtained through controlled data generation. This is framed as a binary classification problem, and we evaluate performance using accuracy and the ϕ-coefficient, following practices applied in other domains for steady-state detection.

Logical validity. Logical validity concerns the soundness and justification of the reasoning underlying a formulated hypothesis [13]. It explains why a hypothesis is considered valid, typically through deduction from prior knowledge and support from an underlying theoretical framework.

In our projects, the hypotheses are not explicitly stated but are implicitly embedded within the research. To ensure the logical validity of these hypotheses, we ground them in logical deduction supported by prior empirical findings from evaluation and performance comparison analyses. For example, in the CV4CDD project, we begin by observing that the performance of existing techniques is often suboptimal in practical scenarios. This is largely due to the fact that these methods are built on specific assumptions about how concept drifts appear in event logs, assumptions that may not consistently reflect real-world behavior. In response, we propose an alternative approach that employs a computer vision model trained to detect drift patterns directly from the data, thereby eliminating the need for predefined statistical or handcrafted rules. Similarly, in the CDC project, we justify the development of a new taxonomy for concept drift classification by identifying limitations in the existing taxonomy. In the SSD project, we demonstrate the impact on process mining outcomes when steady and non-steady states of a business process are not distinguished during downstream tasks. In the PRA project, we emphasize the importance of a data-driven, quantitative assessment of process resilience that complements existing qualitative approaches.

Conclusion validity. Conclusion validity concerns the extent to which conclusions drawn from an analysis are reasonable given the stated hypotheses [19]. While it places strong emphasis on statistical analysis, it also encompasses qualitative considerations [20]. Statistical conclusions are valid when the assumptions of the applied tests are met and the required significance levels are achieved. Complementary qualitative analyses, such as examining outliers or anomalous data points, can provide additional support for conclusions or help identify reasons for unexpected results [21]. Limitations or threats to validity are often reported when reservations about empirical conclusions exist [13].

In all our projects, we address conclusion validity by clearly defining evaluation goals, conducting quantitative experiments with established evaluation metrics, and complementing these with quantitative analyses such as sensitivity analysis. We also report limitations of the proposed designs and evaluations.

1.4 Thesis Outline

The remainder of this thesis is organized into six chapters. The core Chapters 3 to 6 each present a research project that addresses a specific problem:

- *Chapter 2: Background.* Introduces the key concepts in process mining and system-level process mining that form the foundation for the thesis.
- *Chapter 3: Comprehensive Concept Drift Characterization.* Proposes a new taxonomy and a three-step framework for the automatic characterization of concept drifts from event data. The chapter builds on concepts and results previously published in the Information Systems [9].
- *Chapter 4: Concept Drift Detection Using Supervised ML.* This chapter presents a novel ML-based approach for detecting different types of concept drift. The content builds on earlier work published at BPM [7] and its extended version in Process Science [8].
- *Chapter 5: Business Process Steady-State Detection.* Examines the importance of steady-state detection in process mining and presents a framework for identifying steady states from event data. The chapter builds on work described in a paper published at CAiSE [10].
- *Chapter 6: Business Process Resilience Assessment.* Introduces a data-driven method for assessing business process resilience using information recorded in event logs. The chapter is based on concepts and results previously published in the Process Science [11].
- *Chapter 7: Conclusion.* Concludes the thesis by summarizing key results, discussing implications, and outlining future research directions.

Chapter 2

Background

This chapter provides the necessary background for the remainder of the thesis. It consists of two sections. Section 2.1 introduces process mining and discusses key concepts referenced throughout this thesis. Section 2.2 discusses the idea of system-level process mining, which plays a pivotal role in the solutions proposed in this thesis.

2.1 Process Mining

This section provides an overview of process mining and its fundamental concepts. We begin with an introduction to process mining as a discipline. Then, Section 2.1.1 defines the concept of a business process, which forms a core element in process mining, followed by a discussion of event logs in Section 2.1.2, which serve as the primary data source for process mining tasks.

Process Mining. Process mining is an emerging discipline that focuses on the analysis and improvement of business processes by systematically leveraging event data recorded by information systems [22]. With the help of process mining, organizations can improve their processes by conducting a wide range of analytical activities [1, 2, 22], such as:

- discovering and visualizing process flows,
- revealing compliance violations and deviations from expected behavior,
- analyzing bottlenecks and identifying inefficiencies,
- monitoring Process Performance Indicators (PPIs) in real time,
- supporting root-cause analysis of performance problems,
- detecting process concept drifts,
- predicting process outcomes and estimating remaining execution time,
- evaluating resource utilization and workload distribution,

- simulating alternative process scenarios for decision support.

As a result, process mining has been widely adopted across a range of sectors, including manufacturing, logistics, finance, healthcare, education, and public administration [2]. It is important to note that process mining is not limited to business processes and can be applied to any type of process for which events are recorded [2].

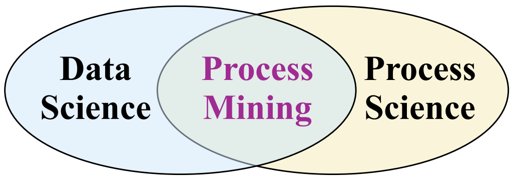


Figure 2.1: Process mining as a discipline (adapted from [22]).

As illustrated in Figure 2.1, process mining is situated at the intersection of *data science* and *process science*, serving as a critical bridge between these foundational fields [22]. Specifically, mainstream data science approaches such as data mining and machine learning are generally process-agnostic, meaning they do not take end-to-end process models into account [2]. In contrast, process science approaches are process-centric but often emphasize the creation of models rather than learning from event data. Process mining brings these perspectives together by combining the model-driven insights of process science with the data-driven techniques of data science [2].

To better understand process mining as a discipline, we briefly discuss the characteristics of *data science* and *process science*.

Data science. Data science is an interdisciplinary field aiming to turn data into value [2]. *Data* refers to raw observations or facts collected from various sources, such as sensors, user interactions, transactions, or system logs, which can be structured (e.g., databases), semi-structured (e.g., JSON files), or unstructured (e.g., text, images, videos). *Value*, in a business context, is a concept that is unique to each organization [23] and includes different tangible and intangible elements [24]. However, in general, it refers to the measurable benefits that an organization gains from its activities, and can take different forms: financial, operational, customer, reputational, etc. [23]. It may be provided in the form of predictions, automated decisions, models learned from data, or any type of data visualization delivering insights [2]. *Turning data into value* involves data extraction, preparation, exploration, transformation, storage and retrieval, computing infrastructure, data mining and machine learning, and the responsible application of results, with attention to ethical, social, legal, and business aspects [2].

Data science focuses on four main objectives: *reporting* (What happened?), *diagnosis* (Why did it happen?), *prediction* (What will happen?), and *recommendation* (What is the best that can happen?) [25]. To achieve these objectives and answer the corresponding questions, data science uses techniques from several established domains [2]:

- *Statistics*: Foundation of data science, divided into descriptive (summarizing data) and inferential (drawing conclusions from samples) methods.
- *Algorithms*: Essential for data analysis; efficiency becomes critical as data size grows. Examples include Apriori, MapReduce, and PageRank.
- *Data Mining*: Analyzes large data sets to uncover hidden patterns or relationships; focuses on practical applications and scalability.
- *Machine Learning*: Develops algorithms that learn from data and improve over time without explicit programming. Enables data-driven predictions and decisions.
- *Predictive Analytics*: Uses historical data to predict future outcomes; often applied in business contexts using mining and learning techniques.
- *Databases*: Store and manage data efficiently. Includes traditional relational databases and modern NoSQL or in-memory systems for large-scale, real-time access.
- *Distributed Systems*: Provide scalable infrastructure for data analysis by executing tasks across multiple computing nodes (e.g., cloud computing).
- *Visualization & Visual Analytics*: Supports human interpretation through interactive and automated visual tools, aiding insight discovery in complex data sets.
- *Behavioral & Social Sciences*: Help interpret data influenced by human behavior; essential for understanding societal and organizational dynamics.
- *Privacy, Security, Law & Ethics*: Ensure responsible data usage, safeguard sensitive information, and prevent biased or unethical analysis outcomes.

Process science. Process science is the interdisciplinary study of business processes as they unfold over time [26]. It can be considered as a post-disciplinary approach that puts processes (as the phenomenon of interest) in the center of attention and invites contributions from as many disciplines as can make a contribution to identifying, understanding and intervening into processes [27]. Process science focuses on understanding and analyzing sequences of business activities that involve both human actors and technological systems. It integrates conceptual process thinking with data-driven methods, extending data science by shifting the unit of analysis from isolated data points to structured, temporal behaviors, placing processes, rather than just data, at the center of investigation [26].

Process science focuses on three main objectives: discovery, explanation, and intervention in business processes using recorded event data [26]:

- *Discovery* emphasizes the detection of (emergent) dynamics that constitute the phenomenon of interest to uncover patterns that may only be understood retrospectively.
- *Explanation* aims to understand how and why processes unfold, identifying cause-effect relations within their temporal and spatial context, and leveraging theory as well as diverse data sources to gain an in-depth understanding. Prediction emerges from this understanding, enabling anticipation of future process states based on patterns and indicators.
- *Intervention* targets the purposeful change of processes, guided by prior explanations and envisioned goals, using established approaches such as design science research, policy development, or experimental design.

Throughout all objectives, process science relies on event data that capture temporal change and stresses the importance of integrating data across different levels of abstraction to understand complex process interplay.

To achieve the objectives, process science uses techniques from several established domains [2]:

- *Stochastics*: Uses probabilistic models (e.g., Markov chains) to analyze random behavior in processes such as waiting times and reliability.
- *Optimization*: Aims to find the best solution from many alternatives using techniques like linear programming, integer linear programming, constraint satisfaction, and dynamic programming.
- *Operations Management & Research*: Focuses on the design and control of processes and systems. Operations Research emphasizes mathematical modeling, while Operations Management deals with practical implementation in business contexts.
- *Business Process Management*: Involves modeling, execution, monitoring, and optimization of business processes, often using formal models such as Business Process Model and Notation (BPMN) and Petri nets.
- *Business Process Improvement*: Covers a range of methods (e.g., Lean Six Sigma) aimed at enhancing efficiency and effectiveness in business processes.
- *Process Automation & Workflow Management*: Focuses on model-driven systems that support and automate operational business processes using workflow engines.
- *Formal Methods & Concurrency Theory*: Uses rigorous mathematical tools to model, analyze, and verify concurrent and distributed systems.

2.1.1 Business Process

A central concept in process mining is the *business process* [22]. In the following sections, we present the key elements of a business process, its formal definition, the main perspectives for analysing business processes, and the concept of a business process model along with its versions in the context of process mining.

Business process elements. As visualized in Figure 2.2, a business process is characterized by the following elements [1]:

- *Events:* Instantaneous occurrences of specific process actions or changes.
- *Activities:* Units of work that take time to complete and may consist of smaller tasks or sub-activities.
- *Decision points:* Conditions or branching logic that determine how the process continues based on specific outcomes.
- *Actors:* Entities involved in process execution, including human participants, organizations, or software systems. Actors can be internal (e.g., employees) or external (e.g., business partners).
- *Objects:* Physical resources (e.g., equipment) and informational items (e.g., documents) that are used or generated during the process.
- *Outcomes:* Results produced by the process, which deliver value.
- *Customer:* These outcomes are consumed by one or more customers, who can be either internal or external to the organization.

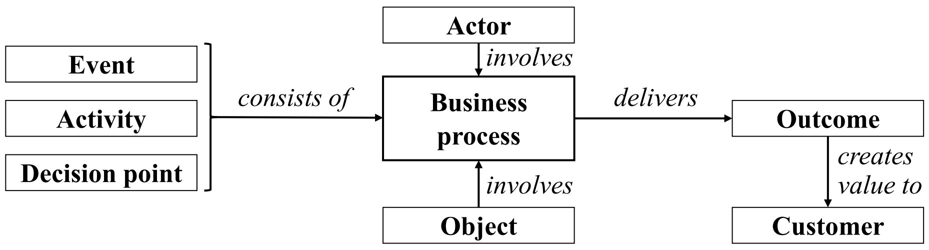


Figure 2.2: Elements of a business process (adapted from [1]).

We formalize the definition of a business process as follows [1, 28]:

Definition 1 (Business Process) *A business process is a collection of interrelated events, activities, decision points, actors, and objects that are performed in an organizational and technical environment and collectively lead to an outcome of value to at least one customer.*

This definition underscores the perspective that a business process functions as an interactive system of interconnected components, working together to drive purposeful organizational behavior and create value.

Business process perspectives. A business process can be analyzed from multiple perspectives, providing a multidimensional understanding of its structure and behavior [1]:

- *Control-flow perspective* focuses on the sequencing and dependencies of activities.
- *Resource perspective* examines the involvement of various resources, such as individuals, systems, or departments, based on event log data, aiming to reveal organizational structures or uncover social networks.
- *Time perspective* addresses the temporal aspects of process execution by leveraging timestamps to detect bottlenecks, assess service levels, monitor resource utilization, and predict the remaining duration of ongoing cases.
- *Data perspective* considers additional contextual information associated with events, such as costs, supplier names, product types, or quantities.

To capture and analyze different perspectives of a business process, business process models provide a structured way to visualize its key elements and flow.

Business process model. A business process model is a structured diagrammatic representation, typically focusing on the control-flow of a business process, and generally consists of three fundamental types of elements [1]:

- *Activity nodes:* Represent units of work carried out by humans, software systems, or a combination of both.
- *Control nodes:* Define the flow of execution between activities, determining the sequence and logic of the process.
- *Event nodes:* Capture occurrences that may trigger or affect the process, such as external messages or internal events that require a response. While not supported in all modeling notations, they are essential for modeling interactions with the process environment.

Other types of elements may be included, but these three serve as the core building blocks of most process models. We formalize the concept of a process model as follows [29]:

Definition 2 (Process Model) *A process model is a tuple $M = (A, E, G, N, F, t)$, where:*

- *A is a finite set of activities,*
- *E is a finite set of events,*
- *G is a finite set of gateways,*
- *$N = A \cup E \cup G$ is a finite set of nodes,*
- *$F \subseteq N \times N$ is the flow relation, such that (N, F) forms a directed graph,*
- *$t : G \to \{and, xor\}$ is a mapping that assigns a type to each gateway.*

Business processes are often represented using a specific process modeling language. There are many languages for modeling business processes diagrammatically [1], such as from Directly-Follows Graphs (DFGs), transition systems, process trees, and Petri nets, Event-driven Process Chains (EPCs), and UML activity diagrams [22].

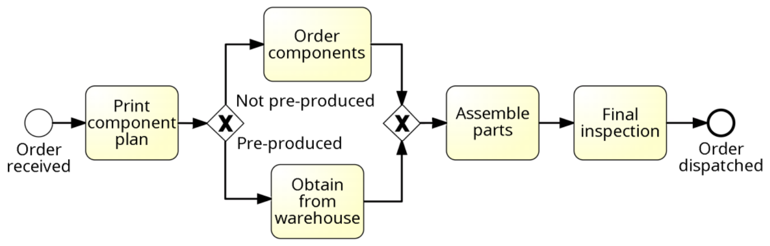


Figure 2.3: Example of a business process model.

In the thesis, we use BPMN [30], a widely recognized standard for business process modeling. Figure 2.3 presents a simple business process model created using BPMN. This model features a start event labeled “Order received”, five business process activities, and an end event marked “Order dispatched.” The activities are carried out in sequence, with a single decision point following the first process activity. The process model focuses on the control-flow perspective of the business process, however, it can be extended, for example, to also represent the resource perspective by adding pools and lanes.

Business process version. Business processes evolve in response to organizational changes, such as regulatory updates and technological advancements [31]. As a result, different versions of the same process may coexist or follow one another over time. Identifying and analyzing process versions is crucial for understanding process evolution, comparing performance across time periods or units, and ensuring compliance [31]. Given the definition of a business process, we define a business process version as:

Definition 3 (Business Process Version) *A business process version is a variant of a business process, defined by a specific combination of involved events, activities, decision points, actors, and objects.*

Different process versions can be represented through corresponding business process models. Figure 2.4 shows two alternative process models that describe different versions of the same process. Compared to the original process version in Figure 2.3, process version 1 presents a simple linear flow without decision points. Process version 2 modifies the original version by simplifying the decision

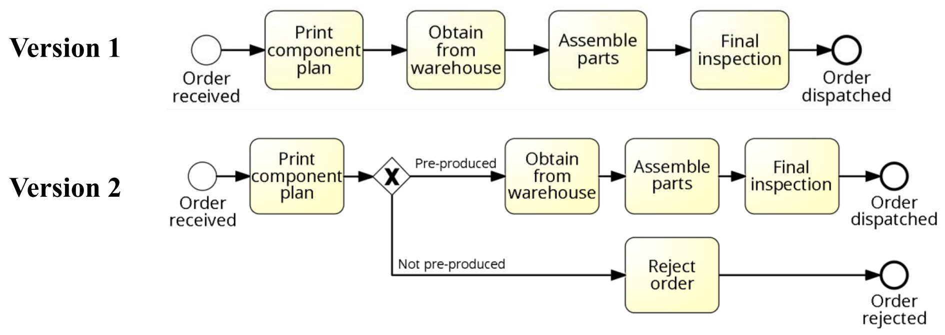


Figure 2.4: Example of a business process model version.

logic and introducing a new outcome: orders with unavailable components are rejected instead of being processed. In this thesis, we refer to different process versions as cases where the underlying process models differ. Two process model versions are considered different if any of their model elements are not identical.

2.1.2 Event Log

In addition to the concept of a business process, another important concept in process mining is the event log. An *event log* contains a collection of traces, where each trace represents the sequence of events recorded for a specific process instance. These events are captured by a process-aware information system during the execution of the process [2]. In the following, we present an event log example and discuss its sources, essential requirements, underlying data model, and the formalization used throughout this PhD thesis.

Event log example. Table 2.1 presents an illustrative snapshot of an event log generated from the execution of the business process shown in Figure 2.3. Each row in the table corresponds to a single event recorded during the execution of a business process. These events reflect the observable behavior of the process and serve as the foundation for discovering, analyzing, and improving business processes. In this example, every event is described by five attributes: *Case ID*, *Timestamp*, *Activity*, *Resource*, and *Cost*. The *Case ID* identifies the process instance (e.g., an order), the *Activity* denotes the operation performed, and the *Timestamp* indicates when it occurred. The additional attributes, such as *Resource* and *Cost*, provide contextual and operational insights that can support more advanced analyses, such as organizational or performance mining. In this illustrative event log, Case 1 and Case 2 represent two separate process instances following a similar sequence of activities with variations in timing and resource allocation.

Table 2.1: Example of an event log.

Case ID	**Timestamp**	**Activity**	**Resource**	**Cost**
1	29-12-2010 14:17	Order received	Susana	0
1	30-12-2010 11:02	Print component plan	Mike	50
1	31-12-2010 10:06	Obtain from warehouse	Sue	200
1	05-01-2011 15:12	Assemble parts	Mike	100
1	06-01-2011 11:18	Final inspection	Sara	200
1	07-01-2011 10:00	Order dispatched	Sara	80
2	30-12-2010 15:34	Order received	Susana	0
2	30-12-2010 11:32	Print component plan	Mike	50
2	30-12-2010 12:12	Order components	Mike	100
2	30-12-2010 14:16	Assemble parts	Pete	400
2	05-01-2011 11:22	Final inspection	Sara	200
2	06-01-2011 11:33	Order dispatched	Sara	80
...	...	...	...	...

Event log sources. Event logs are increasingly available across sectors such as finance, manufacturing, healthcare, education, and e-commerce, driven by the digitalization of business processes. They can be extracted from different sources [22]:

- *Business Process Management Systems:* Highly process-aware and typically provide structured, analysis-ready logs with little preprocessing.
- *Case management and ticketing systems:* Common in service contexts; log status changes and timestamps but may require preprocessing to extract activity labels.
- *Enterprise Resource Planning and Customer Relationship Management systems:* Widely used platforms like SAP and Salesforce offer rich event data but often involve complex extraction procedures.
- *Operational databases:* Support daily operations and may store historical data useful for process mining.
- *Project management tools:* Systems like Jira and Trello generate event data aligned with agile and project workflows.
- *Business intelligence infrastructure:* Data warehouses and data lakes consolidate business data, with lakes offering more flexible, schema-on-read access.
- *Web applications and platforms:* Generate extensive user interaction data, often in JSON format, valuable for customer-centric analysis.
- *Internet of Things:* Produce time-stamped sensor data, increasingly used in domains like manufacturing and healthcare despite structural challenges.

Event log requirements. An event log must meet three key data requirements to be suitable for process mining [22]:

- *Case ID:* Each event must be associated with a unique process instance, al-

lowing related events to be grouped into distinct process executions. Case IDs are essential but may not always be directly available, sometimes requiring event correlation techniques for reconstruction.
- *Activity label:* Each event should correspond to a specific activity within the process. These labels should ideally reflect a business-relevant level of granularity, which may require transforming low-level system events through event abstraction.
- *Event order:* Events must be ordered within each process instance, typically using timestamps. Alternatively, order can sometimes be inferred from the sequence of log entries, provided it reflects the actual execution order.

In addition to these core elements, event logs often include supplementary attributes such as vendor, location, cost, or item details. These may be case-level or event-specific and support filtering, provide contextual insights, and enable advanced analyses.

Event log format. The eXtensible Event Stream (XES) format, standardized by the IEEE in 2016 [32], is a widely adopted format for storing and exchanging event data in process mining. As shown in Figure 2.5, it structures logs as collections of traces, each representing a sequence of events for a specific case. XES supports flexible attribute definitions, with different variable types, and allows semantic enrichment through extensions. XES's flexibility and semantic capabilities make it a foundational standard for event log representation in process mining.

Other formats have also been developed for storing event data. One of the more recent formats is object-centric event log [33], which allows events to be linked to multiple objects of different types, rather than being restricted to a single case. However, in this thesis, we focus solely on the XES format and do not consider alternative event data formats.

Event log formalization. *Events* form *traces*, and traces form *event logs*. In the following, we formally define these concepts.

An event corresponds to the execution of a single action within a process. In the context of this thesis, we formally define events as follows:

Definition 4 (Event) *Let $\mathcal{E}$ denote the universe of all possible events. An event $e \in \mathcal{E}$ is defined as a tuple $e = (id, c, a, t, D)$ of five event attributes, where id is a unique event identifier (event ID), c is the identifier of a business case (case ID), a refers to an activity name, t refers to the timestamp, and D is a set of data attributes that represent additional context.*

Uniqueness of events, provided by the event ID, is essential in process mining because it prevents ambiguities in event ordering, allows precise referencing for analysis tasks such as performance monitoring or anomaly detection, and helps

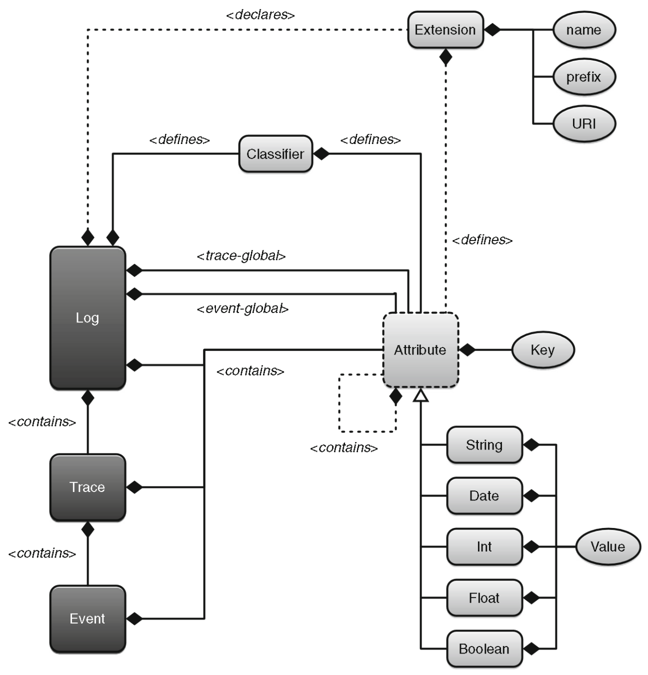


Figure 2.5: Meta model of XES format for event logs [32].

maintain data integrity during preprocessing or integration. If the event ID is missing, uniqueness can be ensured by enumerating all events in the event log after sorting them by their timestamps. In this thesis, we use dot notation as a shorthand to refer to event attributes, for example, $e.a$ for the activity of an event e.

Events sharing the same case ID are grouped into a *trace*, which represents a sequence of events ordered by their time of occurrence within a single process instance. If the timestamp-based ordering of events within a single case does not constitute a strict total order, such as when multiple events share the same timestamp, the event ID attribute can be used as a criterion to enforce a strict total order. This attribute can always be obtained by enumerating all recorded events in an event log, starting from 1 up to $|L|$, the total number of recorded events. The formal definition of such traces is provided below.

Definition 5 (Trace) *A trace $\sigma = \langle e_1, \ldots, e_n \rangle$ is a finite sequence of $n \in \mathbb{N}$ events from $\mathcal{E}$ such that all events share the same case ID and are strictly ordered by their timestamp and, when timestamps are equal, by event ID. A trace variant denotes traces with the same activity sequences.*

Note that, in addition to events having attributes, cases themselves may also include attributes that describe or influence the overall process instance [2]. However, case attributes are not considered within the scope of this thesis. In addition, we often represent traces by listing the sequence of activity names, which may be abbreviated for simplicity. For example, the trace with Case ID 1 in the event log shown in Table 2.1 can be represented as $\langle$Order received, Print component plan, Obtain from warehouse, Assemble parts, Final inspection, Order dispatched$\rangle$, or more concisely as $\langle \text{OR}, \text{PCP}, \text{OfW}, \text{AP}, \text{FI}, \text{OD} \rangle$, where activity names are abbreviated using their initial letters.

Finally, having defined events and traces, we can define an event log as a collection of traces:

Definition 6 (Event log) *An event log L is a finite set of traces. Σ_L denotes the ordered collection of traces, arranged according to the timestamp of their first event. For each index $i \in \mathcal{I} := \{1, 2, \ldots, |\Sigma_L|\}$, σ_i denotes the i-th trace in Σ_L.*

Figure 2.6 shows an example of an event log visualized using a dotted chart in ProM[1]. The chart presents an ordered collection of traces, arranged based on the timestamp of their first event. Each dot represents the execution time of a recorded event. Events that belong to the same trace are aligned horizontally and connected by a line. All approaches proposed in this thesis take as input an event log that can be represented as in this figure.

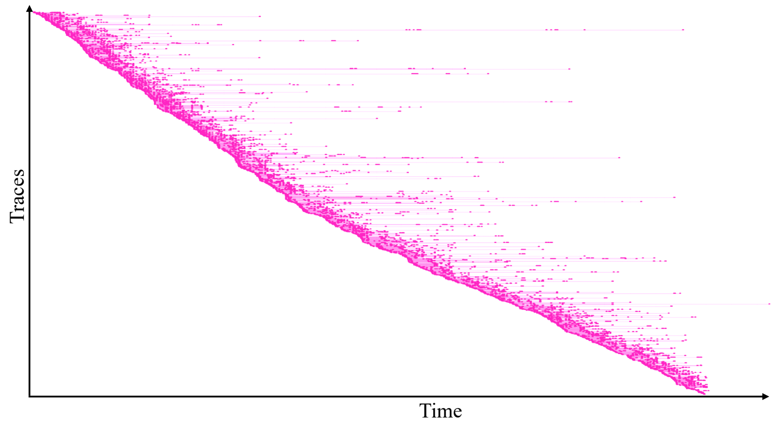


Figure 2.6: An example of an ordered collection of traces in an event log.

Behavioral representation. A *behavioral representation* of a process is a set of defined behavioral relations and their support (e.g., frequency) that characterize a process based on data from an event log L. A commonly used type of behavioral

[1]Process mining tool available online at `https://promtools.org/`

representation in process mining is the directly-follows relation [34], which captures the frequency with which two activities are observed to immediately succeed one another within the same case. For example, using the two cases shown in Table 2.1, we can derive the following directly-follows relations with abbreviated activity names: $(OR \rightarrow PCP)$: 2, $(PCP \rightarrow OfW)$: 1, $(OfW \rightarrow AP)$: 1, $(AP \rightarrow FI)$: 2, $(PCP \rightarrow OC)$: 1, $(OC \rightarrow AP)$: 1, $(FI \rightarrow OD)$: 2. Other behavioral representations, such as eventually-follows relations [34], α-relations [34], behavioral profiles [35], or declarative process constraints [36], are also frequently applied to capture different aspects of process behavior.

2.2 System-Level Process Mining

In this section, we discuss *system-level process mining*, an analysis that enables the data-driven study of *business process dynamics*, i.e., the evolution of high-level process behavior over time. In recent years, it has shown strong potential for addressing both well-established and emerging tasks in process mining, particularly in situations when problems cannot be addressed through case-level analysis, in which cases are examined in isolation. Although the underlying idea is not new and has been implicitly applied in many process mining techniques, it has only recently gained increased attention within the process mining community.

Next, we examine three key aspects that are central to system-level process mining. First, in Section 2.2.1, we discuss why system-level process mining is applicable and relevant for business processes. Then, in Section 2.2.2, we describe the typical system-level process characteristics and how they are derived from an event log. Finally, we demonstrate the practical relevance of system-level process mining by highlighting several prominent process mining tasks where this approach has already been applied in Section 2.2.3.

2.2.1 Business Process as a Complex System

In system theory, a *system* is defined as a set of interrelated components or elements that work together in an organized way to achieve a common goal or purpose [37]. Key characteristics of a system include [37–39]:

- *Elements*: The individual parts that make up the system.
- *Interrelationships*: The interactions between the elements.
- *Boundary*: A perimeter that distinguishes the system from its environment.
- *Environment*: Everything outside the system that interacts with it.
- *Inputs and Outputs*: Systems receive inputs from the environment, process them, and produce outputs.

- *Purpose*: Every system is organized to fulfill a specific function or achieve an objective.

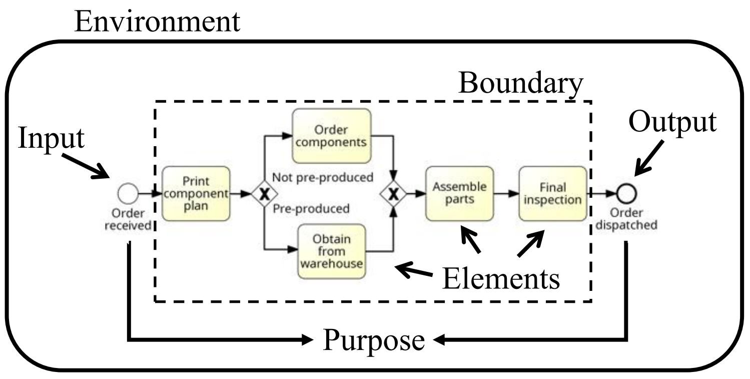


Figure 2.7: Business process as a system.

Given our definition of a business process (see Definition 1), we can consider a business process as a specific type of system, as it exhibits all the aforementioned characteristics of a system. As illustrated in Figure 2.7, the input is represented by incoming orders, while the output consists of dispatched orders. By dispatching orders, the process fulfills its purpose, i.e., meeting customer demand. The elements of the process include its key business activities, the control flow between them, and other process perspectives such as the resources involved. These interconnected elements define the process boundary, which operates within a business environment influenced by various external factors, such as economic conditions that may affect the volume of incoming orders.

Business processes are systems that exhibit dynamic complexity [40], as they are characterized by:

- constant change across multiple time scales,
- tight coupling and interdependence among activities and resources,
- feedback mechanisms that shape outcomes,
- nonlinear relationships between causes and effects,
- history dependence and path constraints,
- self-organizing patterns emerging from interactions,
- adaptation through learning and evolution, and
- trade-offs between short-term and long-term effects.

To study and understand these aspects, it is necessary to consider *process dynamics*, which we define as [41]:

Definition 7 (Process Dynamics) *Process dynamics refers to the evolution of* *high-level process behavior over time that characterizes how the process as a whole responds to internal and external variations.*

The study of process dynamics in process mining within its socio-technical environment [26] is particularly challenging, in contrast to engineered systems, where components are physical and interactions typically follow clear physical laws or rule-based patterns. In business processes, by contrast, interactions among elements leave behind digital traces in information systems, often characterized by indirect or weak connections.

In this thesis, we consider system-level process mining as an approach that enables the meaningful characterization of business process dynamics, which we define as follows:

Definition 8 (System-Level Process Mining) *System-level process mining refers to the analysis that enables the study of process dynamics by capturing the evolution of high-level process behavior and its characteristics that emerge from aggregated data across multiple cases.*

In the following, we consider system-level process characteristics as a basis for studying business process dynamics through system-level process mining.

2.2.2 System-Level Process Characteristics

The study of business process dynamics using system-level process mining relies on system-level process characteristics and their evolution over time. A *system-level process characteristic* is a measurable property or feature that reflects the behavior of a business process as a whole and can be derived from an event log or a sublog that covers a certain period. These characteristics are derived from aggregating event data using a *feature extraction function* that considers multiple cases and corresponding event attributes to capture overarching process dynamics, such as workload, resource utilization, or the number of concurrent cases over time. The exact operationalization of the feature extraction functions depends on the available data and the concrete analysis objectives. The resulting process characteristics should provide a holistic view of the system's behavior over time. We formalize the concept as follows:

Definition 9 (System-Level Process Characteristic) *Let L be an event log and $\mathcal{F}$ a universe of all functions that extract process characteristics from event data. A system-level process characteristic is a measurable property $v \in \mathbb{R}^n$ that is derived by applying $f \in \mathcal{F}$ to $L' \subseteq L$, i.e., $v = f(L')$.*

It is important to note that when examining the system-level characteristics of a business process, we focus on characteristics that provide a holistic description of its behavior that evolves over time, exhibiting notable fluctuation.

Input Characteristics	**Internal Characteristics**	**Output Characteristics**
Number of arrived cases	Number of active cases	Number of completed cases
Avg. case inter-arrival time	Number of executed events, activities	Average lead, cycle, idle time
Distribution of case types	Activity duration distribution	On-time completion rate
...	Resource utilization	Compliance rate
	Number of rework loops	Error rate or defect rate
	Frequency of directly-follows relations	Average cost per case
	Frequency of eventually-follows relations	% of cases requiring rework
	...	...

Table 2.2: Classification of system-level process characteristics.

Classifications. Given that a business process can be viewed as a system, system-level process characteristics can be classified into three groups, as shown in Table 2.2:

Input characteristics. System input refers to the data, resources, and information that enter a system from its environment. In the context of a business process, inputs typically include incoming cases along with their characteristics, such as inter-arrival times, case types, and priority levels. It may also include the number of available employees, the equipment provided, and other operational resources.

Internal characteristics. System internal characteristics describe how the process operates as a whole during execution. These include dynamic metrics such as the number of active cases, the distribution of activity durations, resource utilization, and the occurrence of control-flow patterns.

Output characteristics. Output characteristics capture the outcomes or results produced by the process. These may include the number of completed cases, average lead or cycle time, error or defect rates, and key performance indicators such as service level agreement compliance or on-time completion rates. Such characteristics provide insights into the overall effectiveness, quality, and customer satisfaction aspects of the business process.

Process perspectives. From a process perspective, process characteristics may reflect one or more dimensions of the process. Table 2.3 shows examples of common system-level process characteristics and indicates which perspectives they capture. Some characteristics focus on a single perspective, such as directly-follows relations or average lead time, while others combine multiple perspectives to provide a more comprehensive view of the process, like the average resource cost per hour.

Representation. The representation of system-level process characteristics can

Process Characteristic	Control-flow	Time	Resource	Data
Frequency of directly-follows relations	●			
Average lead, cycle, idle time		●		
Total resource availability			●	
Distribution of arrived case types (e.g., standard vs. gold)				●
Average case variant duration	●	●		
Average resource utilization, waiting time		●	●	
Resource availability by type (e.g., manager, expert)			●	●
Average case inter-arrival time		●		●
Average activity execution duration per resource	●	●	●	
Average resource cost per hour		●	●	●

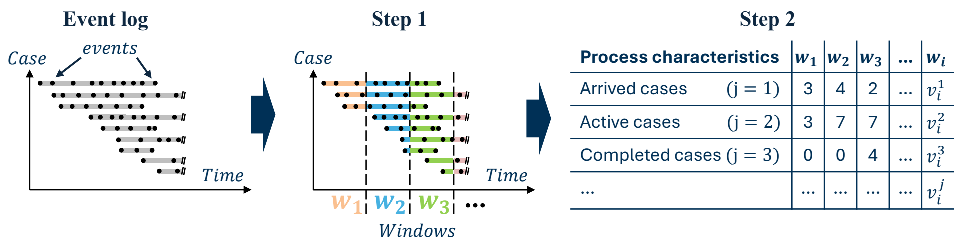
Table 2.3: Process characteristics and affected process perspective.

take various forms. In this thesis, we consider process characteristics in the form of a time series that enables the analysis and understanding of the temporal evolution of system-level process characteristics. A time series typically represents the observation of an underlying process over time, where values are collected through measurements taken at uniformly spaced time intervals, following a predefined sampling rate [42]:

Definition 10 (Time Series) *A time series V is an ordered sequence of n real-valued observations, denoted as:*

$$V = (v_1, v_2, \ldots, v_n), \quad v_i \in \mathbb{R}^d.$$

If $d = 1$, the time series is called univariate, meaning each observation is a single real value. If $d > 1$, the time series is multivariate, where each observation is a vector of d variables recorded simultaneously at the same time instant. In this case, the series captures multiple interrelated sequences evolving in parallel over time.



Process characteristics	w_1	w_2	w_3	...	w_i
Arrived cases (j = 1)	3	4	2	...	v_i^1
Active cases (j = 2)	3	7	7	...	v_i^2
Completed cases (j = 3)	0	0	4	...	v_i^3
...	...	...	...	...	v_i^j

Figure 2.8: Derivation of system-level process characteristics.

Derivation. The derivation of such time series from an event log involves two main steps, as illustrated in Figure 2.8. In *Step 1*, the recorded traces and their corresponding events are divided into equally spaced, non-overlapping time windows, ensuring that each event is assigned to exactly one window. In *Step 2*, the values of system-level process characteristics are computed for each window using a specific feature extraction function. The resulting value v_i^j represents the value of process characteristic j during time window i. These values are then assembled into separate univariate time series, one for each process characteristics of interest. Taken together, the resulting time series can be combined into a multivariate time series that describes the temporal evolution of system-level behavior captured in the event log. Figure 2.9 shows examples of resulting time series.

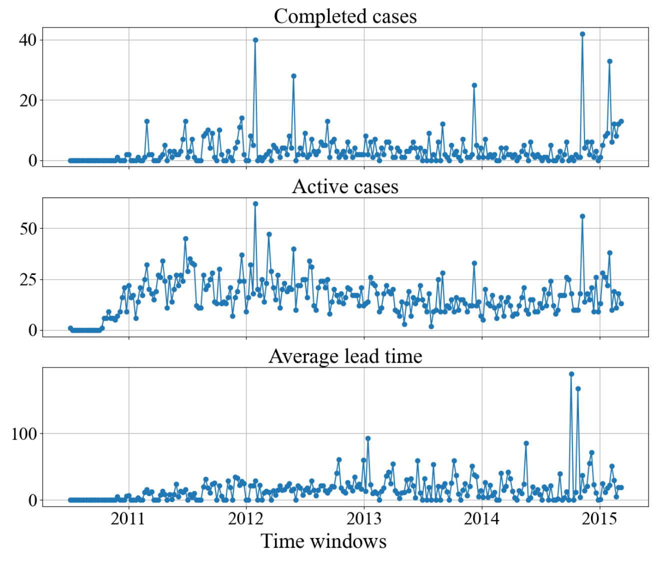


Figure 2.9: Examples of system-level process characteristics.

2.2.3 Applications

In recent years, system-level process mining has been applied in a wide range of process mining applications, where it has taken on different roles. In the following, we highlight these roles and present a set of representative applications.

Importance of system-level process mining. Figure 2.10 shows a Venn diagram that contrasts instance-level and system-level process mining as two complementary approaches. It highlights the importance of system-level process mining by distinguishing three types of tasks that benefit in different ways from system-level analysis and exhibit different levels of realized research potential. Type I tasks can be fully addressed through instance-level process mining, such as process discovery and conformance checking, and therefore do not require system-level process mining. Type II tasks can be addressed by instance-level techniques but benefit significantly from system-level analysis, for example, in predictive process monitoring, where system-level process mining plays a supportive role. Type III tasks can only be addressed using system-level process mining, such as the detection of dynamic bottlenecks or the identification of high-level events, where system-level process mining plays an essential role. Finally, type II and III tasks show the greatest remaining research potential that is addressed in this thesis and can be explored in the future.

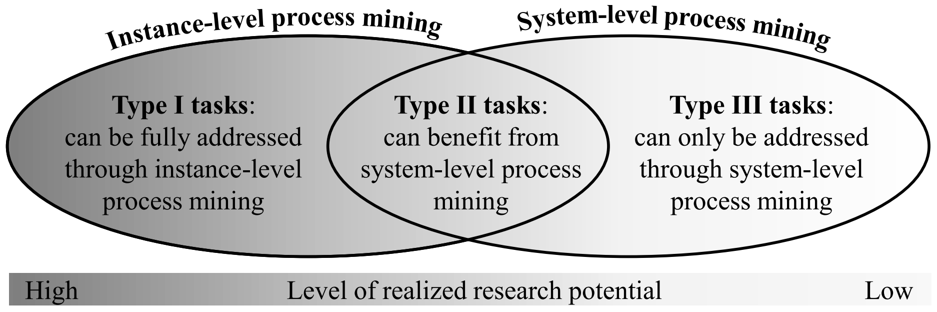

Instance-level process mining
System-level process mining
Type I tasks: can be fully addressed through instance-level process mining
Type II tasks: can benefit from system-level process mining
Type III tasks: can only be addressed through system-level process mining
High
Level of realized research potential
Low

Figure 2.10: Process mining tasks and the level of research potential.

Supportive role of system-level process mining. System-level process mining plays an important supportive role in many well-established process mining tasks. In the following, we briefly discuss several examples that demonstrate the usefulness of system-level process mining:

Performance analysis. Performance analysis is a key task in process mining [22]. It typically considers four main performance dimensions, collectively known as the Devil's Quadrangle: time, cost, quality, and flexibility [1]. In many cases, performance measures derived from an event log require the simultaneous consideration of all process instances. For example, time-related metrics such as average cycle time or lead time are computed based on all cases completed in the event log. However, while the derivation of such a measure describes the entry state of a process, it does not capture the time aspect of process dynamics, namely its evolution. This can be addressed by deriving performance measures for a fixed time period, for example, on a weekly basis. This approach offers a more dynamic view of how process performance evolves over time. Therefore, system-level process mining enables a more precise performance analysis that is differentiated in time.

Predictive process monitoring. Predictive process monitoring is a branch of process mining that focuses on forecasting the future behavior of ongoing (incomplete) process executions [22]. It is a typical example of a task that is commonly addressed at the level of individual process instances, based on the assumption that the value to be predicted depends only on intra-case information—such as the execution history of the specific case [22]. However, this intra-case assumption does not hold in many real-world scenarios [22]. For instance, when cases share limited resources, the completion time of one case may be strongly influenced by other cases running concurrently [5, 43]. In such cases, system-level process characteristics become important. As a result, system-level process mining has played a supportive role in predictive process monitoring by improving the accuracy of predictions made by traditional techniques [5].

Business process simulation. System-level process mining plays an important role in the context of Business Process Simulation (BPS). BPS refers to the use of simulation techniques to analyze, predict, and improve the performance of business processes based on event data extracted from information systems [1]. It is also an important tool in process mining for the redesign of organizational processes. By creating a digital process twin that includes strategic decisions, external factors, and feedback loops [44], simulation supports the estimation of the effects of process changes on several process aspects and enables effective what-if analysis [45]. Through BPS, it also becomes possible to study process dynamics under different operating conditions. However, while traditional BPS approaches usually rely on Discrete Event Simulation (DES) and focus on detailed instance-level behavior [46], system-level process mining provides a complementary foundation by enabling the discovery of system dynamics simulation models [6]. This approach captures aggregated patterns and interactions across many cases and offers a higher-level view. By abstracting individual events into system-level behaviors, system dynamics supports the construction of simulation models that reflect strategic decisions, external influences, and feedback loops found in real-world processes. As a result, system-level process mining connects data driven discovery with system dynamics modeling and provides a holistic simulation framework that complements and extends traditional auto-mined simulation techniques [47–49].

Essential role of system-level process mining. System-level process mining has played a vital role in addressing several emerging tasks that are difficult to handle when using only instance-level process mining.

Concept drift detection. Concept drift detection is a prominent example of a process mining problem that requires system-level process mining. Concept drift in process mining refers to changes in a process that occur while the process is still being analyzed [50]. To detect concept drifts effectively, current methods apply system-level process mining to extract information from multiple process traces. This information is then represented in forms such as multivariate time series [51], process graphs [52], multi-layered event knowledge graphs [53], or other formats that reflect the frequency and patterns of process characteristics. These representations are analyzed using techniques such as statistical analysis [31, 54, 55] or visual analysis [56] to identify possible drifts. In addition to detection, system-level process mining has also been used to explain concept drift in business processes [51]. By constructing time series that describe system-level process characteristics across different process perspectives, it is also possible to identify causal relationships using the concept of Granger causality, offering insights into the cause-and-effect relationships underlying significant changes [57].

Dynamic bottleneck detection. The problem of dynamic bottleneck detection is another important example of a process mining task that can only be addressed through system-level process mining. Dynamic bottlenecks occur when certain cases experience temporary delays at specific stages of a process [3]. These delays can become particularly costly when they propagate through subsequent process steps. Traditional techniques for bottleneck detection, which primarily focus on individual activities that cause delays, cannot identify dynamic bottlenecks that propagate throughout a process. By applying system-level process mining, a dedicated method has been introduced to detect cascades of system-level behavior, that is, patterns that frequently occur before dynamic bottlenecks arise [3]. These patterns are identified through temporal event sequences observed across multiple cases within the same process step. This approach enables the automatic detection of cascading, undesired behaviors that lead to dynamic bottlenecks.

High-level event mining. High-level event mining is an emerging research direction that enables the study of system-level process behavior using event data. High-level event mining can be seen as an alternative form of system-level process mining. Instead of relying on time series, as done in this thesis, it is based on the idea of deriving discrete events to study system behavior. Specifically, it focuses on extracting high-level events from event logs to capture system-level process behavior from a broader perspective [58]. Derivation of high-level events relies on grouping original events that occur in close temporal proximity and share common process characteristics [4]. The specific implementation of the feature extraction function may vary depending on the analysis goal and available data attributes. Each derived high-level event includes attributes such as activity, case, and timestamp. The resulting high-level event log can be used to examine how various process states emerge, spread, and are resolved across the system [58]. It also enables several downstream analysis tasks, such as identifying signs of performance issues, like delays, bottlenecks, long waiting times, or reduced throughput, and analyzing their root causes through correlation and causality analysis [58].

The process mining tasks discussed above are representative examples that illustrate the role system-level process mining has already played in the field. However, considerable research potential remains untapped. In the following chapters, we demonstrate how studying process dynamics through system-level process mining can further advance the state of the art and enable novel process mining tasks that have not yet been explored.

Chapter 3

Comprehensive Concept Drift Characterization

Business process dynamics can serve as a means to better characterize concept drifts. By analyzing these dynamics after process changes, it becomes possible to more effectively relate different changes and assess whether detected variations form part of a more complex drift. This chapter explains how this can be achieved, building on our paper "Comprehensive Characterization of Concept Drifts in Process Mining" by Alexander Kraus and Han van der Aa [9].

The remainder of the chapter is structured as follows. Section 3.1 introduces the topic. Section 3.2 discusses the current taxonomy, its limitations, and presents our proposed taxonomy. Section 3.3 explains the steps of our framework and the proposed algorithms. Our evaluation results are presented in Section 3.4, followed by related work in Section 3.5. Finally, we conclude in Section 3.6.

3.1 Introduction

Business processes are subject to change over time due to various internal and external factors, such as organizational adjustments, process enhancements, policy updates, and technological advancements. These changes can introduce *concept drifts*, which are situations when the characteristics of a business process have undergone significant changes [57] during the period when process execution data has been recorded, resulting in event logs that contain information on different versions of a process. The presence of such drifts in event logs can have a detrimental impact on the accuracy and usefulness of process mining results as these will be (partially) based on historical data that no longer represents the current process.

Therefore, to avoid incorrect or even misleading process mining results, *con-*

cept drift detection aims to identify and characterize concept drifts from event logs, striving to understand how a recorded process evolved over time. To achieve this, concept drift detection addresses the following three key tasks [31]: (1) *drift detection*, which involves detecting when drifts occurred, (2) *drift localization*, which aims to describe what was modified in the process, and (3) *drift characterization*, which considers how drifts manifest themselves over time. In this chapter, we particularly focus on the latter task, i.e., drift characterization, which seeks to understand how drifts in event logs unfold over time, i.e., whether they occur suddenly or gradually, and whether or not changes jointly form more complex patterns in the form of incremental or recurring drifts.

Despite numerous proposed solutions to automatically detect concept drifts [59], none of the existing techniques can comprehensively characterize drifts in event logs [60]. Specifically, most existing techniques focus on detecting isolated process changes that lead to sudden drifts [52, 61–67], with few others also differentiating between sudden and gradual ones [31, 54–56]. These techniques thus only provide a partial picture of detected concept drifts since they cannot recognize interrelations between process changes, which lead to more complex drifts in the form of incremental and recurring drifts.

In many process mining tasks, ignoring such complex drifts can lead to misleading results. For instance, applying existing concept drift detection techniques on a process may reveal that it went through a considerable amount of changes, each leading to possibly different process versions that should be analyzed separately. However, by considering the interrelations of these changes, it may become clear that this process is in fact subject to a seasonal pattern in which just two process versions alternate. Subsequent analyses can then target each of these versions individually. Similarly, performance analysis may be biased if the event data includes an incremental drift. In this case, proper performance analysis should separately consider recorded process behavior before the incremental drift and after it, while disregarding intermediate behavior that occurs during the period of the incremental drift. As a result, the state of the art provides only incomplete insights into the actual evolution of business processes over time.

This chapter addresses this limitation through two contributions. First, we propose an improved taxonomy that can be used as a basis for the comprehensive characterization of concept drifts, since we recognize that existing works are not just limited in their scope, but are actually grounded on imprecise and incomplete definitions. Second, we propose a three-step framework that automatically characterizes detected drifts in a comprehensive manner, following our proposed taxonomy. Our framework starts with the detection of isolated change points in the event log, marking significant shifts in process behavior. Next, using our change type detection algorithm, we identify actual process changes and categorize them as sudden or gradual. Finally, we determine concept drifts and their types from the detected process changes using our change interrelation detection algorithm. Conducted evaluation experiments show the accuracy of the developed framework and its algorithms compared to state-of-the-art techniques.

3.2 Problem Illustration

This section proposes a new taxonomy for the comprehensive characterization of concept drifts in process mining, which overcomes the limitations of existing definitions and can be used to accurately reflect the scope (and limits) of existing concept drift detection techniques. To achieve this, we examine the current taxonomy used to characterize concept drifts in Section 3.2.1, along with its limitations, which are detailed in Section 3.2.2. Finally, we present our proposed, improved taxonomy in Section 3.2.3.

3.2.1 Status Quo Taxonomy

This section provides a brief overview of concept drift detection and the current definitions (i.e., the *status quo taxonomy*) used to categorize and understand different types of concept drifts in process mining, which is provided by Bose et al. [31].

In the context of process mining, a *concept drift* refers to a situation in which a process is changing while being analyzed [31], which happens when an event log contains data stemming from different process versions. Figure 3.1 illustrates an example of a concept drift. In this example, the original process version is replaced with the new one that rejects the order if its components have not been pre-produced.

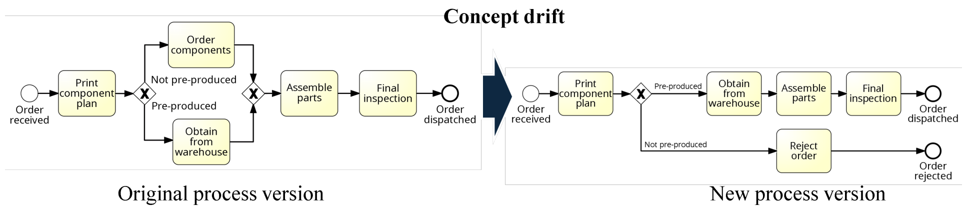


Figure 3.1: Example of a concept drift.

Concept drift detection aims to identify process changes based on event logs to obtain a comprehensive understanding of the overall evolution of a process over time. To achieve this, concept drift detection addresses the following key tasks [31]:

(i) *drift detection*, which involves detecting when drifts occurred,

(ii) *drift localization*, which aims to describe the process perspective(s) (control-flow, data, time, and data) affected by and specific modifications made to the process during a drift, and
(iii) *drift characterization*, which considers how drifts manifest themselves over time (e.g., suddenly versus gradually).

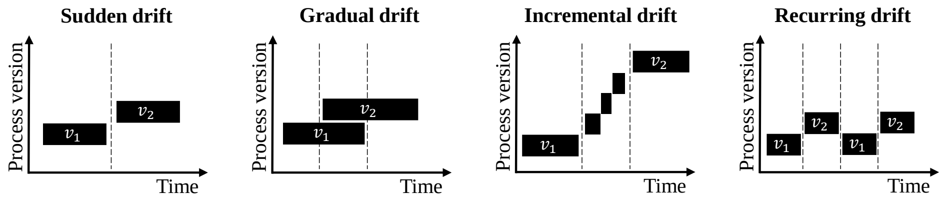


Figure 3.2: Concept drift types (adapted from [31]).

Our work primarily focuses on this latter task, i.e., change characterization, since this is, at best, only partially covered by existing works (see Section 3.5). In the taxonomy by Bose et al. [31], change characterization primarily focuses on classifying drifts into one of four *drift types* [31], which are illustrated in Figure 3.2:

1. A *sudden drift* occurs when a current process version is entirely replaced by a new one at a specific moment, and the new one takes over all ongoing cases [31]. This type of drift can occur in emergencies or when new regulations must be followed [60].
2. A *gradual drift* occurs when a current process version is replaced by a new one, and both versions coexist during a transition period [31]. Throughout this transition period, an increasing number of process instances begin to follow the new process version until the point at which the new version operates exclusively.
3. An *incremental drift* occurs when a current process version is replaced by a new one via smaller incremental changes [31]. For instance, this drift type occurs in organizations implementing successive business process quality improvements as part of a larger initiative.
4. A *recurring drift* occurs when a set of process versions reappear after some time [31]. Recurring drifts can occur in two forms: periodic and non-periodic. Periodic drifts follow seasonal patterns, such as reduced demand and resource needs during the summer holidays, whereas non-periodic drifts stem from changes that do not recur according to predetermined times, but rather from other conditions, such as a change in a process that is implemented when the workload is too high.

3.2.2 Limitation of the Status Quo Taxonomy

The aforementioned status quo taxonomy has several key limitations that hinder its usefulness when aiming to properly characterize concept drifts:

L1: Non-exclusive drift type classification. The four drift types defined by Bose et al. [31] are not mutually exclusive because they encompass two different levels of granularity. Specifically, sudden and gradual drifts characterize how individual process changes manifest themselves, whereas incremental and recurring drifts connect several process changes to each other. As a result, a single concept drift can be an incremental or recurring drift, but consist of individual sudden or gradual (or both) changes, as visualized in Figure 3.3[1]. Due to this limitation, drifts cannot be properly characterized when using the existing four drift types, since we either lose information at the high level, i.e., how changes are connected, or at the low level, i.e., if individual changes in a recurring or incremental drift occurred suddenly or gradually.

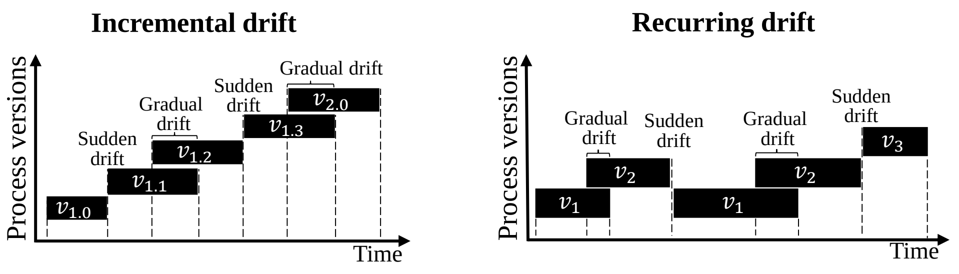


Figure 3.3: Incremental and recurring drifts can consist of sudden and gradual drifts, leading to non-exclusive classification (L1).

L2: Imprecise definition of incremental drifts. The definition of incremental drifts is imprecise, which means that it is not always possible to deterministically assign a drift type to specific observations. Specifically, the existing definition does not specify what makes a change incremental and, therefore, cannot be used to differentiate between a series of unrelated, relatively small drifts and a single, incremental drift, as, e.g., visualized in Figure 3.4.

L3: Incomplete definition of recurring drifts. Finally, the definition of recurring drifts is incomplete (and imprecise) because it does not align with the examples that are used to illustrate these drifts, essentially making the definition too narrow. For example, the definition, which requires a set of process versions to reappear,

[1]Note that Bose et al. [31] indeed remark on this aspect, but it is not picked up by subsequent works.

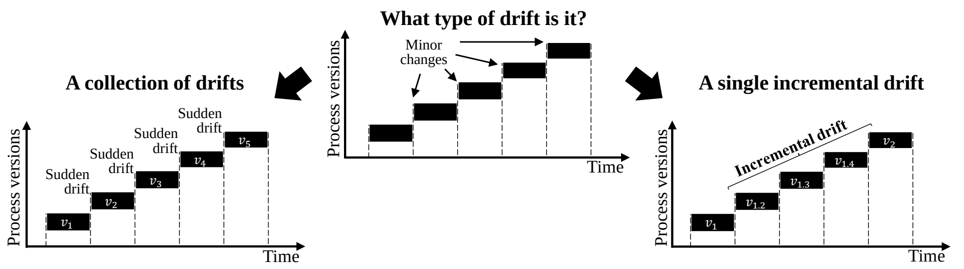


Figure 3.4: Minor changes can form an incremental drift, but also a sequence of independent sudden and gradual drifts (L2).

would exclude situations in which a single process version reappears every so often (whether periodically or not), as, e.g., visualized in case c) of Figure 3.5. In addition, if there are two recurring drifts in an event log, e.g., as depicted in case d) of the figure, then the existing definition would classify them as a single recurring drift, whereas it would be more precise to state that there are two recurring drifts, one involving versions v_1 and v_2 and another involving v_3 and v_4.

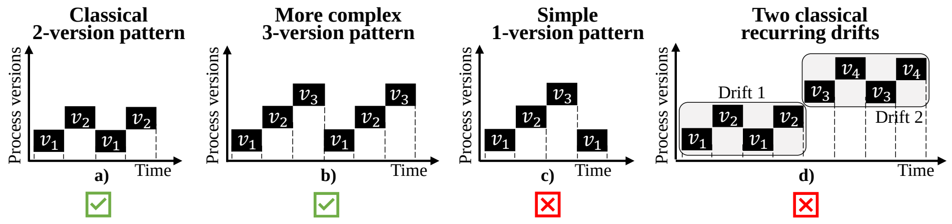

Classical 2-version pattern
More complex 3-version pattern
Simple 1-version pattern
Two classical recurring drifts
Process versions
Time
Drift 1
Drift 2
a)
b)
c)
d)

Figure 3.5: Instances of recurring drifts that align with the current definition (a, b) or do not align (c, d) (L3).

Due to these limitations, concept drift characterization has mainly centered on detecting isolated process changes and their types (sudden vs. gradual), overlooking more “complex" drifts, like recurring and incremental drift types, especially those involving lower-level sudden and gradual changes (see Section 3.5 for an overview).

3.2.3 A New Concept Drift Characterization Taxonomy

To overcome current limitations, we propose a new taxonomy to achieve a more clear and comprehensive characterization of concept drifts. The novelty of our taxonomy lies in its distinction between simple and complex drifts and its recognition that any change that forms or is part of a drift can be either sudden or gradual. In this manner, it overcomes the limitations observed in the status quo taxonomy.

In the following, we explain our taxonomy, visualized in Figure 3.6, in detail, and discuss how it addresses the identified limitations.

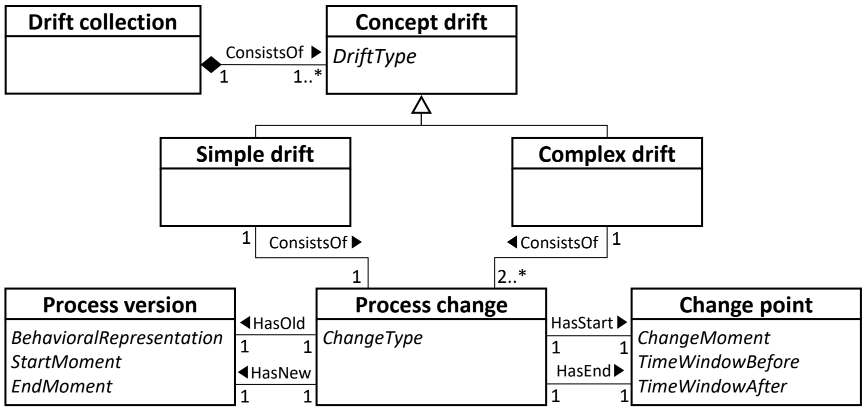


Figure 3.6: Our taxonomy for concept drift characterization.

Key Concepts of the New Taxonomy

As shown in Figure 3.6, a central concept in our taxonomy is a *process change*. We formalize the concept of process change and its characteristics as follows:

Process Change. A process change c is a modification of a business process that is characterized by a tuple:

$$c := (t, p^{start}, p^{end}, v^{previous}, v^{next}), \tag{3.1}$$

where:

- $t \in \{$“sudden", “gradual"$\}$ is the **process change type** that describes how a process change occurred, i.e., suddenly or gradually,

- $p^{start} \in \mathcal{I}$ and $p^{end} \in \mathcal{I}$ are **change points** that denote the start and end moments of a process change in L. They are represented by trace indices from $\mathcal{I} = [1, 2, \ldots, |\Sigma_L|]$, which indicate corresponding traces after which the behavior of recent process executions deviates significantly from prior behavior [55]. In the case of a sudden change type, both change points take the same value: $p^{start} = p^{end}$, whereas in gradual cases, they differ.

- $v^{previous}$ and v^{next} are **process versions** before and after the process change. A process version is a variant of a business process, defined by a specific set of activities and their underlying relations. It is characterized by a behavioral abstraction used to derive the version from an event log and is associated with a defined start and end point that mark the period during which the version was active.

We use $\overline{c} := \langle c_1, \ldots, c_N \rangle$ to denote a sequence of all process changes ordered by p^{start} that are present in L.
From the behavioral representations of the process versions associated with a process change, we can extract further essential characteristics for drift characterization and other concept drift detection tasks:

- *Change Severity* determines how much a process changed, which can be used, e.g., to detect minor changes when searching for incremental drifts. It can be computed using a function that takes the behavioral representation of old and new process versions as input and produces a value in the range (0, 1). Values closer to 0 indicate low severity, while values closer to 1 signify high severity.

- *Change Localization* identifies specific process modifications when comparing old and new process versions. This information is valuable to address the task of localization changes in concept drift detection. Depending on the behavioral representation, change localization can be quantified by examining sets of behavioral relations or patterns that are removed, introduced, or modified.

- *Change Perspective* reveals the affected process perspectives: control-flow, time, resource, and data. This determination aligns with the concept of perspective of change, as discussed by Bose et al. [31], and relies on the selected behavioral representation techniques.

One or more connected process changes constitute a concept drift, which we formalize as follows:

Concept drift. A concept drift d is a modification of a business process that is characterized by a tuple:

$$d := (\mathit{driftType}, c), \tag{3.2}$$

where $\mathit{driftType} \in \{$“sudden", “gradual", “incremental", “recurring"$\}$ and $c := \langle c_i, c_{i+1}, \ldots, c_j \rangle$, $1 \leq i \leq j \leq N$ is a subsequence of $\overline{c}$. Depending on the number of process changes, each drift is categorized into two groups, each consisting of two distinct drift types:

- A **simple drift** is a drift that consists of a single standalone process change, i.e., $|c| = 1$:
  - A sudden drift is given in the case of a sudden change type.

  - A gradual drift is given in the case of a gradual change type.
- A **complex drift** is a drift that consists of two or more connected process changes, i.e., $|c| > 1$:
  - An incremental drift is given by a consecutive sequence of at least two process changes with low change severity that adheres to the same process transformation initiative (business driver).
  - A recurring drift is defined as a collection of process changes that introduce one or more process versions, forming a pattern that recurs at least once.

A **drift collection** $d := \{d_1, \ldots, d_K\}$ is a set, consisting of $K \geq 1$ concept drifts. Given the aforementioned concepts, the objective of concept drift characterization is to establish a drift collection d from an event log L, such that d provides a comprehensive characterization of the concept drifts contained in L according to the taxonomy of Figure 3.6.

Benefits of the New Taxonomy

Our taxonomy jointly addresses the limitations of the status quo taxonomy. First, our taxonomy addresses the first limitation (L1) by introducing a mutually exclusive classification of drift types that is based on a process change as a building block for any concept drift and the number of process changes that belong to a drift (simple and complex drifts). Complex drifts (recurring and incremental), by definition, consist of several process changes, where each process change can happen suddenly or gradually, resolving the issues depicted in Figure 3.3.

We improve the definitions of recurring and incremental drift types, addressing limitations L2 and L3 by specifying how process changes should be related and form complex drifts. In the case of an incremental drift, the proposed notion of a shared business drive, like a BPM initiative to improve the average lead time of a business process, introduces a clear manner to differentiate an incremental drift from a sequence of standalone disconnected process changes. It also opens new opportunities for how incremental drifts can be detected, i.e., the necessary condition of at least two consecutive process changes with low severity can be extended with further conditions or restrictions. For instance, an additional condition could be related to time, i.e., a sequence of process changes should occur within a certain period, or they should be close in terms of change localization (i.e., which parts of a process are changed). In the case of recurring drifts, our definition of a recurring drift as a pattern of one or more process versions that reappear at least one time covers all possible recurring drift instances, including those depicted in Figure 3.5.

3.3 Framework

This section presents our framework for detecting and characterizing simple and complex concept drifts. Section 3.3.1 introduces the framework at a high level, while Sections 3.3.2–3.3.4 describe its main steps in detail.

3.3.1 Framework overview

Figure 3.7 outlines our proposed framework at a high level, detailing its input, main steps, and output.

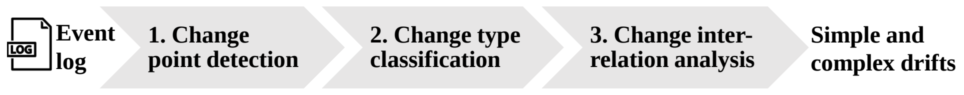

LOG
Event log
1. Change point detection
2. Change type classification
3. Change inter-relation analysis
Simple and complex drifts

Figure 3.7: Overview of the main steps of our framework.

Framework input. Our framework takes as input an event log L. We define the event log together with a trace σ, as well as the ordered collection of traces Σ_L, as specified in Section 2.1.2. Additionally, we use p_{first} and p_{last} as time points, respectively, corresponding to the timestamps of the first and last events in L.

Framework structure. Our framework consists of three key steps. Step 1 focuses on the detection of change points in log L, for which a range of existing state-of-the-art techniques can be employed. In Step 2, our framework turns the sequence of detected change points into a sequence of process changes by differentiating between individual points that correspond to sudden changes and pairs of consecutive change points that indicate a gradual process change. Lastly, in Step 3, we conduct change interrelation analysis to establish connections between the detected process changes, yielding a collection d of simple and complex drifts.

Framework output. Our framework's output is a collection d of identified simple and complex drifts, following the definitions in Section 3.2.3. Each identified drift is thus associated with a drift type and corresponding process changes, including process change types and the associated change points, providing a comprehensive characterization of the drifts in an event log.

3.3.2 Step 1: Change Point Detection

The first step of our framework identifies the moments in an event log when process behavior changes, resulting in a sequence of detected change points and the corresponding time windows, as illustrated in Figure 3.8. This step can be instantiated with any existing technique for change point detection in an event log, as this problem is the most extensively addressed task in concept drift detection, unlike other aspects of our framework, as demonstrated in Section 3.5. Additionally, recent work by Adams et al. [57] underscores the effectiveness of some of these techniques, which we also test and compare in our experimental evaluation in Section 3.4.

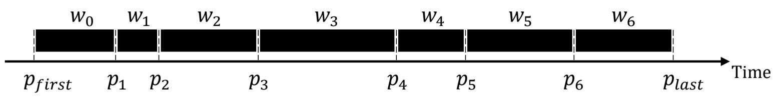


Figure 3.8: Outcome of the first framework step: change points are detected using an existing change point detection technique, splitting the time frame of an event log into time windows.

Once Step 1 is instantiated using any existing change point detection technique, we obtain a sequence of detected change points, which we represent as $p := \langle p_1, \ldots, p_N \rangle$. These change points split the time frame of log L into a sequence of $N+1$ time windows $w := \langle w_0, \ldots, w_N \rangle$, where w_0 represents the time window from p_{first} to p_1, and w_N corresponds to the time window from p_N to p_{last}. Figure 3.8 shows the outcome of the first framework step, assuming an event log with six process change points (our running example for this section).

Note that our framework terminates after the first step if p does not contain any change points; otherwise, the framework continues with the next step to reveal process changes from detected change points.

3.3.3 Step 2: Change Type Classification

The second step of our framework turns the sequence of detected change points into a sequence of sudden and gradual process changes. As illustrated in Figure 3.9, this involves differentiating between two cases:

1. Situations where the behavior that follows a change point p_i (i.e., during window w_i) corresponds to a distinct process version, signaling that a sudden change occurred at point p_i;
2. Situations where the behavior in w_i reflects a mix of the behavior that occurred before p_i (i.e., in window w_{i-1}) and the behavior that happens after the next change point p_{i+1} (i.e., window w_{i+1}). This indicates that the behavior observed in window w_i does not correspond to a distinct process version. Rather, it corresponds to a transition period in which the previous process

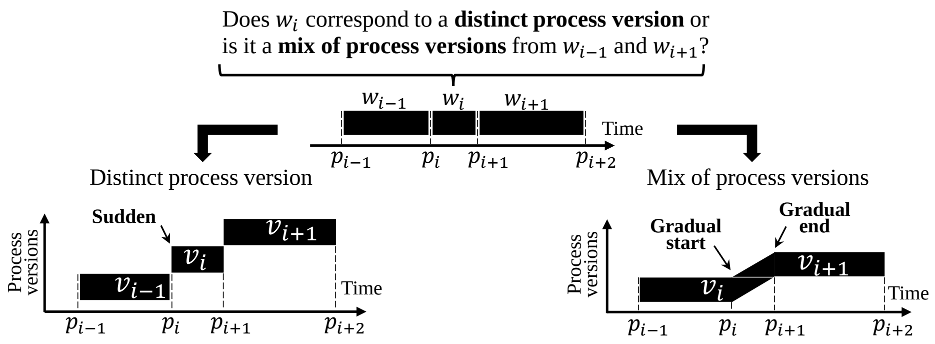

Does w_i correspond to a **distinct process version** or is it a **mix of process versions** from w_{i-1} and w_{i+1}?
w_{i-1} w_i w_{i+1}
Time
p_{i-1} p_i p_{i+1} p_{i+2}
Distinct process version
Mix of process versions
Process versions
Sudden
v_{i-1} v_i v_{i+1}
Gradual start
Gradual end

Figure 3.9: The main idea of the change type classification step.

version v_i is shifted out and the new one v_{i+1} is introduced, signifying a gradual process change that starts at p_i and ends at p_{i+1}.

To operationalize this idea, we have developed a change type classification technique, presented in Algorithm 1. It determines if a change point belongs to a sudden or gradual process change by considering the evolution of the process behavior before and after the detected change points. It takes an event log L and a sequence of change points p, as input and generates a corresponding sequence of $N \leq |p|$ process changes $\overline{c} = \langle c_1, \ldots, c_N \rangle$ as output. Following our definition of a process change in Section 3.2.3, each process change is associated with a change type (“sudden" or “gradual"), two corresponding start and end change points (p^{start} and p^{end}), and two process versions ($v^{previous}$ and v^{next}). For our running example, the output of this step is shown in Figure 3.10, which shows that six detected change points describe four process changes.

Next, we explain the main parts of Algorithm 1: behavioral representation and change point classification.

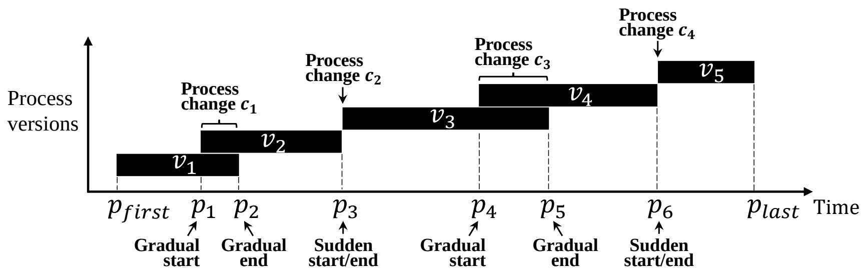


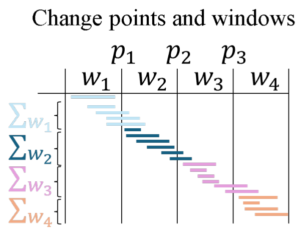
Figure 3.10: Outcome of the second framework step: each change point p_i is associated with a process change.

Algorithm 1 Change Type Classification

```
Input: Event log $L$, sequence of change points $p = \langle p_1, \ldots, p_N \rangle$
Parameter: Trend percentile $\alpha$
Output: Sequence of process changes $\overline{c} = \langle c_1, \ldots, c_N \rangle$
 1: procedure ChangeTypeClassification($L, p, \alpha$)
 2:     $w = \langle w_0, \ldots, w_{|p|} \rangle \leftarrow$ computeWindows($L, p$)    ▷ Define window sequence based on $p$
 3:     $t \leftarrow \langle t_1, \ldots, t_{|p|} \rangle$ with $t_i = \perp$ for all $i$ in $[1, |p|]$    ▷ Initialize a classification sequence
 4:     $\mathcal{B} \leftarrow$ getBehavioralMatrix($L, p$)    ▷ Derive behavioral matrix
 5:     for $i \in [1, |p| - 1]$ do    ▷ Iterate over all but the last change point in $p$
 6:         if $t_i = \perp$ then
 7:             $weigth_s \leftarrow 0, weigth_g \leftarrow 0$    ▷ Initialize the weights for s./gr. classification
 8:             for $b \in \mathcal{B}.relations$ do    ▷ Iterate over behavioral relations
 9:                 $roc_{b,1} \leftarrow$ rateOfChange($b, w_{i-1}, w_i$)    ▷ Get rate of change from $w_{i-1}$ to $w_i$
10:                 $roc_{b,2} \leftarrow$ rateOfChange($b, w_i, w_{i+1}$)    ▷ Gat rate of change from $w_i$ to $w_{i+1}$
11:                 $trend_b \leftarrow$ classifyTrend($roc_{b,1}, roc_{b,2}, \alpha$)    ▷ Sudden, gradual, or unchanged
12:                 if $trend_b$ = “sudden” then
13:                     $weigth_s \leftarrow weigth_s +$ calcWeight($b, [w_{i-1}, w_i, w_{i+1}]$)    ▷ Accum. weights
14:                 else if $trend_b$ = “gradual” then
15:                     $weigth_g \leftarrow weigth_g +$ calcWeight($b, [w_{i-1}, w_i, w_{i+1}]$)    ▷ Accum. weights
16:             if $weigth_g > weigth_s$ then
17:                 $t_i \leftarrow$ “gradual$_{start}$”, $t_{i+1} \leftarrow$ “gradual$_{end}$”    ▷ Assign change types
18:             else
19:                 $t_i \leftarrow$ “sudden"    ▷ Assign sudden change type
20:     if $t_{|p|} = \perp$ then    ▷ Handle edge cases where the final point is not assigned yet
21:         $t_{|p|} \leftarrow$ “sudden"    ▷ Assign sudden change type
22:     $\overline{c} = \langle c_1, \ldots, c_N \rangle \leftarrow$ setProcessChanges($p, t, \mathcal{B}$)    ▷ Define process changes
23:     return $\overline{c}$
```

Behavioral representation. Our algorithm first computes a behavioral representation to characterize the recorded process behavior during the time windows between detected change points (cf. Section 2.1.2). In our framework, we use directly-follows relations [34] observed within a specified time window, a widely used behavioral representation in process mining for concept drift detection [59]. It involves counting the frequency with which two activities are observed to directly follow one another within a single case. However, it is important to note that our algorithm's choice of behavioral representation is flexible, provided that it yields a numeric frequency distribution over a predefined set of relations or patterns across the windows. Therefore, it can also cover other types of relations (e.g., eventually follows), sets of relation types, such as those of a behavioral profile [35], or declarative process constraints [36].

To derive a behavioral representation, our algorithm takes an event log L with the sequence of detected change points p as input and then computes a *behavioral matrix*, denoted as $\mathcal{B}$. The behavior matrix consists of columns $\mathcal{B}$*.windows* that correspond to the time windows and rows $\mathcal{B}$*.relations* that correspond to the behavioral relations. Each cell $\mathcal{B}[b, w]$ of the behavioral matrix corresponds to the relative frequency of a relation b (e.g., a directly-follows relation between two activities) for a window w (line 4). To calculate such a relative frequency, our algorithm first identifies the set of traces $\Sigma_w \subseteq \Sigma_L$ that started during that window (according to the timestamp of the trace's first event)[2]. The algorithm then counts how often relation b is observed in Σ_w, e.g., how often we observe that an activity x is directly followed by activity y in these traces, and divides that count by the number of traces in Σ_w, yielding $\mathcal{B}[b, w]$.



Absolute frequencies

$$\begin{array}{c|cccc} & w_1 & w_2 & w_3 & w_4 \\ \hline b_1 & 10 & 9 & 12 & 15 \\ b_2 & 10 & 10 & 12 & 11 \\ b_3 & 10 & 8 & 10 & 9 \\ b_4 & 0 & 5 & 5 & 5 \end{array}$$

Relative frequencies (behavioral matrix)

$$\begin{array}{c|cccc} & w_1 & w_2 & w_3 & w_4 \\ \hline b_1 & 2.0 & 1.5 & 2.0 & 3.8 \\ b_2 & 2.0 & 1.7 & 2.0 & 2.8 \\ b_3 & 2.0 & 1.3 & 1.7 & 2.3 \\ b_4 & 0 & 0.8 & 0.8 & 1.3 \end{array}$$

Figure 3.11: Example of a behavioral matrix obtained from an event log.

Figure 3.11 illustrates the steps for constructing a behavioral matrix in a simplified example. In this example, an event log containing three change points is first converted into absolute frequencies for each time window, assuming four distinct relations across all recorded traces. These absolute frequencies are then transformed into relative frequencies, producing the final behavioral matrix.

Change point classification. Using the obtained behavioral representation, our algorithm next iteratively goes over the change points in p to classify them. For each index $i \in [1, |p| - 1]$, the algorithm considers a situation such as previously illustrated in Figure 3.9. Specifically, it considers the behavior observed during window w_i, in light of the behavior observed for its preceding (w_{i-1}) and succeeding (w_{i+1}) windows, in order to determine if a change point p_i corresponds to a *sudden* change or to a *gradual start*, i.e., the first change point in a gradual process change from p_i to p_{i+1}.

To decide between these two options, the algorithm first considers each behavioral relation surrounding point p_i individually before classifying p_i as being a *sudden* or *gradual start* point:

[2]The choice to compute a behavioral matrix according to the traces that start during w follows existing work on concept drift detection (cf. [57]) and is based on the assumption that the process version of a trace is fixed when it starts. If this assumption does not hold, the behavioral matrix $\mathcal{B}$ should, instead, be computed according to the events observed during w.

Relation-level classification. Given a change point p_i, our algorithm determines for each behavioral relation $b \in \mathcal{B}.relations$ if it was involved in the changes of points p_i and p_{i+1}, and, if so, if b changed in a sudden or a gradual manner.

To achieve this, we calculate the *rates of change* (roc) [68] of relation b when moving from time window w_{i-1} to w_i, denoted as $roc_{b,1}$, and from w_i to w_{i+1}, denoted as $roc_{b,2}$ (lines 9–10). For this, we use the following function:

$$\texttt{rateOfChange}(b, w_{i-1}, w_i) := \\ = \begin{cases} (\mathcal{B}[b, w_i]/\mathcal{B}[b, w_{i-1}] - 1) * 100 & \text{if } \mathcal{B}[b, w_{i-1}] > 0, \\ 100 & \text{if } \mathcal{B}[b, w_{i-1}] = 0 \text{ and } \mathcal{B}[b, w_i] > 0, \\ 0 & \text{if } \mathcal{B}[b, w_{i-1}] = 0 \text{ and } \mathcal{B}[b, w_i] = 0. \end{cases} \tag{3.3}$$

In Equation 3.3, the first case captures the usual computation of a rate of change, according to established definitions [68]. The second and third cases avoid division by zero errors, setting the change rate of a relation b to 100% if it appears in w_i but not in w_{i-1} (second case) and to 0% if it appears in neither w_i or w_{i-1} (third case).

Next to these change rates, we also consider a *trend percentile* α, which we use to determine if a change rate falls within the normal variance in a process or if it is part of an actual process change. Specifically, we consider the rates of change of all behavioral relations across all pairs of successive windows, i.e., the distribution of rates of change for a particular log. Given this distribution, we consider roc to be significant (i.e., a true process change) if it is lower than the value of the bottom percentile V_α (a significant decrease in the frequency of a relation) or greater than the value of the top percentile $V_{1-\alpha}$ (a significant increase) of all rates of change, otherwise, i.e., if $roc \in [V_\alpha, V_{1-\alpha}]$, it is considered to be part of the normal behavioral variation.

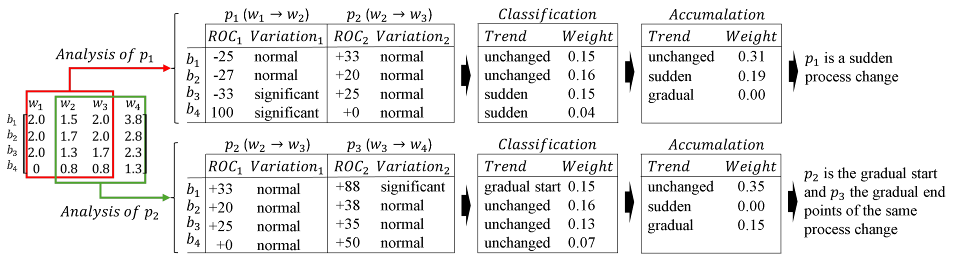
Given the change rates $roc_{b,1}$ and $roc_{b,2}$, and the trend percentile α, we then classify the trend of relation b for this particular change point using the following function (line 11):

$$\texttt{classifyTrend}(roc_{b,1}, roc_{b,2}, \alpha) := \\ = \begin{cases} \textit{unchanged} & \text{if } roc_{b,1} \in [V_\alpha, V_{1-\alpha}] \text{ and } roc_{b,2} \in [V_\alpha, V_{1-\alpha}], \\ \textit{gradual change} & \text{if } \texttt{sign}(roc_{b,1}) = \texttt{sign}(roc_{b,2}) \text{ and} \\ & roc_{b,1} \notin [V_\alpha, V_{1-\alpha}] \vee roc_{b,2} \notin [V_\alpha, V_{1-\alpha}], \\ \textit{sudden change} & \text{otherwise.} \end{cases} \tag{3.4}$$

This function classifies the relation b as *unchanged* when both $roc_{b,1}$ and $roc_{b,2}$ represent normal behavioral variation. If $roc_{b,1}$ and $roc_{b,2}$ indicate shifts in the same direction (both positive or both negative) with at least one reflecting a significant increase or decrease, then b is classified as a gradual change. Finally, the change is classified as *sudden* in all other cases.

Change point-level classification. Next, the algorithm classifies the change point as either *sudden* or as *gradual start* by considering the relevance of the identified trend types per relation. Specifically, given a relation b and a change point p_i, the algorithm assigns a *weight* to that relation. This weight is determined as the average of the relative frequencies of b in windows w_{i-1}, w_i, and w_{i+1} divided by the sum of all averaged relative frequencies of other relations. Using these weights, our algorithm categorizes the change point as *gradual start* if the cumulative weight of identified relations exceeds that of *sudden*; otherwise, it is labeled as a *sudden* (lines 12–19).

Note that if a point p_i is classified as a *gradual start* point, its successor, p_{i+1} is then immediately classified as a corresponding *gradual end* (see the illustration in Figure 3.9). Furthermore, if the last change point has not been assigned a type yet when completing the iteration, which happens when $|p| = 1$ or when the type of the second last change point is *sudden*, then the last change point is classified as *sudden* (line 21).


Analysis of p_1
Analysis of p_2
p_1 is a sudden process change
p_2 is the gradual start and p_3 the gradual end points of the same process change

$$\begin{array}{c|cccc} & w_1 & w_2 & w_3 & w_4 \\ \hline b_1 & 2.0 & 1.5 & 2.0 & 3.8 \\ b_2 & 2.0 & 1.7 & 2.0 & 2.8 \\ b_3 & 2.0 & 1.3 & 1.7 & 2.3 \\ b_4 & 0 & 0.8 & 0.8 & 1.3 \end{array}$$

	p_1 ($w_1 \rightarrow w_2$)		p_2 ($w_2 \rightarrow w_3$)	
	ROC_1	$Variation_1$	ROC_2	$Variation_2$
b_1	-25	normal	+33	normal
b_2	-27	normal	+20	normal
b_3	-33	significant	+25	normal
b_4	100	significant	+0	normal

Classification

Trend	*Weight*
unchanged	0.15
unchanged	0.16
sudden	0.15
sudden	0.04

Accumalation

Trend	*Weight*
unchanged	0.31
sudden	0.19
gradual	0.00

	p_2 ($w_2 \rightarrow w_3$)		p_3 ($w_3 \rightarrow w_4$)	
	ROC_1	$Variation_1$	ROC_2	$Variation_2$
b_1	+33	normal	+88	significant
b_2	+20	normal	+38	normal
b_3	+25	normal	+35	normal
b_4	+0	normal	+50	normal

Classification

Trend	*Weight*
gradual start	0.15
unchanged	0.16
unchanged	0.13
unchanged	0.07

Accumalation

Trend	*Weight*
unchanged	0.35
sudden	0.00
gradual	0.15

Figure 3.12: Example of change point-level classification based on the behavior matrix.

Figure 3.12 illustrate the change point-level classification using the behavior matrix depicted in Figure 3.11. We consider the first three columns of the behavior matrix to decide whether c_1 is a sudden process change or is the start point of a gradual process change. For this, we calculate the corresponding $roc_{b,1}$ and $roc_{b,2}$ for each relation b. Assuming a trend percentile of $\alpha = 3$, we get upper and lower percentiles for roc of 77 and -31, respectively. Given these percentiles, the variations of $roc_{b,1}$ and $roc_{b,2}$ are classified into “normal" or “significant" and then for each relation a trend is defined according to Equation 3.4. Finally, the weights are accumulated according to the trends. Since the total number of relation weights that belong to a sudden change (0.19) is greater than for gradual (0.0), the change point c_1 is classified as sudden. Following the same procedure, the change point c_2 is classified as a gradual start, therefore, the change point c_3 is a gradual end point. Overall, the three change points form two process changes: sudden (c_1) and gradual (c_2 and c_3).

Finally, the algorithm establishes a sequence of detected process changes $\bar{c}$ based on the change points, their classifications, and the behavioral matrix (line 22). Specifically, if a change point p_i is classified as a sudden change, then the algorithm generates a process change instance $c := (sudden, p_i, p_i, v^{previous}, v^{next})$, where $v^{previous}$ and v^{next} corresponding to the behavioral representation recorded in $\mathcal{B}$ for the time windows preceding and following p_i, respectively. Otherwise, if two consecutive change points p_i and p_{i+1} are classified as *gradual start* and *gradual end*, then the algorithm creates a process change instance $c := (gradual, p_i, p_{i+1}, v^{previous}, v^{next})$. Here, $v^{previous}$ and v^{next} correspond to the behavioral representation recorded in $\mathcal{B}$ for the time windows w_{i-1} and w_{i+1}, respectively.

3.3.4 Step 3: Change Interrelation Analysis

In the final step, our framework analyzes connections among the detected process changes to recognize a collection of simple and complex drifts as output. We illustrate this output in Figure 3.13, which shows that the four process changes of our running example form three concept drifts. Specifically, the gradual process change c_1 does not connect to other changes and is therefore classified as a simple drift. By contrast, changes c_2 and c_3 have been recognized to jointly form an incremental drift. Finally, since change c_4 leads to a previously observed process version (v_1), this represents a recurring drift in the process.

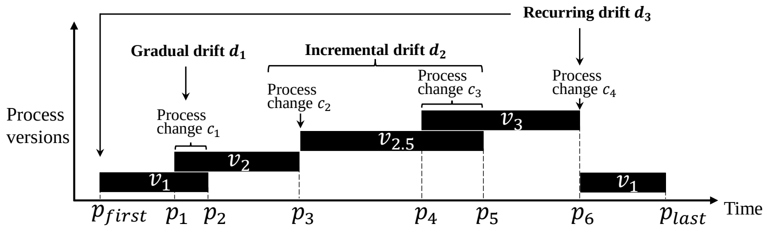


Figure 3.13: Outcome of the final framework step: every process change is either linked with other changes, creating a complex drift, or exists independently as a simple drift.

Algorithm 2 Change Interrelation Detection

Input: Sequence of process changes $\overline{c} = \langle c_1, \ldots, c_M \rangle$
Parameter: Thresholds for recurring and incremental behavioral similarity θ_{rec} and θ_{inc}
Output: Collection of detected drifts d

```
 1: procedure ChangeInterRelationDetection($\overline{c}, \theta_{rec}, \theta_{inc}$)
    ▷ Recurring drift detection
 2:     $\mathcal{S}^{rec} \leftarrow \{\}$    ▷ Initialize a collection to store detected sets of recurring process changes
 3:     $c_0 := (sudden, p_{first}, p_{first}, \perp, c_1.v^{previous})$    ▷ Define artificial start change
 4:     for $c_i$ in $\langle c_0, c_1, \ldots, c_M \rangle$ do    ▷ For all process changes...
 5:         if $c_i \notin \bigcup_{s \in \mathcal{S}^{rec}} s$ then    ▷ ... that are not already assigned
 6:             $s^{rec} \leftarrow \{c_i\}$    ▷ Initialize a set for changes similar to $c_i$
 7:             for $c_j$ in $\langle c_{i+2}, \ldots, c_M \rangle$ do    ▷ For indirect successors of $c_i$...
 8:                 if $c_{j-1} \notin s^{rec} \wedge$ isRecurringChange($c_j, \theta_{rec}$) then
 9:                     $s^{rec}$.add($c_j$)    ▷ Extend $s^{rec}$ with $c_j$
10:             if $|s^{rec}| > 1$ then    ▷ Check if $s^{rec}$ has more than one process change
11:                 $\mathcal{S}^{rec}$.add($s^{rec}$)    ▷ Add $s^{rec}$ to the collection of recurring sets
12:     $d^{rec} \leftarrow$ detectRecurringPatterns($\mathcal{S}^{rec}$)    ▷ Turn recurring sets into recurring drifts
    ▷ Incremental drift detection
13:     $\mathcal{S}^{inc} \leftarrow \{\}, s^{inc} \leftarrow \langle\rangle$    ▷ Initialize a collection to store detected incr. process changes
14:     for $c_i$ in $\overline{c}$ do    ▷ For all process changes...
15:         if $c_i \notin \bigcup_{s \in \mathcal{S}^{rec}} s \wedge$ isMinorChange($c_i, \theta_{inc}$) then    ▷ Check the condition for incr.
    change
16:             $s^{inc}$.extend($c_i$)    ▷ Add $c_i$ to the sequence of incremental process changes
17:         else
18:             if $|s^{inc}| > 1$ then    ▷ Check if $s^{inc}$ has more than one process change
19:                 $\mathcal{S}^{inc}$.add($s^{inc}$)    ▷ Add $s^{inc}$ to $\mathcal{S}^{inc}$
20:             $s^{inc} \leftarrow \{\}$    ▷ Reset $s^{inc}$
21:     $d^{inc} \leftarrow$ detectIncrementalPatterns($\mathcal{S}^{inc}$) ▷ Turn incremental sequence into incr. drifts

22:     $d^{simple} \leftarrow$ detectSimpleDrifts($\overline{c}, d^{rec}, d^{inc}$)    ▷ Turn other changes into simple drifts
23:     $d := d^{simple} \cup d^{rec} \cup d^{inc}$    ▷ Define drift collection
24:     return $d$
```

To operationalize this step, we have developed a change interrelation detection algorithm outlined in Algorithm 2. The algorithm takes the sequence of changes $\overline{c}$ stemming from the previous steps as input and evaluates relationships between the changes by comparing behavioral similarities of associated process versions. It produces a collection of concept drifts d as output, where process changes are connected, forming complex drifts, or independent, representing simple drifts. The main part of our algorithm focuses on detecting complex drifts since simple drifts automatically remain after larger, complex drifts have been detected. To identify complex drifts, our algorithm begins by searching for recurring drifts and then turns to incremental ones. The search for recurring drifts takes priority over incremental ones because it relies on a stronger condition regarding the similarity between different process versions.

Next, we describe Algorithm 2 following its key components: recurring drift detection, incremental drift detection, and simple drift detection.

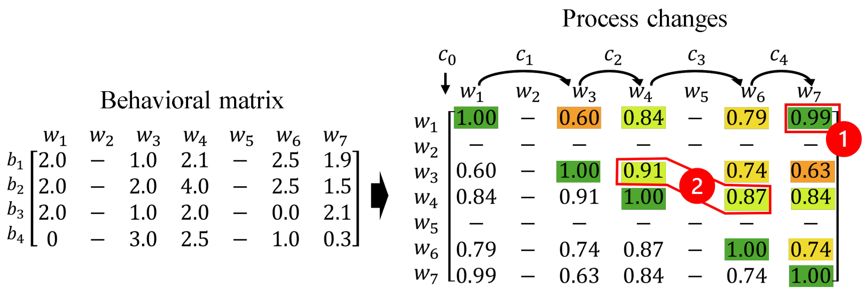
Recurring drift detection. Algorithm 2 detects recurring process changes by looking for process changes that lead to highly similar process behavior.

To do this, our algorithm starts by initializing a collection $\mathcal{S}^{rec}$ to store sets of process changes that lead to instances of the same process version (line 2). Since also the initial process version in w_0 can reoccur (see e.g., Figure 3.13), we establish a dummy change c_0, corresponding to a sudden change that appears at the start of the event log, i.e., at point p_{first} (line 3). Afterwards, the algorithm iterates over all process changes, including c_0 (line 4). For any change c_i that is not yet part of a recurring drift, the algorithm checks if there is any indirect successor c_j that leads to process behavior highly similar to the version following c_i, while ensuring that c_{j-1} is not part of a recurring drift. (line 8). If that is the case, it is recognized that c_j is recurring with respect to c_i.

Here, the function `isRecurringChange` quantifies the similarity between the two process versions $c_i.v^{next}$ and $c_j.v^{next}$. If the similarity is above a threshold θ_{rec}, then the process change c_j is considered to be a recurring change with respect to c_i. The function `isRecurringChange` can be operationalized using any measure that quantifies the similarity between two frequency distributions (over behavioral relations, i.e., the behavioral representation of two windows or process versions), such as the cosine similarity, a vector-based similarity measure, or the Earth mover's distance, which quantifies the distance between distributions, both of which are commonly applied in process mining settings [61, 69]. As a default option, we use cosine similarity, which demonstrates higher sensitivity in detecting added and removed behavioral relationships and achieved the best overall drift detection results in our evaluation.

Recurring changes are collected in a previously instantiated set s^{rec} (line 9). Note that our algorithm ensures that s^{rec} will never contain consecutive process changes, as recurring process version instances correspond to process changes that are separated by at least one other process change. Any set s^{rec} that contains more than one process change, i.e., any set that actually forms a recurring drift, is added to the set of recurring drifts $\mathcal{S}^{rec}$ (lines 10–11).

Finally, the set $\mathcal{S}^{rec}$ is turned into a set of recurring drifts d^{rec} (line 12). To do this, our algorithm looks for sequences of consecutive changes in the sets of S^{rec} that repeat two or more process versions in a particular order. Examples of this are seen in Figure 3.5, where case a) shows versions v_1 and v_2 in an alternating pattern, whereas case b) shows a repetition of a larger sequence $\langle v_1, v_2, v_3 \rangle$. In both cases, the sets that form these larger patterns are combined into a single recurring drift, whereas any set in $\mathcal{S}^{rec}$ that does not form a larger pattern is turned into a recurring drift by itself, as, e.g., seen for the $\{c_0, c_4\}$ in Figure 3.13.



$$\begin{array}{c|ccccccc} & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 \\ \hline b_1 & 2.0 & - & 1.0 & 2.1 & - & 2.5 & 1.9 \\ b_2 & 2.0 & - & 2.0 & 4.0 & - & 2.5 & 1.5 \\ b_3 & 2.0 & - & 1.0 & 2.0 & - & 0.0 & 2.1 \\ b_4 & 0 & - & 3.0 & 2.5 & - & 1.0 & 0.3 \end{array}$$

$$\begin{array}{c|ccccccc} & w_1 & w_2 & w_3 & w_4 & w_5 & w_6 & w_7 \\ \hline w_1 & 1.00 & - & 0.60 & 0.84 & - & 0.79 & \boxed{0.99} \\ w_2 & - & - & - & - & - & - & - \\ w_3 & 0.60 & - & 1.00 & \boxed{0.91} & - & 0.74 & 0.63 \\ w_4 & 0.84 & - & 0.91 & 1.00 & - & \boxed{0.87} & 0.84 \\ w_5 & - & - & - & - & - & - & - \\ w_6 & 0.79 & - & 0.74 & 0.87 & - & 1.00 & 0.74 \\ w_7 & 0.99 & - & 0.63 & 0.84 & - & 0.74 & 1.00 \end{array}$$

Figure 3.14: Example of change interrelation detection using a behavioral matrix.

To illustrate the idea of the recurring drift detection component of our change interrelation detection algorithm, we present an example shown in Figure 3.14. This example includes a behavior matrix corresponding to the scenario in Figure 3.13, along with the matrix displaying the cosine similarity between behavioral representations of different process versions across various windows[3]. Since the process changes c_1 and c_3 are of a gradual type, the windows w_2 and w_5 represent transition periods between two different process versions and, therefore, are excluded from consideration. From this similarity matrix, we can observe how the behavior of process versions before and after each change point compares to one another. Assuming $\theta_{rec} = 0.95$, the algorithm detects that the similarity between the process version in w_1 and the process version in w_7 (i.e., after c_4) is 0.99, which exceeds the threshold of θ_{rec} (see Point 1 in the figure). As a result, the process change c_4, along with c_0, is identified as a recurring drift.

Incremental drift detection. After recurring change detection, we start with the detection of incremental changes by identifying sequences of minor consecutive process changes in the sequence $\overline{c}$.

To do this, we start by initializing a collection $\mathcal{S}^{inc}$ to store sequences of consecutive incremental process changes (line 13). For each process change c_i in $\overline{c}$ that is not already part of detected recurring drifts, the algorithm checks if c_i is a minor change (lines 14–15). The underlying function, `isMinorChange` considers a change to be minor if the behavioral similarity between $c_i.v^{previous}$ and $c_i.v^{next}$ is above the threshold θ_{inc} (using the same similarity measure as used for recurring drifts). If c_i is indeed a minor change, we add it to the current sequence s^{inc}

[3]Our algorithm performs similarity analysis by iterating over a sequence of detected process changes. For clarity, however, we consider the matrix as a whole to provide a more straightforward explanation.

(line 16) and continue with the following process change. If c_i is not minor (or was already part of a recurring change), we add the sequence of minor changes observed so far, s^{inc}, to the set of incremental sequences, provided that s^{inc} contains more than one change (lines 18–19). Then, after resetting s^{rec} (line 20), we continue with the next process change.

To illustrate the idea of the incremental drift detection component, we again use the example in Figure 3.14. Since process changes c_0 and c_4 are identified as changes of a recurring drift, we need to only consider the remaining process changes and look for a sequence of at least two process changes with a minor change severity. Assuming $\theta_{inc} = 0.85$, a minor process change is given if the similarity between the process version before and after the process change is above θ_{inc}. In our example, among the remaining process changes c_1, c_2, and c_3, both c_2 and c_3 exhibit similarity values above the incremental threshold of 0.85 (see Point 2 in the figure). Therefore, these changes constitute an incremental drift.

In the end, the algorithm converts the set $\mathcal{S}^{inc}$ into a collection of incremental drifts d^{rec} (line 21), where each detected sequence of consecutive minor process changes within $\mathcal{S}^{inc}$ is turned into an incremental drift.

Simple drift establishment. After identifying complex drifts, the algorithm establishes a set of simple drifts (line 22) by turning any process change in $\overline{c}$ that is not part of a recurring or incremental drift into a stand-alone, simple drift, resulting in a set d^{simple}. In our example, shown in Figure 3.14, the only remaining process change is the gradual process change c_2. Since it is not part of a complex drift, it is classified as a simple drift.

Framework output. Finally, the algorithm returns the collection of detected drifts d given by the union of the detected simple, recurring, and incremental drifts (line 23).

Our framework thus identifies both simple and complex drifts in accordance with the taxonomy presented in Figure 3.6, specifying their drift types along with the associated process changes, distinguishing between sudden and gradual process changes. In this way, our framework supports the comprehensive detection and characterization of concept drifts from event logs.

3.4 Evaluation

This section describes the evaluation experiments we conducted to test the ability of our framework to detect different types of concept drifts recorded in event logs. Section 3.4.1 describes the data collection used for this purpose and Section 3.4.2 presents the general evaluation setup. Afterwards, Sections 3.4.3–3.4.5 describe the quantitative experimental results per framework step, whereas Section 3.4.6

presents a qualitative comparison of our work to a state-of-the-art technique. To ensure reproducibility, we have made the data collection, implementation, configurations, and raw results accessible in our public repository[4].

3.4.1 Data Collection

To evaluate our work, we require a collection of event logs that contain known (i.e., gold-standard) concept drifts of all types. Since a publicly available collection does not exist, we generated synthetic datasets using CDLG [12], a flexible tool that produces event logs with known concept drifts. We used CDLG to generate a dataset of 100 event logs. Each log is derived from a randomly generated process tree using PTandLogGenerator [70], which is then modified by a sequence of one to three randomly generated drifts, of different types.

PTandLogGenerator Parameter		**Process Tree Complexity**		
Type	**Name**	**Simple**	**Middle**	**Complex**
Number of activities	Minimum	6	14	20
	Mode	9	18	25
	Maximum	12	20	30
Control-flow probabilities	Sequence	0.70	0.25	0.25
	Choice	0.10	0.30	0.30
	Parallel	0.15	0.25	0.25
	Loop	0.05	0.20	0.20

Table 3.1: Key settings in PTandLogGenerator [70] used to produce process trees of various complexity levels.

To generate a diverse set of scenarios for concept drift detection, we varied two factors during log generation. First, we created the initial process trees with three levels of complexity: simple, middle, and complex. Table 3.1 shows the key parameter settings, such as the number of activities and the probabilities of control-flow operators, used in PTandLogGenerator for each complexity level. Examples of generated process trees for simple and complex cases are shown in Figure 3.15. Second, we also varied the characteristics of the drifts, including their severity and distribution over the timeline of the event log[5]. The complete set of parameters

[4]Project repository: `https://gitlab.uni-mannheim.de/processanalytics/concept-drift-characterization`.

[5]We intentionally excluded logs with only a single sudden drift, as these contain just one change point and therefore do not represent a classification problem.

used to generate this dataset, along with the initial and modified process trees and the event logs themselves, is available in our repository.

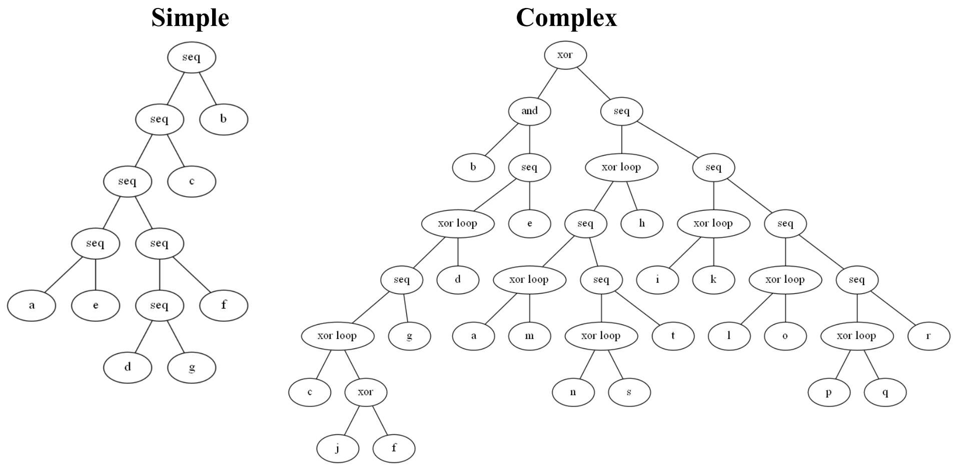


Figure 3.15: Examples of initial process trees of different complexity levels.

Table 3.2 provides an overview of the base dataset that contains 100 event logs. The logs contain a total of 198 drifts, i.e., 38 logs with 1 drift, 26 logs with 2 drifts, and 36 logs with 3 drifts. Simple sudden and gradual drifts involve a single process change, whereas complex recurring and incremental drifts comprise three process changes, resulting in a total of 426 process changes. Each process change can occur suddenly, resulting in a single change point, or gradually, resulting in two change points (gradual start and end), leading to 651 change points in total.

Table 3.2: Characteristics of the drifts in our base dataset consisting of 100 event logs.

Drift type	**# of drifts**	**# of process changes**	**Change points**			
			Total	**Sudden**	**Grad. start**	**Grad. end**
Sudden	35	35	35	35	—	—
Gradual	49	49	98	—	49	49
Incremental	58	174	263	85	89	89
Recurring	56	168	255	81	87	87
Total	**198**	**426**	**651**	**201**	**225**	**225**

To evaluate the robustness of our detection framework, we generated two variations of the base dataset by introducing noise into the event logs Specifically, we use an existing noise-insertion technique [71] that randomly inserts, removes, and swaps events in a fraction of the traces in an event log, obtaining datasets with 20% and 40% noisy traces (with all other characteristics, such as the number of drifts and change points, the same as depicted in Table 3.2). Consequently, the data collection used in our evaluation consists of 600 event logs.

3.4.2 General Setup

Framework implementation. We used Python 3 to implement a prototype of our framework, which is publicly available through the aforementioned project repository. Our implementation uses the *PM4Py* [72] library to handle event logs and the python library *Pandas*[6] for data processing steps.

Framework configurations. Step 1 of our framework can be instantiated using any of a range of existing change point detection techniques. Therefore, in Section 3.4.3, we evaluate existing techniques on our data collection and select the one with the best accuracy for the remaining experiments. In Step 2, to build a behavioral matrix, we rely on directly-follows relations over the control flow perspective to obtain behavioral representations of the process execution for each window. In particular, we extract all directly-follows relations from each trace and allocate them to the window containing the first event in the trace. Finally, in Step 3, we use cosine similarity to compare the behavioral similarity between windows in the behavioral matrix when performing incremental and recurring change analyses.

In our algorithms, we set the hyperparameters as follows: the trend percentile is $\alpha = 3$, the incremental threshold is $\theta_{inc} = 0.80$, and the recurring threshold is $\theta_{rec} = 0.95$. We determined these hyperparameters by testing a range of options: $\alpha \in [1, 2, 3, 4, 5, 10]$, $\theta_{inc} \in [0.5, 0.1, \ldots, 0.4]$, and $\theta_{rec} \in [0.50, 0.55, \ldots, 0.95]$ and selecting the combination that yielded the best average score, as reported in Experiment 3.2 (see Section 3.4.5) using an additional data collection for fine-tuning. This data collection was obtained using the same method as the evaluation dataset and exhibits similar concept drifts and noise levels.

Evaluation measures. To evaluate our framework's accuracy, we compare the detected change points, drifts, and drift types against those in the gold standard using precision, recall, and F1-score. Below, we provide a general representation of these measures since their specific operationalization differs per experiment.

Precision measures how many detected change points or drifts are correct based on their alignment with the gold standard. It is given as the ratio of true positive detection over the total number of true positive and false positive detec-

[6]Available at `https://pandas.pydata.org`

tions:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} \tag{3.5}$$

Recall measures the fraction of gold standard changes or drifts that are detected by our framework. It is given as the ratio of true positive detections over the total number of true positive and false negative detections:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}}. \tag{3.6}$$

Finally, the *F1-score* is the harmonic mean of precision and recall:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision + Recall}}. \tag{3.7}$$

Depending on the experimental setup, we use additional measures that aggregate the results, such as the weighted F1-score, which considers all drift types or change points and their respective support (number of instances).

3.4.3 Step 1: Change Point Detection

This section discusses experiments conducted to test the performance of various existing techniques that can be used to instantiate Step 1 of our framework, which detects change points in a given event log.

Experimental Setup

The first step of our framework relies on existing solutions for change point detection. For this reason, we evaluate existing change point detection techniques on our data collection and select the one with the best results for the remaining experiments.

Change point detection techniques. We test the same seven change point detection techniques used in a recent experimental comparison by Adams et al. [57]:

1. PRGRAPHS: Seeliger et al. [52] derives process graph features from an event log with the corresponding edge and node frequencies, using a sliding window to extract features. Change points are recognized if p-values from a statistical test fall below a threshold.
2. EMD: Brockhoff et al. [61] use sliding windows where each trace becomes a local multi-activity feature. Earth Mover's Distance measures distribution differences between different windows. Change points are identified as local maxima in a graph.

3. ADWIN/J: Martjushev et al. [54] address window size limitations in Bose et al. [31]. They introduce an adaptive window size, using recursive hypothesis tests on smaller windows near low p-values for precise drift location. High p-values lead to adaptive window growth, enabling segment skipping.
4. RINV: Zheng et al. [62] propose drift detection using a boolean relation matrix. Matrix entries show direct and eventual activity relations within cases. Drifts are identified by change point candidates. These are clustered using DBSCAN, with final change points found based on minimal centroid distance.
5. LCDD: Lin et al. [63] detect drifts by monitoring changes in directly-follows relations. They use two windows: a static complete window and a sliding detection window. The complete window size is determined based on local directly-follows completeness, ensuring all relations of the active process are included. A drift is reported when the two windows diverge concerning a directly-follows relation.
6. BOSE/J: Bose et al. [31] propose activity pair-based features. Using a fixed sliding window, features are extracted locally or globally. Their importance is assessed using J-measure, which quantifies the goodness of a rule. Then, populations are compared using non-parametric tests, and drifts are identified based on the resulting p-values.
7. PRODRIFT: Maaradji et al. [55] convert traces to partial orders of activities (called "runs"). Runs emerge from ordered traces, handling concurrent activities. Using sliding windows, run populations are extracted and then compared using statistical tests.

To identify the optimal parameter configuration for each technique, we employ the experimental framework [57]. This framework tests these techniques with different parameter options and reports various evaluation measures. Based on the best F1-score, we select the best parameter configuration for each technique, depicted in Table 3.3.

Evaluation measures. We use precision, recall, and F1-score introduced in Equation 3.5 to Equation 3.7. In this experiment, a true positive is recorded if a detected change point is correctly assigned to a gold-standard change point. To achieve this optimal assignment, we solve a linear program proposed by Adams et al. [57]. This program finds the best match between detected and actual change points (both are given via trace ID, i.e., the ordinal number of a trace in the log following the change), minimizing the total distance in terms of the number of traces between them. In contrast to Adams et al., who use a fixed absolute deviation of 200 traces, we use a *relative accepted deviation* between the detected and the actual change point. This relative acceptance deviation ensures a fair accuracy analysis for our

Table 3.3: Optimal parameter configurations for each change point detection technique.

Technique	**Parameters**
PRGRAPHS	Window size: 300, max. window size: 400, p-value:0.1
EMD	Window size: 150, step size: 1
ADWIN/J	Min/max adaptive window: 200/700, p-value: 0.4, step size: 20
RINV	Minimum relation invariance distance: 600, epsilon: 180
LCDD	Window size complete/detection: 400/400, stable period: 5
BOSE/J	Window size: 150, step size: 2
PRODRIFT	Window size: 400, step size: 2

data collection containing event logs with significant differences in the number of traces. In our evaluation, we consider relative acceptable deviation of 1%, 5%, and 10% and discuss results obtained for a relative acceptable deviation of 5% [7]. This choice is based on empirical justification. Given the characteristics of the data and the capabilities of the applied techniques, a relative acceptable deviation of 5% allows for effective differentiation in performance. When the deviation is set below 5%, the evaluation values are too low, as none of the techniques can detect changes at that level. In contrast, deviations above 5% lead to high values, as all techniques detect changes, making it harder to distinguish their performance. For all assignments, we identify a true positive when the distance between the detected change point and the gold-standard change point is less than or equal to 5% of the total number of traces in the log. Otherwise, the detected change points are classified as a false positive detection. The sum of false negatives and true positives (the denominator in the recall calculation) equals the total number of gold-standard change points.

Results

We present the evaluation results of the change point detection techniques, highlighting overall performance and the impact of noise, various change patterns, and different levels of change severity.

Overall performance. Table 3.4 provides an overview of the evaluation results. In terms of overall performance, we can distinguish three clear groups. The first group consists of two techniques demonstrating top performance: PRGRAPHS and EMD. PRGRAPHS is the best performer, with an average F1-score of approximately 0.70 across all datasets, while EMD also shows good performance but

[7]Our repository also contains results for other relative acceptable deviations.

achieves a slightly lower average F1-score of 0.67. In the second group, we find ADWIN/J, RINV, and LCDD, attaining average F1-scores between 0.53 and 0.58. Lastly, the third group encompasses BOSE/J and PRODRIFT, with average F1-scores of 0.31–0.33.

Table 3.4: Performance of the change point detection techniques across logs with different noise levels. Bold numbers indicate the best score for the particular column.

Technique	**Logs w/o noise**			**Logs w. 20% noise**			**Logs w. 40% noise**			**Avg. F1**
	Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	
PRGRAPHS	0.63	0.74	0.68	0.66	0.76	**0.71**	0.68	0.72	**0.70**	**0.70**
EMD	0.71	0.67	0.69	0.67	0.65	0.66	0.68	0.66	0.67	0.67
ADWIN/J	0.84	0.45	0.59	0.84	0.44	0.58	0.85	0.43	0.57	0.58
RINV	0.72	**0.87**	**0.78**	0.57	0.32	0.41	0.49	0.37	0.42	0.54
LCDD	0.57	0.63	0.60	0.34	**0.95**	0.50	0.35	**0.88**	0.50	0.53
BOSE/J	0.52	0.23	0.32	0.64	0.23	0.34	0.66	0.22	0.33	0.33
PRODRIFT	**0.99**	0.55	0.71	**0.89**	0.09	0.16	**1.00**	0.02	0.05	0.31

Noise impact. Regarding robustness to noise, PRGRAPHS, EMD, ADWIN/J, and BOSE/J maintain stable evaluation measures across noise levels, while RINV, PRODRIFT, and LCDD achieve lower accuracy for the noisy logs. Specifically, RINV's F1-score drops by close to 50% between the dataset without noise and those with 20% noisy traces (from 0.78 to 0.41). However, additional noise does not noticeably impact its performance further. By contrast, PRODRIFT experiences a substantial performance decline, with an 80% drop in F1-score when 20% noise is introduced (from 0.71 to 0.16), followed by an additional 70% decrease for the logs with 40% noise (from 0.16 to 0.05). The significant drop in performance is primarily due to a decline in recall when noise is introduced, decreasing to 0.09 at 20% noise and further to 0.02 at 40% noise. However, precision remains consistently high, exceeding 0.90. LCDD also appears highly sensitive to noise, often detecting non-existent changes in its results. Its precision is consistently low, while recall increases significantly for the noisy logs, reaching up to 0.95. Interestingly, both PRGRAPHS and BOSE/J exhibit an increase in precision as noise levels rise.

Change pattern impact. In addition to the noise impact, we assess the accuracy of change point detection techniques across different change patterns. We consider 6 different change patterns that are derived from three basic process changes and any combination of them: insertion of new activities ($\oplus$), deletion of existing activities ($\ominus$), and relocation of activities ($\Leftrightarrow$). Based on all correctly detected change points, we compute recall for each change pattern. As false positives do not corre-

Table 3.5: Performance of the change point detection techniques across different change patterns.

	Overall		**Recall (by change patterns)**						
Technique	**Prc.**	**Rec.**	$\oplus$	$\ominus$	$\Leftrightarrow$	$\oplus$&$\ominus$	$\oplus$&$\Leftrightarrow$	$\ominus$&$\Leftrightarrow$	$\oplus$&$\ominus$&$\Leftrightarrow$
PrGraphs	0.65	0.74	0.68	**0.73**	0.59	**0.90**	**0.88**	0.82	**0.97**
EMD	0.69	0.66	0.66	0.69	0.64	0.57	0.66	0.74	0.75
Adwin/J	0.84	0.44	0.54	0.51	0.34	0.42	0.25	0.17	0.28
Rinv	0.61	0.52	0.56	0.61	0.46	0.52	0.33	0.39	0.30
Lcdd	0.38	**0.82**	**0.86**	**0.73**	**0.80**	0.86	0.87	**0.91**	0.94
Bose/J	0.60	0.23	0.27	0.32	0.14	0.20	0.06	0.00	0.12
ProDrift	**0.98**	0.22	0.21	0.19	0.19	0.25	0.30	0.30	0.32
Support			555	537	309	279	138	66	69

Legend: “$\oplus$"- Insertion, “$\ominus$" - Deletion, “$\Leftrightarrow$" - Relocation.

spond to any specific change pattern, we report precision as an overall measure per technique.

Table 3.5 shows the evaluation results for different change patterns, revealing two main findings. First, detection capabilities vary across techniques for different change patterns. For example, LCDD achieves the highest overall recall of 0.82, showing the best performance for basic process changes. However, PRGRAPHS demonstrates superior change point detection accuracy for complex patterns, outperforming LCDD's recall in 3 out of 4 cases. Second, the complexity of the change pattern impacts detection accuracy differently among techniques. For some techniques, accuracy improves with complex patterns that involve combinations of two or three basic process changes, while for others, accuracy declines as complexity increases. For instance, PRGRAPHS, EMD, LCDD, and PRODRIFT show improved performance with complex patterns, with recall increasing by up to 30 percentage points (as observed for PRGRAPHS) compared to their average recall on simpler patterns with only one change type. Conversely, other techniques experience a drop in recall, indicating challenges in detecting more complex process changes.

Change severity impact. To complement the analysis of change pattern impact, we assess the effect of change severity on detection accuracy. To do this, we assign each change point a severity level, defined as the percentage of behavioral alteration following a process change. This percentage is calculated by comparing the gold-standard process trees before and after a process change. Specifically, we generate all possible traces (setting loop sizes to one) and derive directly-follow relation-

Table 3.6: Performance of the change point detection techniques across different change severity levels.

	Overall		**Recall (by change severity, in %)**				
Technique	**Precision**	**Recall**	$(0, 20]$	$(20, 30]$	$(30, 40]$	$(40, 50]$	$(50, 100]$
PrGraphs	0.65	0.74	0.71	0.70	0.75	0.78	**0.81**
EMD	0.69	0.66	0.66	0.64	0.71	0.68	0.62
Adwin/J	0.84	0.44	0.39	0.41	0.53	0.42	0.49
Rinv	0.61	0.52	0.49	0.46	0.52	0.52	0.65
Lcdd	0.38	**0.82**	**0.81**	**0.83**	**0.78**	**0.89**	0.78
Bose/J	0.60	0.23	0.25	0.16	0.21	0.26	0.29
ProDrift	**0.98**	0.22	0.20	0.25	0.24	0.20	0.22
Support			468	495	348	312	330

ships. These sets of relationships are then compared using the Jaccard coefficient, a similarity measure between finite sets defined as the ratio of the intersection size to the union size of the sample sets [73]. Finally, we group the resulting values into five intervals with comparable support.

Table 3.6 presents the obtained evaluation results across different levels of change severity. Regarding peak performance, Lccd achieves the highest recall for change severities up to 50%. However, for process changes that lead to extreme behavioral shifts above 50%, PrGraphs surpasses Lccd in performance. When analyzing the impact of change severity on accuracy across techniques, we observe that both PrGraphs and Rinv demonstrate improved accuracy as change severity increases, while the recall for other techniques remains relatively stable. Given these evaluation results, we adopt the PrGraphs technique to instantiate Step 1 of our framework in the remaining experiments (where applicable).

3.4.4 Step 2: Change Type Classification

This section discusses experiments conducted to test the performance of Step 2 of our framework, which aims to detect if the detected change points belong to sudden or gradual process changes.

Experimental Setup

Experiments. To comprehensively assess the performance of our framework when it comes to change type classification, we conduct two experiments: In *Experiment 2.1*, we assess the performance of Step 2 in isolation, which tests how well our change type detection technique works when provided with the gold-standard change points as input. Afterward, in *Experiment 2.2*, we assess the combined performance of Steps 1 and 2, i.e., using the change points detected by the PrGraphs technique as input for Step 2, which tests how well our framework can recognize sudden and gradual changes in general.

Evaluation measures. We use the following measures in the two described experiments:

Experiment 2.1 represents a classical classification problem, where each change point from the gold standard is classified as either *sudden*, *gradual start*, or *gradual end*. A true positive is thus recorded if the detected change point type is correctly assigned to its gold-standard type.

In Experiment 2.2, a true positive is recorded under two conditions: (1) the detected change point is correctly assigned to an actual change point, with a deviation of less than 5% of the total traces in the event log (same as for the evaluation of Step 1), and (2) the detected change point type (sudden, gradual start, or gradual end) corresponds to the gold standard. Otherwise, it is a false positive.

Baseline. To put the performance of Step 2 of our framework into perspective, we compare its accuracy against the ProDrift technique proposed by Maaradji et al. [55]. We select this technique as a baseline for two reasons: First, it stands out as the only technique that focuses on the automated detection of both sudden and gradual drifts, without requiring a manual indication of the drift type to be searched (e.g., in contrast to Bose/J [31]). Second, this technique uses a similar two-step procedure that first detects change points and then aims to detect gradual changes by considering sequences of three consecutive windows (see Figure 3.9). The technique is conceptually different, though, since their second step relies on a statistical test on distributions of partially ordered runs within sliding windows of traces, while our framework considers individual behavioral patterns.

Baseline implementation. To be able to use ProDrift as a baseline for both experiments, we need to decouple its two steps as well, allowing us to provide its second step with the gold-standard change points as input. Since its available Java implementation[8] does not support this, we implemented the second step in Python, following the procedure given in the paper [55, Definition 4]. Instead of using the Java-based JOptimitzer as the solver for the non-linear program, we use the *Optimize* module from SciPy[9] (version 1.10.0), using BFGS as the selected optimization method with the initialization (3, 3).

[8]Available at `http://apromore.org/platform/tools`

[9]Available at `https://docs.scipy.org/doc/scipy/`

Results

Experiment 2.1. Table 3.7 presents the change type classification results when using the gold-standard change points as input for Step 2. In this table, we compare the accuracy of our second framework step (see Algorithm 1) against the baseline across datasets with varying noise levels. For each dataset, we report the obtained evaluation measures, including overall measures weighted by the support for each type.

Our framework's algorithm greatly outperforms the PRODRIFT baseline, showing higher F1-scores for all change types, achieving a weighted F1-score of 0.79 for the logs without noise, versus 0.35 of the baseline. Our algorithm maintains a balanced precision-recall ratio across different change types. In contrast, the baseline struggles with identifying gradual change start and end instances, often mistaking them for sudden changes. This is evident in the baseline's high recall (0.86) and low precision (0.36) for sudden changes, along with relatively higher precision compared to recall for gradual start and end change types.

Table 3.7: Results of Experiment 2.1: Change type classification using gold-standard change points as input.

Technique	**Type**	**Logs w/o noise**			**Logs w. 20% noise**			**Logs w. 40% noise**		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
Framework Step 2	Sudden	0.63	0.85	0.72	0.76	0.79	0.77	0.75	0.73	0.74
	Gradual start	0.88	0.75	0.81	0.88	0.87	0.87	0.82	0.83	0.83
	Gradual end	0.89	0.76	0.82	0.89	0.88	0.88	0.83	0.84	0.83
	Overall	0.81	0.78	0.79	0.85	0.84	0.85	0.80	0.80	0.80
PRODRIFT Step 2 (baseline)	Sudden	0.36	0.86	0.50	0.31	0.92	0.46	0.31	0.85	0.46
	Gradual start	0.55	0.20	0.29	0.30	0.03	0.06	0.51	0.12	0.19
	Gradual end	0.49	0.18	0.26	0.35	0.04	0.06	0.53	0.12	0.20
	Overall	0.47	0.40	0.35	0.32	0.31	0.18	0.46	0.34	0.27

Support: Sudden - 201, Gradual start - 225, Gradual end - 225.

The difference between our algorithm and the baseline is even more pronounced when considering the datasets with noise. The performance of our algorithm even slightly improves for these datasets (from 0.79 to 0.85 and 0.80), whereas the baseline's performance further drops (from 0.35 to 0.18 and 0.27). A particular issue for the baseline is the detection of gradual changes in noisy settings, as, e.g., shown by the recall scores of 0.03 and 0.04 for the logs with 20% noise.

Experiment 2.2. Table 3.8 presents the change type classification results obtained

Table 3.8: Results of Experiment 2.2: Change type classification using change points from Step 1 as input.

Technique	**Type**	**Logs w/o noise**			**Logs w. 20% noise**			**Logs w. 40% noise**		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
Framework Steps 1–2	Sudden	0.27	0.41	0.32	0.29	0.42	0.35	0.31	0.39	0.35
	Gradual start	0.50	0.51	0.50	0.46	0.48	0.47	0.44	0.43	0.44
	Gradual end	0.49	0.48	0.49	0.46	0.46	0.46	0.44	0.43	0.43
	Overall	0.42	0.47	0.44	0.41	0.45	0.43	0.40	0.42	0.40
ProDrift Python-based (baseline)	Sudden	0.45	0.60	0.51	0.56	0.21	0.30	0.52	0.05	0.10
	Gradual start	0.57	0.13	0.22	0.00	0.00	0.00	0.00	0.00	0.00
	Gradual end	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00
	Overall	0.34	0.23	0.24	0.17	0.07	0.09	0.16	0.02	0.03
ProDrift Java-based (baseline)	Sudden	0.41	0.66	0.50	0.00	0.00	0.00	0.00	0.00	0.00
	Gradual start	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Gradual end	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	Overall	0.13	0.20	0.16	0.00	0.00	0.00	0.00	0.00	0.00

Support: Sudden - 201, Gradual start - 225, Gradual end - 225.

using the change points detected in Step 1. Similar to Table 3.7, this table provides evaluation measures for different change point types across datasets with varying levels of noise. However, we compare the joint accuracy of our framework's Steps 1 and 2 against both versions of the baseline.

First of all, compared to the results of Experiment 2.1, the evaluation results indicate a notable drop in performance for both techniques: For the logs without noise, our framework's weighted F1-score drops from 0.77 to 0.44, whereas the baseline's performance drops from 0.35 to 0.24. This is because the accuracy of change type classification (Step 2) depends on the accuracy of change point detection (Step 1). If a change point is not detected, it cannot be classified accurately, and if a change point is incorrectly detected, it also leads to classification errors. Therefore, if the quality of available change point detection techniques improves, the quality of our framework's subsequent steps will follow.

Despite the performance drop, it is important to note that our framework consistently outperforms the baseline in precision, recall, and F1-score across all change types. Furthermore, the results remain robust in the presence of noise. Notably, the Python-based implementation of the baseline struggles to identify gradual drifts when noise is present. To verify that this is not due to an implementation error, we provide evaluation results for the baseline using the existing Java-based tool. Using run-based configurations with adaptive window size, the obtained results for sudden process changes align with our Python-based implementation (including the change point detection accuracy of Step 1 corresponding to the results in Table 3.4). However, the Java-based implementation still struggles to detect gradual process changes, which is consistent with our Python-based results.

3.4.5 Step 3: Change Interrelation Analysis

This section discusses experiments conducted to test the performance of Step 3 of our framework, which goal is to detect concept drifts and determine their drift types from identified process changes.

Experimental Setup

Experiments. To demonstrate the accuracy of the framework's step, we conduct two experiments similar to the experiments in Section 3.4.4. In *Experiment 3.1*, we assume 100% accuracy of the first and second framework steps to measure the unbiased accuracy of Step 3. Then, in *Experiment 3.2*, we assume 100% accuracy only of the first framework step (since this is based on existing work) to evaluate the joint accuracy of Steps 2 and 3, thus evaluating how well our framework can detect simple and complex drifts based on detected change points.

Evaluation measures. For both experiments, we consider evaluation measures at two levels. First, we assess drift type detection accuracy at the *change-point level*, for which we check if change points are assigned to the right drift type (i.e., sudden, gradual, incremental, or recurring). Second, we assess accuracy at the *drift level*, for which we check if change points have been grouped together into drifts (of the right type).

Change-point level. To assess drift type detection accuracy at a change-point level, we consider a multi-class classification problem, where each change point is classified into one of four drift types: sudden, gradual, incremental, and recurring. We record a true positive if the detected drift type of a change point is correct given the gold standard; otherwise, it is a false positive for the detected type and a false negative for the gold-standard type. Note that we here report on weighted precision, recall, and F1-score, to account for imbalances in the dataset.

Drift level. Assessing drift type detection accuracy at a drift level is more complex than at a change-point level, since, in this case, it involves the (possible) assignment of multiple change points to a single drift, which must also be of the right type. Therefore, we consider a drift to be correctly detected (i.e., a *true positive*) if it has the right drift type and contains the right set of associated change points. If the drift type is correct, but the set of detected change points differs, we use the *Jaccard similarity* to quantify to what degree the set of change points is correct. The Jaccard similarity is calculated by dividing the number of change points in both the actual and detected drifts by the number of observations in either set. For example, if a detected incremental drift d includes three change points $\{c_1, c_2, c_3\}$, while the gold standard specifies four change points $\{c_1, c_2, c_3, c_4\}$, d is considered to be a 0.75 true positive. A drift is considered a false positive if its type is incorrect or if the Jaccard similarity is 0. The denominator in recall, representing the sum of true positive and false negative detections for each drift type, is determined by the total number of drifts that type in the gold standard.

Note that this assessment requires us to establish an alignment between the sets of detected and actual (gold-standard) drifts, i.e., determining which gold-standard drift corresponds to a detected drift, if any. This task is equivalent to solving the *two-dimensional rectangular assignment problem* [74], a well-known problem in operations research. The goal of this assignment is to efficiently distribute a pool of resources (such as individuals or employees) among a limited set of tasks, with the aim of minimizing the total associated cost matrix. In our case, the detected drifts serve as resources, and the actual drifts represent the tasks. The cost matrix is derived from the Jaccard similarity (with a negative sign) between pairs of drifts. We solve this problem using the Jonker-Volgenant algorithm [75], implemented in the Python library *scipy.optimize*.

Table 3.9: Results of Experiment 3.1: Drift type detection accuracy using gold-standard change points and types (Steps 1-2).

Drift type	**Supp.**	**Logs w/o noise**			**Logs w. 20% noise**			**Logs w. 40% noise**		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
Change-point level										
Sudden	35	0.46	0.91	0.62	0.46	0.89	0.61	0.46	0.89	0.60
Gradual	98	0.78	0.86	0.82	0.78	0.86	0.82	0.78	0.86	0.82
Incremental	263	0.93	0.69	0.79	0.93	0.68	0.79	0.93	0.67	0.78
Recurring	255	0.90	0.98	0.94	0.89	0.98	0.93	0.88	0.98	0.93
Overall (weighted)		0.87	0.84	0.85	0.87	0.84	0.84	0.86	0.83	0.83
Drift level										
Sudden	35	0.46	0.91	0.62	0.46	0.89	0.61	0.46	0.89	0.60
Gradual	49	0.78	0.86	0.82	0.78	0.86	0.82	0.78	0.86	0.82
Incremental	58	0.93	0.66	0.77	0.93	0.65	0.76	0.92	0.63	0.75
Recurring	56	0.83	0.79	0.81	0.82	0.81	0.82	0.81	0.81	0.81
Overall (weighted)		0.78	0.79	0.77	0.78	0.79	0.77	0.77	0.78	0.76

Results

Experiment 3.1. Table 3.9 presents the evaluation results of Experiment 3.1 for both measurement levels.

The results at the change-point level show that our framework performs well, achieving an average weighted F1-score of about 0.84 across all datasets. The results also show our framework's robustness to noise, maintaining consistent performance across all noise levels.

With respect to drift types, there are some drift-specific findings. We achieve good accuracy for gradual drifts with precision and recall consistently remains at approximately 0.78 and 0.86, respectively. The detection of recurring drifts yields a perfect recall of 0.98 and a precision close to 0.90. Notably, sudden drift detection displays relatively lower precision, but remains consistently high in recall. In contrast, detecting incremental drifts reveals an opposing trend, indicating that incremental changes are occasionally mistaken for sudden ones. This is closely related to the problem of differentiating a sequence of simple drifts from an incremental drift, as illustrated in Figure 3.4. Following our definition of a concept drift in Section 3.2.3, our algorithm considers the two necessary conditions for detecting incremental drifts: (1) a sequence of at least two consecutive process changes with (2) low change severity. Improving detection accuracy can be achieved by introducing additional criteria to assess whether or not process changes are part of the same transformation initiative, indicating they belong to the same incremental drifts.

At the drift level, the overall detection accuracy drops by about 8%pt. (from 0.84 to 0.76) compared to the change level. This decline relates to complex drifts (recurring and incremental) since the accuracy for simple drifts remains the same. This aligns with expectations, as complex drifts involve multiple process changes, making them more prone to misidentification at the drift level.

The decline in detection accuracy in recurring drifts is the primary factor contributing to the overall accuracy drop. The average recall over all datasets drops by about 18%pt. (from 0.98 to 0.80). Given the high recall at the change level, this suggests that recurring change points are occasionally assigned to the wrong recurring drifts. Consequently, precision also drops by about 7%pt. Another contributing factor is the drop of around 3%pt. in recall for incremental drifts. This can be attributed to the detection challenge between simple and incremental drifts.

Experiment 3.2. Table 3.10 presents the evaluation results of Experiment 3.2 for both measurement levels.

The results on the change-point level show solid performance with an average weighted F1-score of about 0.79 across all datasets. The results remain robust to noise, ensuring a stable performance across all noise levels.

Table 3.10: Results of Experiment 3.2: Drift type detection accuracy using gold-standard change points (Step 1) as input.

Drift type	**Supp.**	**Logs w/o noise**			**Logs w. 20% noise**			**Logs w. 40% noise**		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
Change-point level										
Sudden	35	0.35	0.77	0.48	0.44	0.77	0.56	0.45	0.71	0.55
Gradual	98	0.68	0.66	0.67	0.64	0.78	0.70	0.69	0.76	0.72
Incremental	263	0.87	0.61	0.72	0.90	0.64	0.75	0.89	0.69	0.78
Recurring	255	0.84	0.97	0.90	0.87	0.96	0.91	0.88	0.98	0.92
Overall (weighted)		0.80	0.77	0.77	0.82	0.79	0.80	0.83	0.81	0.81
Drift level										
Sudden	35	0.35	0.77	0.48	0.44	0.77	0.56	0.45	0.71	0.55
Gradual	49	0.70	0.66	0.68	0.64	0.78	0.70	0.70	0.76	0.73
Incremental	58	0.85	0.57	0.68	0.88	0.60	0.71	0.88	0.61	0.72
Recurring	56	0.75	0.75	0.75	0.78	0.77	0.77	0.81	0.78	0.80
Overall (weighted)		0.70	0.68	0.66	0.71	0.72	0.70	0.74	0.71	0.72

Compared to the previous experiment, the overall accuracy drops by about 5%pt (from 0.84 to 0.79), caused by errors from the change type classification algorithm (Step 2). The primary factor contributing to the decline in performance is the decrease in accuracy in simple drift detection. Specifically, we observe an average F1-score drop of about 12%pt. for gradual and about 8%pt. for sudden drifts across all datasets. This is anticipated, as simple drift detection relies on the identified change type from Step 2. If there is an error in identifying the change type, it follows that the corresponding drift type will be inaccurately determined. The errors in Step 2 also impact the accuracy of complex drift detection; however, the drop for recurring and incremental drift types is below 5%pt.

At the drift level, the average F1-score drops by 10%pt. compared to the change-level evaluation. The main drivers and their contributions are proportional to what we observe in Experiment 3.1.

Overall, our algorithms proposed for Steps 2 and 3 demonstrate their effectiveness at detecting drift types, with clearly evident noise resilience, since they maintain consistent results across different noise levels. In addition, our evaluation at the drift level highlights the challenges in accurately detecting and distinguishing complex drifts.

3.4.6 Steps 1-3: Comparison with the Baseline

In this section, we apply all three steps of our framework and highlight the advantages of the obtained results compared to the state-of-the-art solution.

Experimental setup

Baseline and configurations. We use the Visual Drift Detection (VDD) technique [56] as the state-of-the-art technique. It can detect simple and complex drift types from an event log and is the only (partially) comparable solution (see Section 3.5), although the approach is not automated. In our experiments, we use the online version of the VDD technique[10] with the suggested default parameters: window size 300, slide size 150, cut threshold 300.

Experiment. The VDD technique is not fully automated, requiring a user's interpretation of visualizations for complex drift detection. This makes automated comparisons infeasible. Therefore, we illustrate our framework's advantages by comparing results for a specific event log from our data collection (i.e., log number 90, with 20% noise). The selected log contains 64,594 events, 9,169 traces, 1,237 trace variants, and 8 distinct activities. The left-hand side of Table 3.11 shows that the log contains 8 change points that jointly form 5 changes and 3 drifts. Specifically, there are individual sudden and gradual drifts and a larger incremental drift, which encompasses 3 changes and 5 change points.

Results

Our framework. In Table 3.11, the right-hand side displays the outcomes of our framework. The first framework step accurately identifies 7 out of 8 change points, with one false positive (identifying a non-existent change point) and one false negative (not recognizing an actual change point), resulting in a F1-score of 0.88. The change type detection step identifies 1 out of 2 sudden change types, with one false negative detection, and successfully identifies 2 out of 3 gradual start/end changes, with also one false positive detection. This leads to a weighted F1-score of 0.67. The errors in the classification of change types appear primarily due to the errors in the previous step. In the final detection step, all change points are correctly assigned to their corresponding drift types, except for the first two, which together form a simple gradual drift, leading to the weighted F1-score of 0.58 (evaluated at the change level).

[10]Available online through the URL `https://yesanton.github.io/Process-Drift-Visualization-With-Declare/client/build/`

Table 3.11: Actual concept drift information for the event log 90 with 20% noise vs. detected drift information using our framework.

Gold standard			Detected drift information (our framework)		
Change point*	**Change type**	**Drift type**	**Change point***	**Change type**	**Drift type**
1042	gradual start	gradual	(+) 1429	(+) gradual start	(–) incr.
1748	gradual end	gradual	(+) 1879	(+) gradual end	(–) incr.
			(–) 2593	(–) gradual start	(–) incr.
2855	sudden	incr.	(+) 2989	(–) gradual end	(+) incr.
4089	gradual start	incr.	(+) 4195	(–) sudden	(+) incr.
4623	gradual end	incr.			
5847	gradual start	incr.	(+) 5976	(+) gradual start	(+) incr.
6623	gradual end	incr.	(+) 6694	(+) gradual end	(+) incr.
7991	sudden	sudden	(+) 7947	(+) sudden	(+) sudden

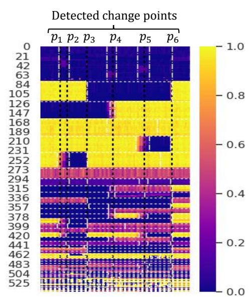
* We use trace ID to indicate the change point in a log.

Baseline. Figure 3.16 presents a primary outcome of the baseline technique, i.e., the Drift Map, Drift Charts, and autocorrelation plots with further measures that are used to support visual analysis. The Drift Map (Figure 3.16a) shows over 525 detected behavioral rules (y-axis), organized into 55 behavioral clusters (indicated by horizontal dashed white lines). Within each cluster, white vertical lines indicate change points, while black vertical dashed lines highlight global change points spanning over all clusters. The Drift Charts (Figure 3.16b depicts a drift chart for a selected cluster) categorize drifts, helping to determine if drifts exist, their patterns over time, and the stability or drift in behavior. Finally, Figure 3.16c depicts autocorrelation, to detect recurring drifts, and erratic measure to differentiate, for instance, gradual drift from incremental.

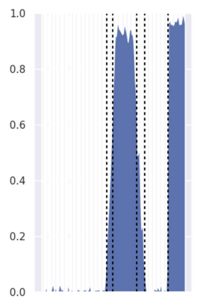
Next, we use VDD to try to obtain the same types of insights provided by our framework's three steps:

Step 1: Change point detection. The baseline identifies a total of 6 change points. The change points p_1, p_2, p_3, and p_6 align with the gold standard, while the remaining p_4 and p_5 are within the gradual transition phases. Unfortunately, the figure does not allow for a precise assessment of whether p_4 and p_5 adhere to the accepted deviation of 5% from the log size. Consequently, our estimation of the accuracy in the change point detection step yields a precision range of 0.67 to 1.00 and a recall range of 0.50 to 0.75, depending on whether or not the change points p_4 and p_5 are considered correct. This results in an F1-score spanning from 0.57 to 0.86 with an average value of 0.71. Even in the best-case scenario (F1-score of 0.86), the performance is below the accuracy of our framework's Step 1.

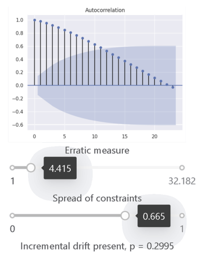
Step 2: Change type classification. Regarding process change detection, the



(a) Drift Map.



(b) Drift chart (cl. 13).



Erratic measure: 4.415 (1 – 32.182)
Spread of constraints: 0.665 (0 – 1)
Incremental drift present, p = 0.2995

(c) Autocorr. (cl. 13).

Figure 3.16: The output of the VDD technique.

baseline does not automatically distinguish between gradual and sudden process changes. However, by visualizing different behavioral clusters, we can identify when detected change points suggest a gradual process change. For example, multiple clusters between the first and second change points indicate a gradual shift in certain behavioral patterns. Thus, we can conclude that the baseline correctly identified the first gradual process change, the second sudden process change, and the final sudden process changes. However, change points p_4 and p_5 are misclassified as sudden process changes due to an error in the change point detection step. Overall, VDD thus leads to a perfect recall and precision of 0.50 for sudden process changes, with perfect precision and 0.33 recall for gradual process changes, resulting in a total weighted F1-score of 0.54. This result is 13%pt. below the accuracy of our framework's Step 2, which, furthermore, does not rely on manual interpretation of various visualizations.

Step 3: Change interrelation detection. The identification of drifts and their drift types using VDD is highly challenging. The tool's visualization of clusters does not differentiate between affected behavioral rules that relate to different drifts. Therefore, based on the Drift Map and Drift Charts, it is not feasible to understand the big picture, i.e., that the first two change points as well as the last one are simple drifts that do not belong to the incremental drift in between. Moreover, some visualized clusters indicate a recurring drift pattern, though such patterns are not present in the gold standard. Therefore, drawing conclusions about the overall drift scenario, in terms of exact simple and complex drifts, remains speculative.

Overall insights. In summary, the VDD technique visualizes and locates change points well, but understanding the different types of process changes and their relationships, and determining if they form a single incremental drift or a set of unrelated drifts, is still difficult. Consequently, achieving a complete understanding of the overall concept drift is not feasible, even with manual effort. In contrast, our proposed framework enables an automated detection of drifts and achieves better results for all three steps compared to the baselines. Therefore, our framework provides an important step towards automated and comprehensive detection of concept drifts from event logs.

3.5 Related Work

In this section, we discuss existing concept drift detection techniques in light of our taxonomy, specifically showing their coverage of the different change and drift types. Given the scope of the addressed problem, we focus on offline detection techniques that use event logs as input and yield information about the type of change or drift as output. We exclude online concept drift detection techniques that rely on event streams from consideration, as this problem setting introduces additional constraints and limitations (such as increased computational overhead, real-time processing requirements, and the need for continuous monitoring) that make direct comparisons with our framework and other offline techniques unreasonable.

Table 3.12: Classification of different concept drift detection and characterization techniques according to our taxonomy.

Technique	**Change point detection**	**Change type detection**	**Drift type detection**			
			Simple drifts		**Complex drifts**	
			Sud.	**Grad.**	**Incr.**	**Rec.**
Various works	✷✷					
Bose et al. [31]	✷✷	✷	✷	✷		
Martushev et al. [54]	✷✷	✷	✷	✷		
Maaradji et al. [55]	✷✷	✷✷	✷✷	✷✷		
Yeshchenko at el. [56]	✷✷	✷	✷✷	(✷)	(✷)	(✷)
Proposed framework	✷✷	✷✷	✷✷	✷✷	✷✷	✷✷

Legend: "✷✷" - automated, "✷" - semi-automated, "(✷)" - non-automated.

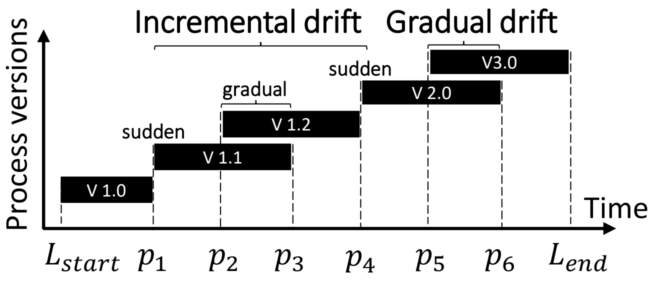
Table 3.12 provides an overview of the scope and automation level of existing drift detection and characterization techniques. Since establishing the problem and importance of concept drift detection in process mining more than a decade ago [50], various techniques have been proposed to detect and characterize them, as highlighted in recent literature reviews [59, 60]. However, the majority of existing techniques detect change points in an event log, aiming to identify a moment when a process behavior significantly changes [52, 54, 61–67]. In terms of drift characterization, they do not contribute to the understanding of the underlying drift types.

Change type detection. When it comes to detecting not only process change points but also process change types, there are two techniques that distinguish between gradual and sudden changes:

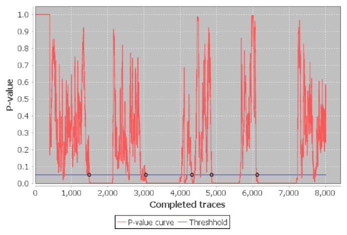
Bose et al. [31] introduced concept drift detection in process mining and presented a method for automatically detecting process change types. Their approach uses statistical testing of feature vectors. However, users should indicate which change type should be searched for, i.e., the techniques can only detect sudden or gradual drifts, not both. The method lacks automation, requiring user manual feature selection, assuming prior knowledge of drift characteristics. Additionally, testing all possible activity combinations can also be computationally demanding. Finally, users must specify a window size for drift detection, potentially missing some drift occurrences. To address the window size limitation, Martushev et al. [54] introduced adaptive windowing, which automatically adjusts the window size when searching for drifts. However, this approach requires users to define minimum and maximum window size parameters as upper and lower boundaries for automated window adaptation. Given the mentioned constraints, both techniques have limited capability in detecting process change types, especially in an automated manner.

The work by Maaradji et al. [55] introduced an alternative technique for change type detection, addressing the limitations of Bose's technique. Their method offers an automated and statistically grounded solution for identifying both sudden and gradual process change types, representing the current state of the art in simple drift detection. Their approach involves a two-step process: initially, it detects change points to identify sudden drifts (see Figure 3.9), and then it employs postprocessing on the output of the sudden drift detection algorithm to detect gradual drifts. Specifically, they analyze the behavior within the intervals between two change points by statistically assessing whether it exhibits a mixture of behavior distributions before and after these points [55]. However, this distribution-based method has a significant drawback. In situations where noise is present in an event log, particularly concerning gradual drift detection, their approach experiences a notable decrease in detection accuracy, as demonstrated in our evaluation (Section 3.4).

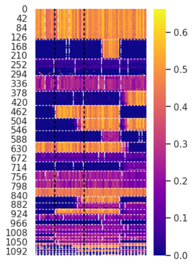
Drift type detection. Change type detection techniques can only detect simple drifts. Therefore, their usefulness in practice is notably limited when it comes to drift characterization since event logs might include complex drifts, which is apriori unknown. These techniques do not consider the interrelations between process changes, leading to a collection of simple drifts if the underlying drift in an event log is complex. We demonstrate it using an exemplary concept drift scenario with two drifts: a complex drift of incremental type followed by a simple drift of gradual type, depicted in Figure 3.17a. The incremental drift consists of three process changes: two sudden drifts at the change points p_1 and p_4, and a gradual change between p_2 and p_3. Figure 3.17b shows the detected change points using the technique by Maaradji et al. [55]. Given the output, it is impossible to conclude that the first four change points belong to an incremental drift and the last one is an independent gradual drift.



(a) Concept drift scenario with incremental and gradual drifts.



(b) Outcome of concept drift detection technique for simple drifts [55].



(c) Outcome of concept drift detection technique for complex drifts [56].

Figure 3.17: The outcome of the state-of-the-art concept drift detection techniques cannot comprehensively characterize a complex concept drift.

The comprehensive detection and characterization of drift types requires techniques that can simultaneously detect simple and complex drifts. The Visual Drift Detection (VDD) technique, introduced by Yeshchenko at el. [56], represents the current state of the art. VDD uses concepts like temporal logic, DECLARE constraints [76], and time series analysis. It groups similar declarative behavioral constraints and automatically identifies change points. The system provides visual aids such as the Drift Map, Drift Charts, and a directly-follows graph. While these visualizations effectively localize change points, it has a significant limitation. The process of identifying simple gradual and complex drifts is not automated and depends on a user's visual interpretation. As a result, recognizing complex drift types or several different drifts within the same event log can be challenging and subjective, limiting the ability to characterize drift types and the overall process evolution. Figure 3.17c shows the Drift Map, the main output of the VDD technique, for the same concept drift scenario in Figure 3.17a. Given the visualization, it is impossible to see if detected change points belong to a complex or form a sequence of simple drifts. In our evaluation (Section 3.4), we further demonstrate these limitations and compare the VDD approach to our work.

3.6 Conclusion

In this chapter, we contribute to the more comprehensive characterization of concept drifts recorded in event logs by introducing an improved drift type characterization taxonomy and presenting a three-step framework for the automated detection of drift types.

Our improved drift type classification taxonomy classifies drifts into simple and complex ones, relying on process changes and their properties as the core elements of concept drifts. Our taxonomy addresses existing inconsistencies by providing an exclusive drift type classification. Our taxonomy enhances the detection and evaluation of concept drifts, especially complex drifts with inter-connected process changes. Following our taxonomy, we proposed an automated, three-step framework for the comprehensive detection and characterization of concept drifts. Our experiments demonstrated that our change type and change interrelation detection algorithms used in Steps 2 and 3 provide accurate results, offering a more comprehensive understanding of drift types and the overall evolution of the process compared to state-of-the-art solutions.

In future work, we plan to enhance our framework in several directions. First, we aim to refine the change point detection step, which significantly influences the overall framework accuracy. Our evaluation of existing change point detection techniques revealed a need for a more precise approach to detect change points that correspond to sudden, gradual start and end points, especially, in the presence of noise. Second, we plan to improve the identification of incremental drifts by considering additional aspects, like change localization and time aspects, that can improve detection accuracy. Finally, we see the potential to expand this framework with drift localization information that, based on identified drifts and process versions, goes beyond localization for individual change points.

Chapter 4

Concept Drift Detection Using Computer Vision

Business process dynamics can be used not only to better characterize concept drifts, as demonstrated in the previous chapter, but also to detect them with higher accuracy and greater robustness to noise than state-of-the-art approaches that do not account for process dynamics. By visualizing process dynamics over the full timeframe of an event log, it becomes possible to more effectively relate different changes and assess whether detected variations represent simple or more complex drifts. This chapter explains how this can be achieved, building on our papers "Looking for Change: A Computer Vision Approach for Concept Drift Detection in Process Mining" [7] and "Machine Learning-based Detection of Concept Drifts in Business Processes" [8], both by Alexander Kraus and Han van der Aa.

In the remainder, Section 4.1 introduces the topic. We define the scope of the work in Section 4.2. Our CV4CDD approach, including the input to the problem and desired output, is detailed in Section 4.3. Section 4.4 presents the evaluation of our approach against state-of-the-art techniques. Finally, Section 4.5 reflects on related work before Section 4.6 concludes the chapter.

4.1 Introduction

The importance of concept drift detection in process mining has been widely acknowledged [50], resulting in a range of proposed concept drift detection techniques [60]. However, these existing techniques exhibit notable limitations concerning their accuracy and scope. With respect to their *accuracy*, the performance of existing techniques tends to significantly decline in situations where noise, varying types of drifts, or different levels of change severity are present in an event log.

Such declines occur because the techniques depend on algorithmic design choices, often based on heuristic strategies and assumptions about how drifts manifest in event logs. Since, these assumptions are not applicable to all situations, algorithms based on them can be subject to issues such as a lack of robustness or generally poor accuracy. With respect to their *scope*, existing techniques typically detect only a subset of the commonly-known drift types, typically just focusing on *sudden drifts*, with few techniques also considering *gradual* ones [59]. The detection of more complex drifts, such as *incremental* and *recurring drifts* [31], is generally overlooked and has not yet been tackled in an automated manner. Due to these limitations, existing concept drift detection techniques are thus unable to provide a precise and comprehensive understanding of how processes evolve over time.

To overcome this, we propose CV4CDD, a concept drift detection approach that can detect sudden, gradual, incremental, and recurring drifts in an automated manner, with high accuracy. We achieve this by following an entirely different paradigm for drift detection in process mining. Specifically, unlike existing techniques, our approach uses a machine learning model trained on a large dataset of labeled event logs, enabling it to learn how drifts of different types manifest themselves in event logs. The possibility of training such a model has only recently emerged, thanks to a tool for generating large collections of logs with known concept drifts [12]. However, even with such data, the challenge of applying (supervised) machine learning to concept drift detection is far from straightforward. This difficulty stems from the challenge of capturing the progression of an entire process over time, in a way that it can serve as suitable for input into a machine learning model. To address this challenge, we draw inspiration from research that uses image-based representations to encode multi-faceted data about processes [77, 78]. Therefore, we first turn an event log into an image that visualizes differences in process behavior over time. This enables us to employ a state-of-the-art object detection model [79] (from the field of computer vision), fine-tuned on a large collection of event logs with known concept drifts, to recognize where drifts occur in unseen event logs. Our experiments reveal the efficacy of this idea, showing that our approach significantly improves the state of the art in terms of accuracy, robustness, and automation in detecting drifts, while covering a broader range of drift types.

This chapter extends our earlier work [7], where we introduced a computer vision-based approach for concept drift detection in process mining. We broaden the scope to cover not only sudden and gradual drifts but also incremental and recurring drifts, and we expand the evaluation with further experiments. In addition, we include a sensitivity analysis, a qualitative study, and a benchmark comparison on real-world event logs to provide deeper insights into our approach's performance.

4.2 Problem Scope

Our work addresses the problem of detecting concept drifts in the control-flow perspective of a process based on data recorded in an event log. Conventionally, such concept drifts encompass four types: sudden, gradual, incremental, and recurring drifts [31], as illustrated in Figure 4.1. We divide these four drift types into two groups, as discussed in Chapter 3:

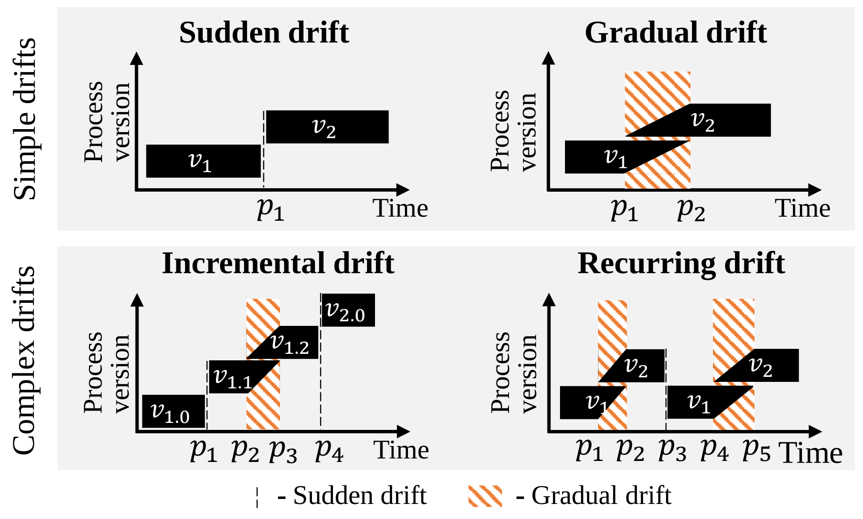


Figure 4.1: Problem scope: detection of sudden, gradual, incremental, and recurring drifts.

Simple drifts. We jointly refer to sudden and gradual drifts as *simple drifts*, since they correspond to a single change in a process from one version to another:

- *Sudden drift:* A sudden drift describes an abrupt replacement of one process version by another. For instance, Figure 4.1 shows the replacement of process version v_1 by version v_2 at a certain moment, i.e., the change point p_1. This means that all process instances that start after p_1 will follow process version v_2. Sudden drifts are often observed in scenarios such as emergency response planning, where airlines and airports may alter their security procedures in response to new regulations [31].
- *Gradual drift:* A gradual drift describes a situation where the replacement of process version v_1 by v_2 involves a transition period in which both versions coexist. In these cases, after an initial change point p_1 an increasing fraction of new process instances will follow version v_2, until the roll-out of

that version is completed at change point p_2. Gradual drifts, for instance, will occur when an organization starts training its employees in a spread-out manner regarding a new way of working, so that more and more employees start following the new version during the training period.

Complex drifts. We jointly refer to incremental and gradual drifts as *complex drifts*, since they consist of several, related simple drifts:

- *Incremental drift:* An incremental drift occurs when one process version is replaced by another through a sequence of simple drifts, rather than at once. For instance, in Figure 4.1, process version $v_{1.0}$ is replaced by $v_{2.0}$ through a sequence of sudden and gradual drifts. This results in two intermediate versions, $v_{1.1}$ and $v_{1.2}$, along with a total of four change points. Generally, the simple drifts that make up an incremental drift correspond to relatively small changes, whereas this also must relate to the same transformation initiative. A notable example of this type of drift are the process changes that arise from agile business process management methodologies [31].
- *Recurring drift:* A recurring drift is characterized by a situation when different versions of a process reoccur in an alternating fashion. For instance, Figure 4.1 illustrates a situation where a process switches between versions v_1 and v_2, following a sequence of sudden and gradual drifts. Recurring drifts are, for instance, common in processes with seasonal patterns.

The aforementioned definitions are consistent with our concept drift characterization taxonomy, presented in Section 3.2.3.

In the following section, we detail our approach that can detect drifts of each of these four types.

4.3 Approach

This section introduces CV4CDD, our computer vision approach for concept drift detection. As visualized in Figure 4.2, our approach consists of two main steps. First, we transform an event log into an image that captures the behavioral (dis)similarity of a process over time. Then, the image is passed to our fine-tuned computer vision model, which identifies if drifts are present in an event log and, if so, determines their type (sudden, gradual, incremental, recurring) and corresponding change points.

Before describing the two steps of our approach in detail, we define the approach's input and desired outcome.

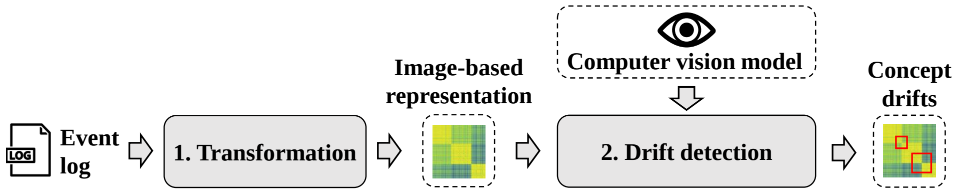


Figure 4.2: Our CV4CDD approach: overview of the main steps.

Input. Our approach takes as input an *event log* L. We define the event log L together with a trace σ, as well as the ordered collection of traces Σ_L, as specified in Section 2.1.2.

Output. Given an event log L as input, the desired output is a collection of drifts $D := \{d_1, \cdots, d_n\}$, where each drift d_i is represented by a tuple, $d_i := (\textit{type}, p_{start}, p_{end})$, with $d_i.\textit{type}$ specifying the drift type, and $d_i.p_{start}$ and $d_i.p_{end}$ denoting the drift's start and end change points, respectively. In case of a sudden drift, the start and end points are the same. For a complex drift, which consists of a sequence of sudden and gradual drifts, the start and end change points are defined by the start change point of the first drift and the end change point of the last drift in the sequence, respectively.

4.3.1 Transformation of Event Log into Image

The first approach step takes as input an event log and transforms it into an image. The image visualizes the behavioral (dis)similarity of a process over time recorded, which can be used to recognize concept drifts. The transformation includes four sub-steps, as depicted in Figure 4.3.

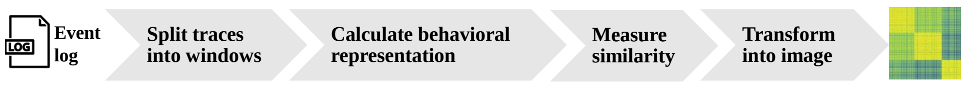

Event log
Split traces into windows
Calculate behavioral representation
Measure similarity
Transform into image

Figure 4.3: First approach step: transforming an event log into an image.

Split traces into windows. The approach first splits the chronologically ordered traces in Σ_L into an ordered collection of N windows, $W := \langle w_1, \ldots, w_N \rangle$. These windows are non-overlapping and collectively cover all recorded traces in Σ_L, with each window w_i containing approximately $|\Sigma_L|/N$ traces.

During the fine-tuning of the computer vision model, we use $N = 200$ as a default value for the number of windows so that each window w_i covers about 0.5%

of all traces from the log. For the training collection (see Table 4.1), with logs containing about 1,000–21,000 traces, this setting provides an effective image-based representation per log for the problem of concept drift detection. We recommend using 200 windows also as the default setting for detecting concept drifts using CV4CDD on a new event log. However, for small event logs (e.g., with fewer than 2,000 traces), decreasing the number of windows avoids having too few traces per window, as demonstrated in our sensitivity analysis in Experiment 1 (see Section 4.4.1). Conversely, larger event logs (especially those spanning a long time range) may benefit from having more windows, to prevent drifts occurring within the span of a single window.

Calculate behavioral representation. After establishing W, our approach computes a *behavioral representation*, $B(w_i)$ for each window w_i, which characterizes the process behavior of w_i's traces. Each $B(w_i)$ consists of a collection of two-dimensional tuples, each storing a behavioral pattern and its frequency, as visualized in Figure 4.4.

A common behavioral representation used in process mining is to capture the directly-follows frequencies observed during a time window [59], which we use as the default for our approach. It counts how often two activities were observed to directly succeed each other for a case. However, it is important to note that the choice for a behavioral representation is flexible, provided that it yields a numeric frequency distribution over a predefined set of relations or patterns across the window. Therefore, CV4CDD can also cover other types of relations (e.g., eventually follows), sets of relation types, such as those of a behavioral profile [35], or declarative constraints [36].

Windows	**Traces**	**Behavioral representation**	**Similarity measure**
w_i	$\langle$a,b,c$\rangle^2$ $\langle$a,c$\rangle$	$B(w_i) = \begin{pmatrix} a \to b : 2 \\ b \to c : 2 \\ a \to c : 1 \\ c \to d : 0 \end{pmatrix}$	$S[i,j] = sim(B(w_i), B(w_j))$
w_j	$\langle$a,b,c$\rangle$ $\langle$a,c,d$\rangle$	$B(w_j) = \begin{pmatrix} a \to b : 1 \\ b \to c : 1 \\ a \to c : 1 \\ c \to d : 1 \end{pmatrix}$	$= cosine(\begin{bmatrix} 2 \\ 2 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix})$ $= 0.83$

Figure 4.4: Illustration of the similarity calculation between two windows.

Measure similarity. Next, our approach compares the behavioral representations obtained for the different windows, quantifying their similarity. This comparison is done for each pair w_i and w_j from W, resulting in a symmetric similarity matrix denoted as S. Each entry $S[i, j]$ in this matrix shows the similarity between the behavior represented by $B(w_i)$ and $B(w_j)$.

The similarity matrix S can be established using various similarity measures. Common options employed in process mining contexts include the cosine similarity (our default choice) and the Earth mover's distance [61]. Figure 4.4 illustrates the calculation of the similarity measure between two windows, using a behavioral representation based on directly-follows frequencies and the cosine similarity measure.

Transform into image. Finally, to enable image-based concept drift detection, the similarity matrix S is transformed into an image. For this transformation, our approach normalizes the matrix values to a range between 0 and 1, where the maximum similarity corresponds to 1 and the minimum similarity is 0. Each normalized value is then scaled by 255 and converted to integers, resulting in a range between 0 and 255. Finally, using the Python Imaging Library[1] and a color map, images are generated from the normalized values.

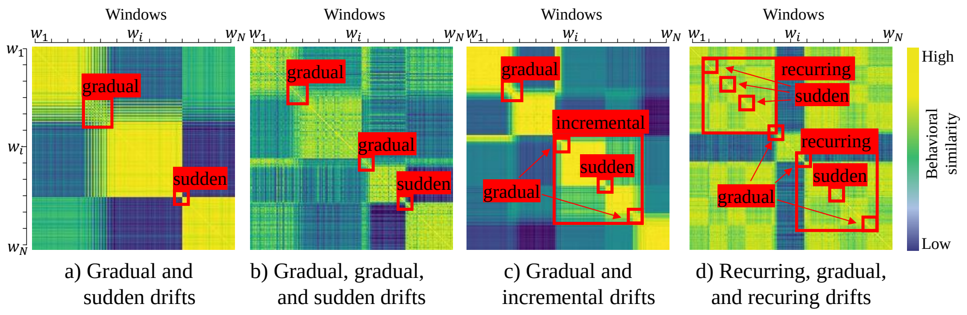


Figure 4.5: Output of the first approach step (incl. annotations).

Figure 4.5 depicts a few examples of the outcome of this step, covering different drift scenarios. Note that the annotation that is shown is covered in Section 4.3.2.

4.3.2 Drift Detection

The second step of CV4CDD takes as input the image obtained from the previous step and applies a fine-tuned object detection model to detect concept drifts. In this section, we present an overview of the object detection task and the employed object detection model, elaborate on the training data and its annotation, and clarify the training configurations.

[1]Available online: `https://python-pillow.org`

Object detection using RetinaNet. Object detection is a fundamental task in computer vision, where the goal is to identify and locate objects within images. Deep learning methods have significantly advanced this field by directly learning features from data, leading to breakthroughs in object detection [79]. *RetinaNet* [80] is a recent addition to deep learning-based object detection models. Known for its effectiveness and reliability, RetinaNet has become widely adopted in both research and practical applications, setting new standards in object detection performance.
RetinaNet Architecture. The RetinaNet model consists of three main components [80]:

- *Backbone Network:* The backbone network identifies patterns necessary for object detection. RetinaNet uses ResNet, a deep convolutional neural network, which improves learning efficiency by allowing inputs to bypass certain layers.

- *Feature Pyramid Network (FPN):* The FPN processes multi-scale feature maps from the backbone to create a pyramid of features at different resolutions. It enhances the model's ability to detect objects of varying sizes by combining high-resolution feature maps from earlier layers with lower-resolution feature maps from deeper layers.

- *Classification and Bounding Box Subnetworks:* These subnetworks enable accurate object detection. The classification subnet predicts the probability of an object belonging to a particular class at each location on the feature map, operating at multiple levels of the FPN. The box regression subnet predicts the coordinates of bounding boxes for objects and refines their positions.

Additionally, RetinaNet uses a specialized loss function, *Focal Loss*, to address class imbalance by down-weighting easily classified background examples and focusing on difficult-to-detect objects. This makes RetinaNet effective in detecting concept drift and handling noise across various sections of an image.
Training data and annotation. We use a training set of event logs with known concept drifts. Each event log in the training set is transformed into an image using the first step of our CV4CDD approach (Section 4.3.1). Then, each image is annotated based on the drift information stored in the gold standard, capturing where different drifts occur and what their types are. For the annotation, we define bounding boxes using the widely-employed COCO (Common Objects in Context) format [81].

In our case, we use these bounding boxes to capture where drifts of certain types occurred in the image, as shown for various scenarios in Figure 4.5. To annotate sudden drifts, characterized by a single change point p that belongs to a window w_i, we establish a bounding box around w_i that spans in total 2 windows in both directions from w_i, resulting in a total length of 5 windows. For gradual drifts, we create annotations using windows that correspond to the start and end change points. Each change point p_{start} and p_{end} is associated with a trace index, which belongs to a particular window in W. The corresponding windows w_i and w_j, allow us to link p_{start} and p_{end} to their window indices i and j. We use these indices to define a bounding box for gradual drift within an image.

Training configurations. To operationalize CV4CDD, we specifically use the RetinaNet model from the TensorFlow Model Garden[2], based on the SpineNet backbone (ID 143). The model is pre-trained on the COCO dataset[3], a widely-used dataset for object detection. To adapt it to our specific task, we fine-tune the model on a training set and halt fine-tuning using a validation set to prevent overfitting.

Image input, batch size, anchor boxes. We use a fixed input size of 256×256 for fine-tuning the model[4] with a batch size of 128 images, taking 312 iterations per epoch. The model undergoes 500 epochs of training, with augmented images to increase diversity and robustness by scaling up to two times or down to one-tenth of their original size. We employ anchor boxes with a 1:1 aspect ratio, concentrating exclusively on square-shaped bounding boxes, since all annotations correspond to squares of different sizes along the diagonal of the image.

Optimization and learning rate. We use stochastic gradient descent with a momentum of 0.9 and clip norm of 10, known for its simplicity and efficiency in training deep learning models, especially with large datasets. Momentum aids convergence by leveraging past gradients, while gradient clipping prevents exploding gradients, ensuring stability, particularly in complex neural networks. Additionally, we employ a cosine learning rate, widely adopted for its simplicity and ability to enhance convergence and generalization. This schedule adjusts the learning rate throughout training, following a cosine-shaped function.

Using the fine-tuned RetinaNet model, our CV4CDD approach can be directly applied to detect concept drifts in unseen event logs.

[2]TensorFlow Model Garden, Available online: `https://github.com/tensorflow/models`

[3]COCO dataset, Available online: `https://cocodataset.org/`

[4]If an input image provided for inference has a different size, i.e., because it was established using a different number of windows N, RetinaNet automatically rescales the image to the default size.

4.4 Evaluation

This section presents three experiments conducted to comprehensively evaluate the performance of our CV4CDD approach for concept drift detection from multiple perspectives. In Experiment 1, we assess the accuracy of our approach in detecting change points in event logs and compare our results to various baseline techniques that address this critical task for concept drift detection. Next, in Experiment 2, we evaluate the accuracy of our approach in detecting drifts and their types and highlight its advantages over a comparable state-of-the-art technique. Finally, in Experiment 3, we apply our approach to real-life event logs and compare the insights obtained with findings from the state-of-the-art technique.

To ensure reproducibility, the data collection, implementation, configurations, and raw results are accessible in our public repository[5].

4.4.1 Experiment 1: Change Point Detection

In this section, we assess the ability of our approach to detect change points in event logs in comparison to existing baselines using synthetic data. We consider this problem in isolation from the detection of drifts, given its fundamental role in concept drift detection, as also evidenced by the various techniques that have been proposed to address it. In the following, we discuss the evaluation setup, obtained results, and findings from a sensitivity analysis.

Evaluation Setup

Below we elaborate on the details of the data collection, baselines, evaluation measures, as well as configurations used to evaluate our approach.
Data collection. Our data collection comprises two datasets, summarized in Table 4.1.
CDLG dataset. To train, validate, and test our drift detection approach, we require a large collection of event logs that contain known (i.e., gold-standard) concept drifts of sudden, gradual, incremental, and recurring types. Since such a collection is not publicly available, we, therefore, generated synthetic datasets using CDLG (Concept Drift Log Generator) [12], a tool for the automated generation of event logs with concept drifts, which comes with a wide range of parameters.
We used CDLG to generate 50,000 event logs, allocating 80% for training, 5% for validation, and 15% testing. The generated event logs have the following characteristics:

[5]Project repository: `https://gitlab.uni-mannheim.de/processanalytics/cv4cdd`.

Table 4.1: Characteristics of the two synthetic datasets.

Characteristics	**CDLG dataset**			**CDRIFT**
(Number of)	**Training**	**Validation**	**Test**	**dataset**
Event logs	40 000	2500	7500	115
→ without drifts	9834	586	1834	0
→ with noise	19 908	1250	3768	60
Drifts	112 660	7069	21 229	156
→ Sudden drifts	41 295	2587	7827	156
→ Gradual drifts	41 395	2615	7776	0
→ Incremental drifts	14 967	986	2823	0
→ Recurring drifts	15 003	881	2803	0
Change points	120 151	7501	22 567	156

- The logs are generated from process trees containing between 6 and 20 activities, as well as sequential, choice, parallel, and loop operators.
- Each event log has between 1,000 and 21,000 traces (with an average of around 7,200 traces per log), and the average trace length varies from 1 to 65 events.
- The event logs have 0 to 3 drifts each (with equal probability). Incremental and recurring drifts consist of 3 simple drifts (of sudden or gradual type), yielding a maximum of 18 change points per log.
- Each drift introduces changes up to 30% of the process tree elements (activities and operators) through deletion, insertion, or swapping.
- A quarter of the logs contain randomly inserted noise in 30% of the traces and another quarter in 60% of the traces. The other half are noise-free.

CDRIFT dataset. To assess the generalizability of our approach and verify that its performance is not restricted to the characteristics of the CDLG dataset, we also consider a dataset used in a recent experimental study [57], which we refer to as the CDRIFT dataset. This set consists of 115 synthetic event logs, previously employed in evaluating various concept drift detection techniques, stemming from three sources [50, 82, 83]. The logs have about 1700 traces on average and contain between 1 and 4 change points. Notably, the CDRIFT dataset only contains sudden drifts, though.

Baselines. We compare our approach to seven techniques for the detection of change points that were used in a recent benchmark study by Adams et al. [57]:

1. BOSE/J by Bose et al. [31] uses non-overlapping and continuous fixed-size windowing with activity pair-based feature extraction and statistical testing.
2. ADWIN/J by Martjushev et al. [54] improves the BOSE/J technique by introducing adaptive windowing using the ADWIN approach.
3. PROGRAPHS by Seeliger et al. [52] implements non-overlapping and continuous adaptive-size windowing, uses graph-based process features alongside Heuristics Miner, and employs statistical testing.
4. PRODRIFT by Maaradji et al. [55] employs non-overlapping, fixed, and adaptive-size windowing, statistical testing, and an oscillation filter.
5. RINV by Zheng et al. [62] uses behavioral profiles, a process similarity measure, and DBSCAN clustering.
6. EMD by Brockhoff et al. [61] employs a sliding window approach with local multi-activity feature extraction and the Earth Mover's Distance.
7. LCDD by Lin et al. [63] uses both static and adaptive sliding windows, incorporates directly-follows relations, and ensures local completeness.

Evaluation measures. We report on results obtained using established evaluation measures for detecting change points [57]. Specifically, for each event log, we compare the sequences of detected $P^d = \langle p_1^d, \ldots, p_n^d \rangle$ and gold-standard $P^g = \langle p_1^g, \ldots, p_m^g \rangle$ change points ($n, m \geq 0$), where each change point is represented by the ordinal number of the first trace that started after the change.

To identify which gold-standard change points have been successfully detected, we use the linear program proposed by Adams et al. [57] to establish a pairwise mapping between the points in P^d and P^g. This program finds an optimal mapping M, assigning as many points to each other as possible, while minimizing the distance between corresponding change points. Note that no point in P^d is assigned to multiple points in P^g or vice versa. Furthermore, M will only include pairs $p_i^d \sim p_j^g$ that are within an acceptable distance from each other, which we refer to as the allowed *latency*. We define latency as a percentage of the total traces in Σ_L, i.e., it must hold that $|p_i^d - p_j^g| \leq |\Sigma_L| * latency$. We report on results obtained using latency levels of 1%, 2.5%, and 5%.

Due to the consideration of latency, each correspondence in M is regarded as a true positive. From this, we derive *precision* (Prc.) as $|M|/|P^d|$, i.e., the fraction of detected change points that are correct according to the gold standard, *recall* (Rec.) as $|M|/|P^g|$, i.e., the fraction of correctly detected gold-standard change points, and the F1-score as the harmonic mean of precision and recall.

Configurations. For different datasets, we use different configurations for the baselines and our approach.

Baselines. When reporting on the performance of the baseline techniques, we use the parameter settings that we found to achieve the highest F1-score. To find these settings for the CDLG data set, we applied the experimental framework by Adams

et al. [57], which assesses different parameter configurations, on the CDLG validation set. For the CDRIFT dataset, we ran experiments using all configurations that are tested in the Adams et al. [57] framework and report on the results obtained using the best parameter settings. The exact parameter settings used for the different techniques per dataset are detailed in our repository.
Our approach. We fine-tuned the RetinaNet model used by our approach with the CDLG training and validation sets and parameters described in the second step of our approach. Given such a fine-tuned model, the only parameter to set for inference is the number of windows N to be used. For the CDLG test set, we use the same number of windows as used during the fine-tuning ($N = 200$). For the CDRIFT dataset, similar to the baseline's parameters, we report on the results obtained for the best parameters ($N = 70$) derived from a sensitivity analysis (see Section 4.4.1).

Results

In the following, we present the results obtained for the two datasets, also focusing on different latency and noise levels.[6]
Accuracy. In the following, we describe the results for each of the two datasets.
CDLG test set. For the CDLG test set, our CV4CDD approach consistently outperforms the baselines, demonstrating F1-scores ranging from 0.80 at 1% latency to 0.83 at 2.5% and 5% latencies. It already reaches its peak performance at just 2.5% latency, surpassing the best baseline, EMD, by 0.27. In terms of recall, our approach outperforms the baseline scores by 0.30 and 0.15 at 1% and 2.5% latencies, respectively. At 5% latencies, our approach achieves also the highest recall of 0.77, however, the LCDD, EMD, and BOSE/J techniques achieve comparable recall scores, each exceeding 0.70. Despite this, they exhibit lower precision, resulting in significantly lower F1-scores. Finally, in terms of precision, our CV4CDD approach surpasses the best-performing baseline, PRODRIFT, by margins of 0.21, 0.15, and 0.13 for 1%, 2.5%, and 5% latency, respectively.
CDRIFT dataset. We obtain overall similar results for the CDRIFT dataset. Our CV4CDD approach outperforms all baselines across latency levels, achieving F1-scores of 0.56, 0.87, and 0.94 at 1%, 2.5%, and 5% latencies, respectively. These higher values can be attributed to the fact that the CDRIFT dataset contains only sudden drifts, which are relatively easier to detect for our approach. Only at 1% latency does PRODRIFT achieve a higher precision of 0.91 compared to 0.61 for

[6]Given the non-determinism involved in training deep learning models, we repeated the training and inference procedure of our approach five times. These repetitions resulted in mean standard deviations of less than 0.1 percentage points across all measures (for CDLG test). We report on the results of the first run in the remainder.

Table 4.2: Overall results for detecting change points.

Dataset	**Technique**	**Latency 1%**			**Latency 2.5%**			**Latency 5%**		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
CDLG (test)	BOSE/J	0.32	0.39	0.25	0.49	0.60	0.54	0.57	0.70	0.63
	ADWIN/J	0.43	0.28	0.34	0.58	0.38	0.46	0.63	0.42	0.50
	PROGRAPHS	0.24	0.26	0.25	0.48	0.52	0.50	0.58	0.64	0.61
	PRODRIFT	0.55	0.22	0.32	0.74	0.30	0.43	0.76	0.31	0.44
	RINV	0.36	0.44	0.40	0.46	0.56	0.51	0.53	0.64	0.58
	EMD	0.36	0.44	0.39	0.51	0.62	0.56	0.58	0.71	0.64
	LCDD	0.27	0.41	0.32	0.39	0.61	0.48	0.46	0.72	0.56
	CV4CDD	**0.86**	**0.74**	**0.80**	**0.89**	**0.77**	**0.83**	**0.89**	**0.77**	**0.83**
CDRIFT	BOSE/J	0.08	0.07	0.07	0.60	0.52	0.56	0.75	0.66	0.70
	ADWIN/J	0.15	0.11	0.13	0.40	0.29	0.34	0.71	0.51	0.59
	PROGRAPHS	0.21	0.18	0.19	0.48	0.41	0.45	0.78	0.67	0.72
	PRODRIFT	**0.91**	0.32	0.48	**1.00**	0.35	0.52	**1.00**	0.35	0.52
	RINV	0.01	0.00	0.00	0.23	0.18	0.20	0.47	0.36	0.41
	EMD	0.05	0.03	0.04	0.88	0.59	0.71	0.97	0.66	0.79
	LCDD	0.00	0.01	0.01	0.02	0.04	0.02	0.24	0.64	0.35
	CV4CDD	0.61	**0.52**	**0.56**	**1.00**	**0.86**	**0.92**	**1.00**	**0.86**	**0.92**

Support for CDLG (test): 22567 change points, CDRIFT: 156 change points.

our approach. The reason for the lower precision of our approach is rather technical. It is mainly attributed to the annotation of sudden drifts using bounding boxes of 5 windows spanning around the position of an actual sudden drift in an image. In scenarios with 70 windows and a latency of just 1%, inaccuracies arise during the transformation from the coordinates of the bounding box to the corresponding window index and subsequently to the first trace within the window, leading to low precision and recall. This is supported by the correctly positioned bounding boxes in the respective images, along with the observation that accuracy sharply increases to its peak values at the next latency of 2.5%.

Noise impact. To evaluate the robustness of our approach, we report the results for the event logs with different noise levels in the CDLG test set (using 5% latency).

As summarized in Table 4.3, our CV4CDD approach maintains consistent performance regardless of noise, achieving the highest F1-scores from 0.82 for logs without noise to 0.81 for logs with 30% and 60% noisy traces. In noise-free conditions, three baselines (PRODRIFT, LCCD, and RINV) come close to our results.

Table 4.3: Noise impact on detecting change points.

Technique	**W/o noise**			**With 30% noise**			**With 60% noise**		
	Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
BOSE/J	0.59	0.70	0.64	0.57	0.70	0.63	0.55	0.71	0.62
ADWIN/J	0.64	0.43	0.51	0.62	0.41	0.50	0.62	0.40	0.49
PROGRAPHS	0.55	0.64	0.59	0.60	0.66	0.63	0.64	0.61	0.63
PRODRIFT	0.76	0.60	0.67	0.67	0.00	0.00	0.33	0.00	0.00
RINV	0.63	**0.83**	0.72	0.39	0.41	0.40	0.40	0.48	0.44
EMD	0.58	0.72	0.64	0.59	0.70	0.64	0.57	0.70	0.62
LCDD	0.63	0.76	0.69	0.35	0.77	0.48	0.36	0.60	0.45
CV4CDD	**0.88**	0.77	**0.82**	**0.88**	**0.76**	**0.81**	**0.88**	**0.75**	**0.81**

Support: 22,567 change points.

The LCCD technique shows an outstanding recall of 0.83, while PRODRIFT maintains its lead across the baselines in precision, also seen for the noise-free CDRIFT dataset. However, all three of these baselines experience a notable decline in accuracy when noise is introduced, particularly PRODRIFT. Conversely, baselines with relatively lower accuracy on the noise-free logs (EMD, ADWIN/J, and PROGRAPHS) demonstrate less vulnerability to noise. This reveals that the baselines are subject to a trade-off between performance in noise-free conditions and robustness to noise, which does not apply to our approach.

Sensitivity Analysis

Finally, we discuss how the number of windows, N, specified by the user affects the performance of our approach. To investigate this, we conduct a sensitivity analysis that examines a range of windows and the corresponding evaluation measures for the two datasets. Considering the average size of the event logs, we analyze windows between 100 and 300 for the CDLG test set and between 60 and 200 for the CDRIFT dataset.

CDLG test set. Figure 4.6 shows the effects of different number of windows on the evaluation measures across the latency levels. All three figures show the same major trend. Specifically, we observe that the F1-score remains within a corridor of $\pm 5\%$pt. for each latency level, with a slight decline in performance at both extremes of the examined range of windows. This indicates that the approach is generally robust to the choice of the number of windows when detecting drifts in the CDLG test set.

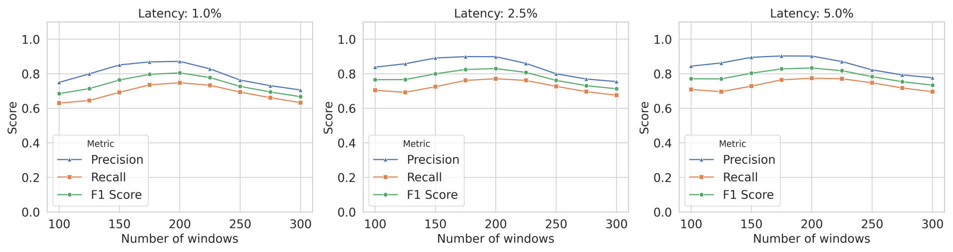


Figure 4.6: Sensitivity analysis (CDLG test set).

CDRIFT dataset. Figure 4.7 illustrates the results of our sensitivity analysis, where we can see two major findings. First, the results suggest that for event logs with relatively few traces (as in the CDRIFT dataset), it is reasonable to reduce the number of windows from the default value of 200 to 100 or fewer, allowing each window to capture more traces and better represent process behavior. Second, a noticeable decline in recall occurs between 100 and 160 windows, with the lowest performance observed at 130 windows. This outcome can be attributed to the structure of the synthetic event logs in the CDRIFT dataset, which typically feature either one change point in the middle of the log (in the majority of logs) or multiple evenly spaced change points. At 130 windows, change points fall near the center of a window, causing the distance from the start of the window (used for the detection of change points) to exceed the allowable latency, resulting in reduced performance.

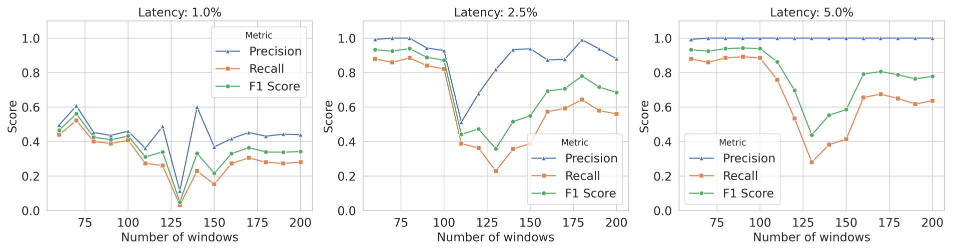


Figure 4.7: Sensitivity analysis (CDRIFT dataset).

Overall, we can observe in this experiment that our approach achieves a notable performance improvement with respect to latency and shows consistent robustness to noise when it comes to the detection of change points in event logs.

4.4.2 Experiment 2: Concept Drift Detection

This section discusses the experiment conducted to evaluate the performance of our approach to detect drift and their types, also in comparison with the state-of-the-art technique. In the following, we discuss the evaluation setup, obtained results, and insights from a sensitivity analysis.

Evaluation Setup

First, we provide information regarding the baseline, data collection, configurations, and evaluation measures used in this experiment.
Baseline. We compare our approach against the Visual Drift Detection (VDD) technique proposed by Yeshchenko at el. [56]. We selected this technique because it stands out as the only existing technique that can be used to detect four types of drifts from an event log and is the only (partially) comparable solution to our approach, due to a lag in automation. In our experiments, we use the online version of the VDD technique[7].
Data collection. We use the CDLG test set to evaluate the accuracy of our approach in detecting concept drift, as no other collections of event logs encompass the necessary drift scenarios that include a mix of different numbers, types of drifts, noise levels, and severities of process changes. However, since the VDD technique is not fully automated and depends on user interpretation of visualizations, we showcase the advantages of our approach using a specific event log from our CDLG test set (log number 5436). The selected log contains 60% noise, 72,433 events, 10,103 traces, 3,787 trace variants, and 9 distinct activities. The left side of Table 4.6 indicates that the log includes two complex drifts: incremental and recurring. Both drifts consist of a sequence of three simple drifts: sudden, gradual, and sudden, leading to a total of 8 change points.
Configurations. For the baseline, we use the suggested default parameters of the online version of the tool for the selected event log: window size 330, slide size 165, cut threshold 300. For our CV4CDD approach, we use the default value of 200 windows.
Evaluation measures. We report on precision, recall, and F1-score by comparing a collection of detected drifts D^d to the gold-standard drifts D^g. For each drift type, t, and the corresponding detected $D^d(t) \subseteq D^d$ and gold-standard drifts $D^g(t) \subseteq D^g$, a true positive (tp) occurs if there is a detected drift $d^d_k \in D^d(t)$ of which both the start and end change points correspond to those of a gold-standard drift $d^g_l \in D^g(t)$ (given a certain latency level). However, if only the start or end

[7]Available online: `https://yesanton.github.io/Process-Drift-Visualization-With-Declare/client/build/`

point of d_l^g is detected correctly, we still count it as 0.5 of a true positive (as well as 0.5 of a false positive). If neither of the detected change points for a drift corresponds to the gold standard, it is considered a false positive (fp). Finally, the number of false negative and true positives ($fn + tp$) is given by the number of actual drifts of a given type.

Given these scores, we compute *precision* (Prc.) as $tp/(tp + fp)$, *recall* (Rec.) as $tp/(fn+tp)$, and the F1-score per drift type (and logs without drifts), as well for the overall detection (using weights to account for their different support values).

Results

In the following, we present the overall results of our approach with respect to different latency and noise levels. Then, we show the advantage of using our approach in comparison to the baseline.

Table 4.4: Concept drift detection results by latency levels.

Drift	**Support**	**Latency 1%**			**Latency 2.5%**			**Latency 5%**		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
No drifts	1834	0.91	1.00	0.95	0.91	1.00	0.95	0.91	1.00	0.95
Sudden	11333	0.99	0.76	0.86	0.99	0.76	0.86	0.99	0.76	0.86
Gradual	11234	0.99	0.60	0.75	1.00	0.62	0.77	1.00	0.62	0.77
Incremental	2823	0.97	0.92	0.95	0.98	0.97	0.98	0.99	0.98	0.98
Recurring	2803	0.99	0.97	0.98	0.99	0.99	0.99	0.99	0.99	0.99
Overall		**0.98**	**0.75**	**0.84**	**0.99**	**0.76**	**0.86**	**0.99**	**0.76**	**0.86**

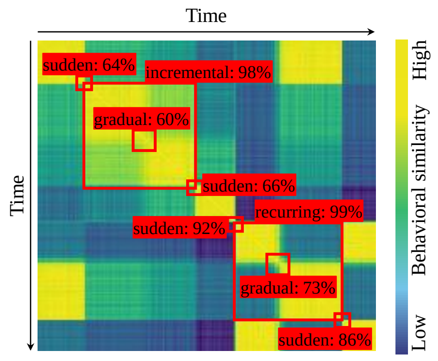
Accuracy. Table 4.4 presents the results obtained for different latency levels. Our CV4CDD approach shows F1-scores ranging from 0.84 at 1% latency to 0.86 at 5% latency. For event logs without any drifts, the approach achieves a perfect recall of 1.00 and a precision of 0.91 across all latency levels. This indicates that it correctly identifies all event logs without drifts. However, in 1 out of 10 cases, the approach incorrectly detects a drift in an event log where no drift exists. For sudden and gradual drifts, precision remains high (above 0.99), but recall drops to 0.75 for sudden drifts and ranges between 0.60 and 0.62 for gradual drifts. This suggests that some actual sudden and gradual drifts, particularly gradual ones, are not detected as such. This can be attributed to the fact that our test set includes event logs with varying noise levels and process changes of different severities, making accurate detection a challenging task. Lastly, for incremental and recurring drifts, the approach achieves results above 0.92 for all measures and latency levels, indicating that it accurately detects the start and end points of these more complex drifts.

Table 4.5: Concept drift detection results by noise levels.

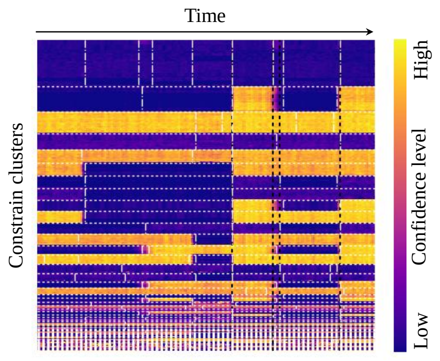
Drift	**Support**	**W/o noise**			**With 30% noise**			**With 60% noise**		
		Prc.	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**	**Prc.**	**Rec.**	**F1**
No drifts	1834	0.91	1.00	0.95	0.92	1.00	0.96	0.90	1.00	0.95
Sudden	11333	0.99	0.76	0.86	0.99	0.77	0.86	0.99	0.75	0.85
Gradual	11234	1.00	0.63	0.77	0.99	0.60	0.75	0.99	0.59	0.74
Incremental	2823	0.98	0.95	0.97	0.99	0.97	0.98	0.98	0.95	0.96
Recurring	2803	0.99	0.98	0.99	0.99	0.98	0.99	0.99	0.98	0.99
Overall		**0.99**	**0.77**	**0.86**	**0.99**	**0.76**	**0.85**	**0.98**	**0.75**	**0.84**

Noise impact. Table 4.5 shows the results for different noise levels. Overall, the results remain consistent across all noise levels, demonstrating that our approach is robust to noise. The only notable variation is a slight decrease in recall for gradual drifts, from 0.63 for noise-free event logs to 0.59 for logs with 60% of noise. This suggests that the relatively low recall observed in Table 4.4 is primarily due to the complexity of the drift scenario rather than the influence of noise.
Comparison with the baseline. Figure 4.8 presents the results of our approach compared to the baseline for the selected event log.
Our approach. Figure 4.8a illustrates the results of our CV4CDD approach, highlighting the detected drifts and their corresponding confidence levels. Table 4.6 shows the corresponding summary of all detected drifts, including their type and respective start and end points. Based on the actual drifts and the deviations between detected and actual change points, our approach successfully identifies all drifts in this event log for a 1% latency, which allows deviations of at most 101 traces, given the log size of 10,103 traces. The average deviation across all change points is approximately 23 traces, which is consistent with the expected deviation[8].
VDD technique. Figure 4.8b presents the primary outcome of the baseline technique: the Drift Map. This map displays over 900 detected behavioral rules (on the y-axis), organized into 61 behavioral clusters, which are indicated by horizontal dashed white lines. The Drift Map highlights four sudden drifts indicated by vertical dashed black lines. While the first and the last drifts are correctly identified, the two other sudden drifts actually mark two change points that denote the start and end moments of a gradual drift. This gradual drift can only be identified through visual inspection of the gradual change in the confidence level in certain behavior

[8]For a log with 10,103 traces divided into 200 windows, each window contains approximately 50 traces. In cases of accurate drift detection, the expected error between the actual trace index and the first trace in the window is half the window size, i.e., 25 traces.



(a) CV4CDD (our approach).



(b) VDD technique (baseline).

Figure 4.8: Drift detection results for the selected log.

clusters. Using the Drift Map, users can also observe that several clusters exhibit recurring behavior in the second part of the log, suggesting that detected drifts belong to a recurring drift. However, analyzing the first part of the log, which contains incremental drifts, proves to be more challenging. Although the Drift Map detects several changes across different behavioral clusters (indicated by a white dashed line within clusters), it becomes nearly impossible to conclude that these changes are part of the same incremental drift.

Table 4.6: Actual drifts vs. detected drifts using our approach.

Actual drifts			**Detected drifts**			**Deviation**	
Type	**Start**	**End**	**Type**	**Start**	**End**	**Start**	**End**
Incremental	1387	4668	(+) Incremental	(+) 1351	(+) 4651	−36	−17
→ Sudden	1387	1387	(+) → Sudden	(+) 1351	(+) 1351	−36	−36
→ Gradual	2986	3463	(+) → Gradual	(+) 2951	(+) 3401	−35	−62
→ Sudden	4668	4668	(+) → Sudden	(+) 4651	(+) 4651	−17	−17
Recurring	5835	9001	(+) Recurring	(+) 5801	(+) 9001	−34	0
→ Sudden	5835	5835	(+) → Sudden	(+) 5801	(+) 5801	−34	−34
→ Gradual	6947	7321	(+) → Gradual	(+) 6951	(+) 7301	4	−20
→ Sudden	9001	9001	(+) → Sudden	(+) 9001	(+) 9001	0	0

“(+)" indicates that the detected information correspond to the actual, given 1% latency.

Sensitivity Analysis

Similar to the first experiment, we conclude this experiment by presenting the insights gained from the sensitivity analysis conducted on the CDLG test set. We consider again different number of windows between 100 and 300 and illustrate the impact of the number of windows on drift detection accuracy across three latency levels.

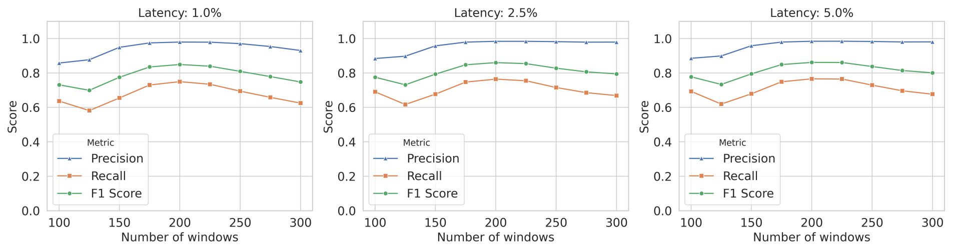


Figure 4.9: Sensitivity analysis (CDLG test set).

Figure 4.9 illustrates how varying the number of windows influences evaluation measures across different latency levels. Similar to the sensitivity results from Experiment 1 (for the CDLG test set), we again observe a slight decline in performance at both extremes of the examined number of windows. However, regardless of latency, the evaluation measures remain within a corridor of $\pm 5\%$pt., indicating that drift detection is also robust to the choice of the number of windows.

In summary, the evaluation results of this experiment suggest that our approach detects various types of drifts with high precision and acceptable recall across different latencies, while remaining robust to noise and choice of the number of windows. Compared to the state-of-the-art techniques, our approach represents a notable advancement towards the automated and thorough detection of concept drifts.

4.4.3 Experiment 3: Evaluation on Real-Life Event Logs

In this experiment, we report results obtained using our approach on real-life event logs and compare them with findings obtained in a comparable study. In the following, we discuss the evaluation setup and obtained results.

Evaluation Setup

We provide a summary of the characteristics of the real-life event logs and detail the configurations used in our approach.
Data collection. To allow a direct comparison of insights, we selected the same three real-life event logs used in the study by Yeshchenko et al. [56], where the authors employed their VDD technique to detect four types of drifts. The characteristics of the selected event logs are summarized in Table 4.7. These logs exhibit diverse characteristics in terms of the number of traces, trace variants (unique sequence of executed activities), number of events, and distinct activities.

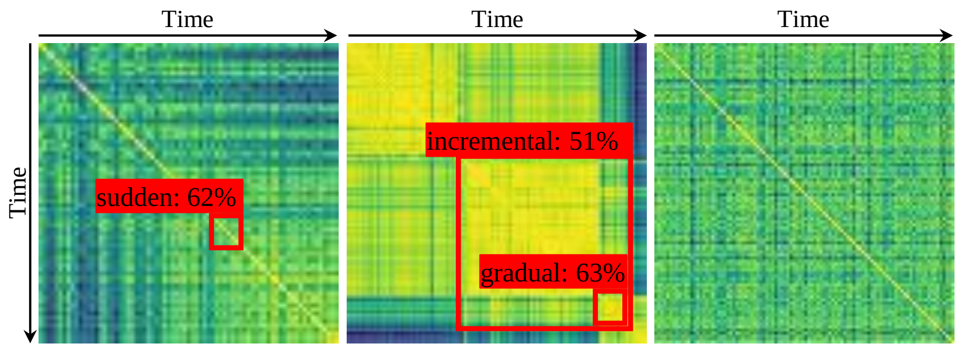
Table 4.7: Characteristics of the real-life event logs.

Log name	**#Traces**	**#Trace variants**	**#Events**	**#Unique activities**
Hospital Log	1,143	981	150,291	624
Help desk Log	4,580	226	21,348	14
Sepsis Log	1,050	846	15,214	16

Configuration. To accommodate the relatively low number of traces in the selected real-life event logs, we applied our approach using an adjusted number of windows. Specifically, we report results using 70 windows for the Hospital and Sepsis logs (which contain around 1,000 traces) and 100 windows for the Helpdesk event log (which has approximately 4,500 traces).

Results

Figure 4.10 presents the results obtained by our approach for the real-life event logs, which we compare with the findings reported using the VDD technique [56].
Hospital Log. For the Hospital event log, our method identifies a single sudden drift around June 12, 2006. This detection partially aligns with the second sudden drift reported by the VDD technique (around July 07, 2006) [56]. However, the Drift Map identified an additional sudden drift in November 2005, which our approach does not capture. Lastly, neither our approach nor the VDD tool detected any complex drifts within the log.
Help Desk Log. In the case of the Help desk event log, our approach identifies an incremental drift that begins on September 5, 2012, and concludes on July 10, 2013. Within this period, we also detect a gradual drift between June 8, 2011, and August 9, 2013, as part of the incremental drift process. The start points of these two drifts correspond to the sudden drifts detected by the VDD technique. However, according to our findings, these drifts represent change points within the detected incremental drift. For certain behavioral clusters, the VDD technique



a) Hospital log b) Help desk log c) Sepsis log

Figure 4.10: Detected drifts by our CV4CDD approach.

also detects gradual and incremental drifts, which are temporally aligned with the two drifts identified by our approach. Additionally, the VDD tool detects several sudden drifts in some clusters, though, as noted by the authors, these are considered outliers rather than true drifts. Our image-based representation of the event logs depicts these outliers, though our approach does not classify them as drifts.

Sepsis Log. Our approach does not detect any notable drifts in the Sepsis event log. As shown in Figure 4.10, the process behavior remains relatively homogeneous throughout the entire recorded period. Similarly, the VDD technique does not show any patterns of change over time, indicating that the major clusters of behavior do not exhibit significant drifts [56]. However, the authors identify some recurring patterns in a few minor clusters, which are associated with two specific activities. While these patterns may indeed suggest recurring drifts, they could also be attributed to the specific cases within these minor clusters, as they represent only a small portion of the overall behavior.

Overall, this experiment demonstrates that our CV4CDD approach can be effectively applied to real-life event logs, providing insights that align with and also extend the findings of the state-of-the-art technique. However, our approach automatically identifies when drifts occur and what their types are without the need of any additional interpretation of different visualizations.

4.5 Related Work

Since establishing the problem and importance of concept drift detection in process mining more than a decade ago [50], various techniques have been proposed to address this problem, as highlighted in recent literature reviews [59, 60]. In this section, we discuss such existing techniques that take event logs as input to identify drifts and their types.

Table 4.8: Classification of different concept drift detection techniques.

Technique	**Change point detection**	**Drift type detection**			
		Sudden	**Gradual**	**Incr.**	**Rec.**
Various works	✷				
Bose et al. [31]	✷	(✷)	(✷)		
Martushev et al. [54]	✷	(✷)	(✷)		
Maaradji et al. [55]	✷	✷	✷		
Yeshchenko at el. [56]	✷	✷	(✷)	(✷)	(✷)
Our approach (CV4CDD)	✷	✷	✷	✷	✷

Legend: "✷" - automated, "(✷)" - semi-automated.

Change point detection. Table 4.8 provides an overview of existing concept drift detection techniques, including their scope and automation level. As shown, the majority of existing techniques [52, 61–65, 67], as shown in the first row of the table, focus on detecting change points in an event log since this it is a required first step in concept drift detection. They use a wide range of ways to tackle this task, including statistical testing, various kinds of feature representations, windowing strategies, and clustering. Many of the techniques achieve good results, as demonstrated in a recent evaluation framework [57]. However, our evaluation experiments demonstrate that our proposed CV4CDD approach outperforms them in terms of accuracy and robustness to noise when detecting change points. Furthermore, these techniques do not go beyond the detection of change points, meaning that they are only able to recognize *when* a process's behavior significantly changed, but they do not provide insights into *what* type of drift actually occurred. As a result, they do not provide information about how process evolved over time.

Drift type detection. As shown in Table 4.8, a few techniques [31, 54, 55] are able to detect drifts and their types (though often only semi-automatically).

Bose et al. [31] introduced concept drift detection in process mining, presenting a method for automatically detecting sudden and gradual drifts using statistical testing of feature vectors. Although this approach is pioneering, it has limitations in automated drift detection. Users must specify the type of drift in advance and are required to manually select features, relying on prior knowledge of drift characteristics. Otherwise, they may face the computational burden of testing all feature combinations. Martushev et al. [54] enhance this method by introducing adaptive windowing, which allows a more accurate detection of drifts. However, users still need to define the type of drift beforehand, and both techniques remain limited to detecting only sudden and gradual drifts, neglecting recurring and incremental drifts.

Maaradji et al. [55] introduced an alternative technique (ProDrift) for sudden and gradual drift detection, addressing the main limitation of Bose's method. Their approach provides an automated and statistically grounded solution for identifying both sudden and gradual drifts, representing the current state of the art in this area. However, a significant drawback arises in event logs containing noise, where detection accuracy notably decreases, as shown in our evaluation.

The Visual Drift Detection (VDD) technique, introduced by Yeshchenko et al. [56], is a stand-alone solution for detecting four types of concept drifts in event logs. This technique leverages concepts such as temporal logic, DECLARE constraints [76], and time series analysis. To identify different types of drifts, it provides visual aids, including Drift Maps, Drift Charts, and directly-follows graphs. However, the identification of gradual, recurring, and incremental drifts remains a manual process that relies on user interpretation. As a result, recognizing drifts and their types within the same event log can be challenging and subjective.

Overall, it is evident that the comprehensive detection and characterization of drifts in event logs has not been adequately addressed. The approach presented in this chapter overcomes existing limitations and advances the state of the art by offering a reliable and automated method for identifying all four types of drifts. This innovative paradigm uses machine learning to learn how drifts manifest in event logs, moving beyond traditional hand-crafted unsupervised techniques.

4.6 Conclusion

In this chapter, we proposed CV4CDD, a concept drift detection approach that can detect sudden, gradual, incremental, and recurring drifts in an automated manner. It is based on a novel idea to detect drifts in an event log using an object detection model (RetinaNet) fine-tuned with a large collection of event logs that contain known concept drifts. In the conducted experiments, we demonstrated that CV4CDD considerably outperforms available baselines for detecting change points in event logs across several datasets, including well-established datasets commonly used to evaluate concept drift detection techniques. We also demonstrated the accuracy in detecting all four types of drifts and the robustness of our approach to noise. Additionally, we highlighted its advantages in performing qualitative analysis on various real-world event logs compared to the state of the art. Finally, it is worth noting that CV4CDD stands out not only as the first approach using techniques from computer vision for concept drift detection in process mining, but as the first approach using supervised machine learning in general.

In future work, we aim to address the limitations of our approach and enhance its capabilities. We plan to refine the annotation of sudden drifts, as the current bounding box of 5 windows may be too large for small event logs. Additionally, our goal is to develop an algorithm to determine the optimal number of windows based on event log characteristics, removing the need for user selection. We also intend to train our model using diverse data sources, moving beyond our current reliance on a single tool for generating event logs with known concept drifts. We will enhance our capabilities in concept drift detection to encompass multiple process perspectives, including time, resources, and data. Additionally, we target including drift localization to gain insights into the changes that occur after each drift.

Chapter 5

Business Process Steady-State Detection

In the previous chapters, we addressed the problem of concept drift detection and characterization, where the use of process dynamics proved beneficial. In this chapter, we shift the focus to the process dynamics itself. Any dynamic system alternates between phases of relative stability and periods of instability. In the context of business processes, detecting such dynamics is a non-trivial task with considerable relevance for many process mining activities. This chapter emphasizes the importance of steady-state detection in process mining and explains how steady and non-steady states of a business process can be identified. It builds on our paper "On the Use of Steady-State Detection for Process Mining: Achieving More Accurate Insights" by Alexander Kraus, Keyvan Amiri Elyasi, and Han van der Aa [10].

The remainder of this chapter is organized as follows. Section 5.1 introduces the topic. Section 5.2 provides background information and illustrates the importance of SSD in process mining. Then, we present our framework for SSD for business processes in Section 5.3. In Section 5.4, we present the results of our evaluation experiments, demonstrating the framework's accuracy and usefulness. Finally, Section 5.5 discusses the relationship between SSD and other related problems in process mining, while Section 5.6 summarizes our findings and suggests potential directions for future work.

5.1 Introduction

Business processes are often supported by information systems that record execution data in event logs, which are then used in process mining to extract data-driven insights [2]. However, these event logs often capture business processes executed in both steady and non-steady states. *Steady states* refer to periods when process behavior remains stable and consistent over time [84], while *non-steady states* are marked by fluctuations and irregularities due to the dynamic environments in which processes operate. These non-steady states can arise from factors such as increased case arrivals during peak seasons or reduced resource availability during holidays, causing the process to deviate from its usual operations and performance levels.

The distinction between steady and non-steady states of processes is crucial for various process mining tasks. As shown later in this chapter, failing to distinguish between such states can, for instance, distort performance insights obtained through lead-time analysis or hurt the accuracy of predictive process monitoring models. Recognizing the impact that state fluctuations have in dynamic environments, the task of *steady-state detection* (SSD) aims to identify periods when a system operates in a steady state (or when it does not). Various techniques for this task have already been developed and tested in different application contexts, such as industrial systems [85] and signal processing [86]. However, their application in process mining has been largely overlooked so far, despite the potential of SSD to improve the accuracy of process mining insights.

Therefore, this chapter highlights the importance of SSD in process mining and investigates the applicability of existing SSD solutions within this domain. To operationalize this, we propose a framework designed to identify steady states in business processes based on event data. The framework consists of two steps: (1) extracting time series from an event log that capture the progression of relevant process characteristics and (2) applying an established SSD technique to detect steady and non-steady states per process characteristic and aggregating these results to detect steady states at the process level. The effectiveness of our framework is evaluated in two experiments: one assessing its accuracy in a controlled environment based on simulated event logs and the other demonstrating its practical benefits in a downstream process mining task, specifically for remaining time prediction. Our findings showcase that our framework indeed enables the use of SSD for process mining and highlight the potential of SSD to provide more accurate insights into the operations of organizations.

5.2 Background and Problem Illustration

In this section, we provide background information on steady states and demonstrate the importance of their consideration in process mining.

Steady states and their properties. A steady state refers to a condition in which the behavior of a system remains constant over time [84]. The analysis of steady

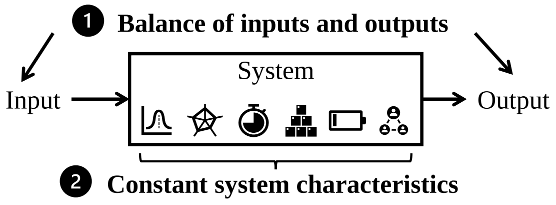

1 Balance of inputs and outputs
System
Input
Output
2 Constant system characteristics

Figure 5.1: Two key properties of a steady state.

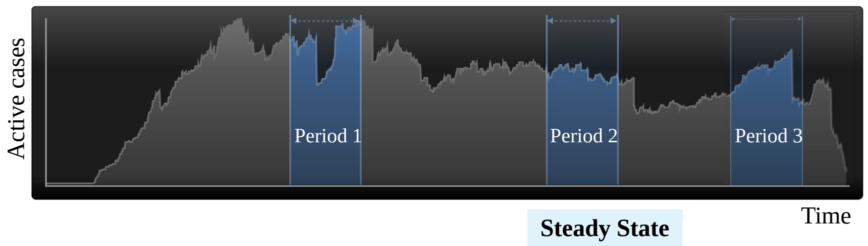
states has a long-standing history and has demonstrated its significance across various fields, including thermodynamics [87], biology [88], ecology [89], and economics [90]. As visualized in Figure 5.1, any system (including a business process) in a steady state can be characterized by the following two properties:

1. *Balance of system's inputs/outputs*: A system in a steady state maintains a balance between input and output rates, ensuring that no measurable quantity accumulates or depletes over time. In the context of a business process, this means, e.g., that the number of incoming and completed cases remains consistent over time.
2. *Constant system's characteristics*: The characteristics of a system in a steady state remain constant, without significant fluctuations. For a business process, this could mean that, e.g., the number of active cases and available resources remain stable.

Given these properties, the system's behavior becomes predictable, allowing for more precise and meaningful analysis.

Importance of SSD in process mining. To illustrate the importance of SSD in process mining, we examine how the performance of a business process, measured by average and median lead times, can differ between steady and non-steady periods, and the implications this may have on a downstream process mining task. For this purpose, we use a real-life event log describing a permit application process at a municipality (BPIC2015-2) [91] as a running example. The event log contains 44,354 events, capturing the execution of 832 cases over a period of approximately 5 years. During this period, the process exhibits an average lead time of 22.9 weeks, with a median lead time of 15.5 weeks.

The number of active cases, shown in Figure 5.2, indicates that the process was not steady throughout the recorded timeframe, with both stable and unstable periods. For instance, in Period 1, spanning 5 months and involving 23 cases, the process shows instability, marked by a significant drop in active cases. The average lead time is 10.7 weeks, with a median of 11.3 weeks. Period 2, also 5 months long with 23 cases, is more stable, with fewer fluctuations in active cases. The average



			Steady State	
Measure	**Event log**	**Period 1**	**Period 2**	**Period 3**
Number of traces	832	23	23	25
Duration	56 months	5 months	5 months	5 months
Average lead time	22.9 weeks	10.7 weeks	5.6 weeks	12.1 weeks
Median lead time	15.5 weeks	11.3 weeks	3.6 weeks	13.9 weeks

Figure 5.2: Comparison of process performance between different periods.

lead time is 5.6 weeks, and the median is 3.6 weeks. Lastly, Period 3 exhibits a rise and fall in active cases, indicating a non-steady state. It has an average lead time of 12.1 weeks and a median lead time of 13.9 weeks.

Performance in the steady state (Period 2) is nearly twice as good as in the other periods and about four times better than the overall average across all recorded cases. Such differences are particularly relevant for process mining tasks such as remaining time prediction, as demonstrated in our evaluation. Specifically, when significant performance differences exist between steady and non-steady states, it may be beneficial to use SSD as a bucketing method to split the event log into sublogs representing steady and non-steady states. Separate models can then be trained for each sublog, allowing the appropriate model to be applied based on whether the process is currently in a steady or non-steady state, improving the accuracy of predictions for ongoing cases.

5.3 Approach

This section describes our proposed SSD framework. As illustrated in Figure 5.3, the framework takes an event log as input and then extracts time series that represent the progression of relevant process characteristics over time. These time series are subsequently analyzed using existing SSD techniques to identify steady states. Finally, the framework produces as output a sublog of traces that belong to the detected steady states. In the following, we describe these two main steps.

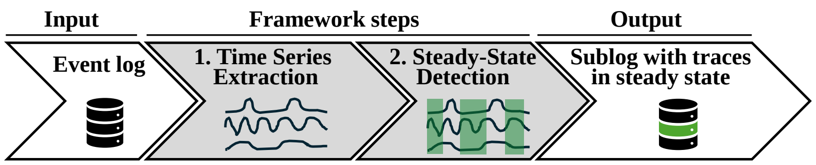

Input
Framework steps
Output
Event log
1. Time Series Extraction
2. Steady-State Detection
Sublog with traces in steady state

Figure 5.3: Overview of the main steps of our framework.

5.3.1 Time Series Extraction

In Step 1, we generate time series from an event log to capture the evolution of process characteristics relevant to SSD. Such transformations are widely used in process mining, for purposes including business process simulation [92], assessing process resilience [11], and evaluating process complexity [93]. Below, we outline the specifics of this step.

Input. Our approach takes an *event log* as input. We define an event log L, a trace σ, the ordered collection of traces Σ_L, and an event e as specified in Section 2.1.2.

Windowing. We divide the entire timeframe of an event log L into $n \in \mathbb{N}$ equally spaced *time windows* $W_l = \langle w_1, \ldots, w_n \rangle$, each with a fixed length l (e.g., a day or a week). Consequently, each event $e \in L$ is assigned to exactly one time window w_t, where $t \in \{1, \ldots, n\}$.

Time series construction. Next, we construct time series over $w_t \in W_l$ for a number of relevant process characteristics. In our framework, we consider 3 process characteristics that are relevant for SSD and can be derived from a standard event log L: *the number of active cases* (ac), *the number of completed cases* (cc), and *the*

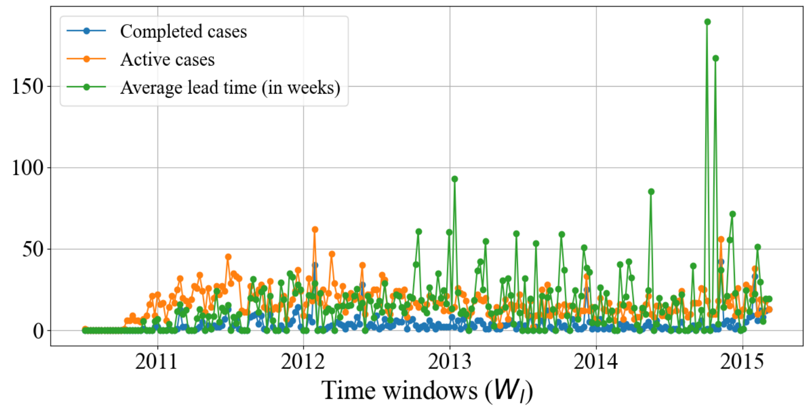


Figure 5.4: Outcome of the first framework step.

average lead time (alt) of completed cases during a time window w_t. If the event log contains further information, such as resource details, additional process characteristics can be incorporated to provide a more comprehensive representation of the process.

We use $y^f_{w_t} \in \mathbb{R}$ to denote the value of a characteristic (or *feature)* $f \in F = \{ac, cc, alt\}$ during a time window w_t. For each feature, we concatenate these values into a *time series* $\{y^f_{w_t}\}^n_{t=1}$, which captures the evolution of f over the time windows in W_l. Figure 5.4 shows the outcome of this step with weekly windowing for the BPIC2015-2 event log, serving as a running example.

5.3.2 Steady-State Detection

After extracting time series that represent relevant process characteristics, the next step is to identify steady states and their associated traces. This involves performing SSD at the time series level (i.e., per characteristic), then determining steady states at the process level, and finally identifying the relevant traces.

SSD at time series level. For each time series $\{y^f_{w_t}\}^n_{t=1}$, we derive a corresponding *binary time series* $\{p^f_{w_t}\}^n_{t=1}$, with $p^f_{w_t} \in \{0, 1\}$ for each time window w_t using an existing SSD technique. This binary time series indicates whether the corresponding process characteristic is in a steady state during w_t, where $p^f_{w_t} = 1$ signifies a steady state and $p^f_{w_t} = 0$ indicates a non-steady state.

To obtain $\{p^f_{w_t}\}^n_{t=1}$, we can use an SSD technique from a range of existing ones. Our framework's implementation currently supports the following options:

- *Rolling Window (RW) [94]*: The RW technique detects steady states in a time series by comparing the short-term and long-term rolling averages of its values. It identifies a drift when the deviation between the short-term and long-term averages exceeds a threshold that is scaled by the standard deviation of the long-term average.
- *Cumulative Sum (CS) [95]*: The CS algorithm monitors cumulative increases and decreases in the data and flags a change when these values exceed a predefined threshold. Once a change is identified, the cumulative calculation resets to ensure continued monitoring.
- *Variance Filter (VR) [96]*: The VR method proposed by Rhinehart uses a variance filter to distinguish between steady and non-steady states based on statistical analysis. It applies a filter that evaluates the ratio of the variance of the signal, with thresholds used to identify steady states.
- *ED Pelt with Transitions (EDP) [97]*: The EDP technique identifies steady states in a time series by splitting the time series into "statistically homogeneous" segments using the pruned exact linear time (Pelt) change point de-

tection algorithm. The Pelt method guarantees optimal segmentation while maintaining a linear computational complexity.

Beyond these techniques, our framework is compatible with any SSD method that accepts a real-valued time series and generates a binary time series.

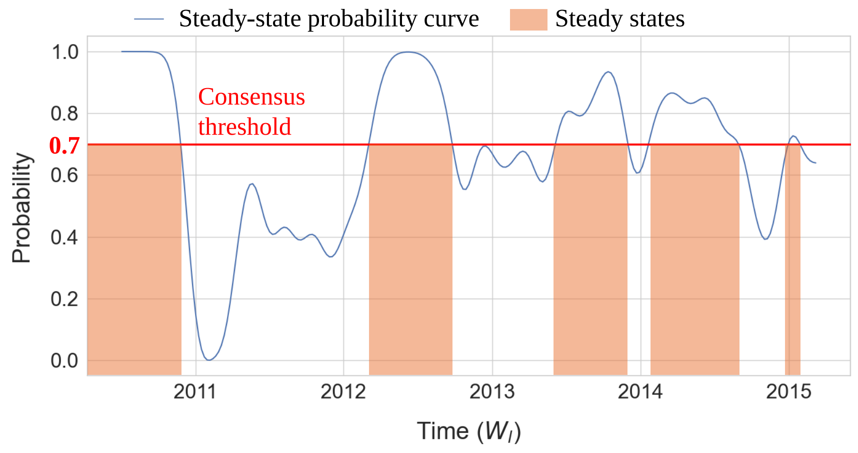


Figure 5.5: SSD using the probability curve and consensus threshold.

SSD at process level. After performing SSD per process characteristic, we next aggregate the information from the binary time series to determine if indeed the entire process can be considered to be in a steady state during a given time window. Our framework supports several aggregation techniques for this:

Kernel-based aggregation computes a *steady-state probability curve* as a time series $\{P_{w_t}\}_{t=1}^{n}$ with $P_{w_t} \in [0,1], \forall w_t$ that represents the likelihood of a time window w_t to record a steady state of a process. To do this, we first aggregate insights from different process characteristics by calculating the average value across all binary time series $\{p_{w_t}^{f}\}_{t=1}^{n}$ for each time window. We then apply a Gaussian filter [98] with a kernel of 4 standard deviations to smooth the curve and reduce fluctuations. After smoothing, the time series is rescaled using Min-Max normalization to ensure that the values lie between 0 and 1. Finally, to identify the steady states of a business process, we compare the values of the steady-state probability curve with a *consensus threshold* $\tau \in [0,1]$. If $P_{w_t} \geq \tau$, the time window w_t is considered to be part of a steady state; otherwise, as non-steady. Figure 5.5 illustrates the outcome of this transformation for the time series shown in Figure 5.4, assuming a consensus threshold $\tau = 0.7$.

In addition to the kernel-based aggregation technique, our framework also supports more straightforward aggregation techniques. *Consensus-based aggregation* considers a time window w_t as a steady state if $p_{w_t}^{f} = 1$ for all process characteristics. *Majority-based aggregation* deems a time window w_t as a steady state if $p^f_{w_t} = 1$ holds for at least 50% of the process characteristics. Finally, *single-source aggregation* classifies a time window w_t as a steady state if $p^f_{w_t} = 1$ holds for at least one process characteristic f.

Detection of steady-state traces. Finally, after identifying the steady states, we identify traces that correspond to the steady states. To do this, we analyze each trace $\sigma \in \Sigma_L$ and check whether the timestamps of its events fall within the identified steady states. If the proportion of such events relative to the total number of events in σ exceeds a predefined *trace acceptance threshold* $\theta \in [0, 1]$, the trace is classified as part of a steady state. As a result, we obtain a sublog $\Sigma^S_L \subseteq \Sigma_L$, which consists of traces associated with the steady states of the business process.

5.4 Evaluation

This section presents two conducted evaluation experiments. In the first experiment, detailed in Section 5.4.1, we evaluate the accuracy of our framework at detecting steady states using synthetic data. The second experiment, explained in Section 5.4.2, demonstrates the usefulness of the framework using real-life event logs and a concrete process mining task, i.e., the prediction of the remaining time for ongoing cases. To ensure reproducibility, we have provided the data, implementation details, configurations, and raw results in a publicly accessible repository[1].

5.4.1 Experiment 1: Accuracy

In the first experiment, we assess the ability of our framework to identify steady states in event logs. In the following, we discuss the data collection, setup, evaluation measure, and obtained results.

Data collection. In this experiment, we generate data by simulating an order-to-cash process for a medium-sized company, as described in the work by Zahoransky et al. [99]. The simulation model is built using the CIW library [100], an open-source tool for discrete event simulation.[2] To introduce steady and non-steady states, we vary the number of incoming cases during the simulation, ensuring a balanced distribution between steady and non-steady periods. Specifically, we create non-steady states by applying periods of linear increases and decreases in the arrival rate, followed by periods of constant arrival rate to establish steady states. We implement up to 5 changes in the arrival rates, starting with either increases or decreases, resulting in 10 distinct scenarios, as shown in Figure 5.6. To ensure

[1]Project repository: `https://gitlab.uni-mannheim.de/processanalytics/ssd`.

[2]Available online: `https://ciw.readthedocs.io/en/latest/index.html`

robust evaluation, we generate 10 event logs for each scenario, producing a total of 100 event logs.

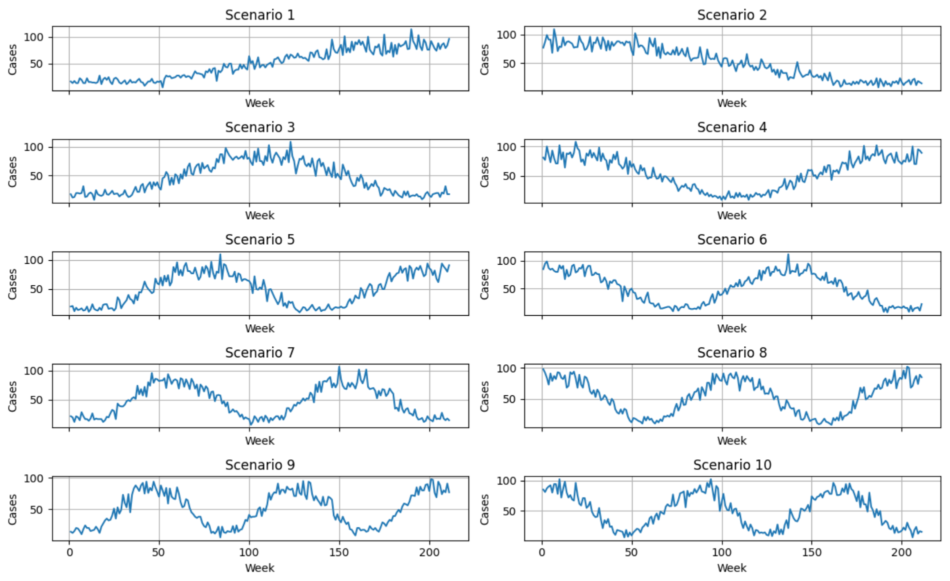


Figure 5.6: Simulated number of arrived cases for each scenario.

Setup. In Step 1 of our framework, we apply weekly windowing and consider 3 process characteristics: the number of active cases, the number of completed cases, and the average lead time of completed cases, i.e., $f \in \{\text{ac}, \text{cc}, \text{alt}\}$.

In Step 2, we evaluate all four implemented techniques for SSD: Rolling Window (RW), Cumulative Sum (CS), Variance Filter (VR), and ED Pelt with Transitions (EDP). For each technique, we test a variety of parameter combinations,[3] resulting in a total of 564 evaluations per event log. To detect steady states of the process, we evaluate 4 aggregation techniques (i.e., aggregation-based SSD) with a trace acceptance threshold of $\theta = 0.8$: kernel-based, consensus-based, majority-based, and single-source. For the kernel-based approach, we set the consensus threshold to $\tau = 0.7$. Additionally, we compare the results of our framework when the decision about steady states is made based solely on a single process characteristic (i.e., feature-based SSD).

Evaluation measure. To measure our framework's accuracy in classifying each time window as a steady or non-steady state, we use the ϕ coefficient [101], a widely used binary classification metric for assessing the strength of observed as-

[3]The exact parameters tested for each technique are specified in our repository.

sociations. It provides a balanced evaluation by considering all elements of the confusion matrix and is defined as follows:

$$\phi = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}},$$

where TP, TN, FP, and FN represent true positives, true negatives, false positives, and false negatives, respectively. TP occurs when the predicted steady-state time window matches the gold standard, while FP occurs when it does not. TN and FN follow similarly for time windows that denote non-steady states. The ϕ coefficient ranges from -1 to +1, where +1 indicates perfect classification, 0 indicates random guessing, and -1 indicates complete disagreement. The closer the metric is to 1, the better.

Table 5.1: Results of Experiment 1: The average ϕ coefficient along with its standard deviation.

SSD configuration	**SSD technique**			
	RW	**CS**	**VF**	**EDP**
Feature-based				
Active cases	0.35 ± 0.03	0.27 ± 0.08	0.19 ± 0.06	**0.43 ± 0.12**
Avg. lead time	0.21 ± 0.06	**0.38 ± 0.12**	0.37 ± 0.10	0.04 ± 0.07
Case completions	0.36 ± 0.03	0.26 ± 0.05	0.25 ± 0.06	**0.43 ± 0.08**
Aggregation-based				
Kernel-based	**0.47 ± 0.04**	0.33 ± 0.06	0.32 ± 0.10	0.35 ± 0.09
Consensus-based	0.32 ± 0.03	0.25 ± 0.05	0.32 ± 0.13	**0.35 ± 0.09**
Majority-based	0.34 ± 0.03	0.30 ± 0.06	0.18 ± 0.05	**0.42 ± 0.08**
Single-source	0.25 ± 0.04	**0.40 ± 0.10**	0.20 ± 0.07	0.07 ± 0.09

Note: The highlighted values show the best results in each row.

Results. Table 5.1 presents the results obtained on our data collection, showing the average and standard deviations of the ϕ coefficient for both feature-based and aggregation-based configurations. The table shows that the SSD technique using rolling windows (RW) achieves the highest ϕ coefficient of 0.47 with kernel-based aggregation, indicating a moderate positive association between the predicted steady and non-steady states. This result is comparable to results observed in other domains [102], though slightly lower. The EDP technique produces similar outcomes, with a ϕ coefficient around 0.43 when using either the number of active cases or the number of case completions. In contrast, the VF technique demonstrates the lowest performance, consistently underperforming relative to other techniques across all configurations.

Overall, existing SSD techniques can be applied in process mining, but the evaluation indicates room for improvement due to the unique properties and interrelations of process characteristics, which differ from other domains, thus requiring domain-specific adjustments to SSD to enhance accuracy and applicability.

5.4.2 Experiment 2: Usefulness

In this experiment, we demonstrate the usefulness of our SSD framework by considering a well-known task in process mining, namely the remaining time prediction problem. Specifically, we compare the prediction accuracy of various state-of-the-art approaches applied to entire event logs with their accuracy when using only data from steady states.

In the following, we discuss the data collection, setup, and obtained results.

Table 5.2: Characteristics of the employed event logs.

Event log	**Number of**				**Case length**		**Case duration**	
	Cases	**Variants**	**Events**	**E. classes**	**Avg.**	**Max**	**Avg.**	**Max**
Steady and non-steady states (Σ_L)								
Hospital	100 000	1020	451 359	18	4.5	217	127.2	1035
Sepsis	1050	846	15 214	16	14.5	185	28.5	422
Helpdesk	4580	226	21 348	14	4.7	15	40.9	60
BPIC12	13 087	4366	262 200	36	20.0	175	8.6	137
BPIC15-1	1199	1170	52 217	398	43.6	101	95.9	1486
BPIC15-2	832	828	44 354	410	53.3	131	160.3	1326
BPIC15-3	1409	1349	59 681	383	42.4	123	62.2	1512
BPIC15-4	1053	1049	47 293	356	44.9	115	116.9	927
BPIC15-5	1156	1153	59 083	389	51.1	153	98.0	1344
Steady states (Σ_L^S)								
Hospital	8315	176	27 117	15	3.3	217	54.3	773
Sepsis	439	378	6242	16	14.2	170	35.8	422
Helpdesk	745	92	3742	10	5.0	14	40.4	60
BPIC12	5692	1417	81 125	36	14.2	142	5.0	67
BPIC15-1	682	667	29 956	377	43.9	93	99.8	1486
BPIC15-2	311	310	17 823	341	57.3	132	152.9	1171
BPIC15-3	521	505	22 363	303	42.9	101	58.3	1261
BPIC15-4	677	674	30 813	321	45.5	116	104.9	831
BPIC15-5	520	519	27 462	329	52.8	108	86.9	812

Note: The attributes case duration is in days.

Data collection. Our data collection consists of 9 publicly available real-life event logs that are commonly used for predicting the remaining runtime of ongoing cases.[4] As summarized in Table 5.2, these logs represent the execution of various processes and display diverse characteristics across multiple dimensions, including the number of cases, variants (i.e., unique traces), recorded events, event classes (i.e., unique activities), average case lengths and durations. Furthermore, we include the characteristics of the sublogs with traces that correspond to steady states (Σ_L^S), as identified using our framework.

Setup. Next, we discuss the framework configurations, employed data split, and used remaining time prediction approaches.

Configurations. In Step 1 of our framework, we apply weekly windowing for all event logs, except for the BPIC12 event log, which covers a relatively short time period. For this log, we use daily windowing instead. We again consider 3 process characteristics, i.e., $f \in \{\mathrm{ac}, \mathrm{cc}, \mathrm{alt}\}$. In Step 2, we use the configuration that yielded the best results in Experiment 1, specifically the rolling window (RW) and kernel-based aggregation with a consensus threshold of $\tau = 0.7$ and a trace acceptance threshold of $\theta = 0.8$.

Data split and prefix generation. We use a 64%-16%-20% chronological holdout split that divides data into training, validation, and testing sets while preserving the natural chronological order. This method mitigates data leakage and simulates real-world scenarios where predictions are made based on historical data [103]. For each trace σ in a split, we extract all prefixes between lengths 2 and $|\sigma| - 1$ to establish prediction problems.

Approaches. We consider 3 remaining time prediction approaches that estimate the remaining time of an ongoing case based on the sequence of already executed activities (and possibly other available attributes):

- DUMMY: A simple baseline that predicts the remaining time of an ongoing case by averaging the remaining time of training cases that share the same sequence of executed activities.
- DALSTM: This deep learning model, based on the LSTM architecture, outperforms other LSTM-based approaches in remaining time prediction [104].
- PGTNET: This approach employs graph transformers to balance learning from the local contexts with capturing long-range dependencies [105], demonstrating state-of-the-art results.

For DALSTM and PGTNET, we use the settings reported in the original papers.

Evaluation measure. To evaluate the prediction accuracy, we use Mean Absolute Error (MAE), which measures the average magnitude of absolute errors between predicted and actual remaining time. Formally, MAE is defined as

[4] We excluded event logs from the BPI Challenge 2013 and 2020 due to long periods of process inactivity, the Traffic Fine log for its strong batching behavior, and the Environment Permit log for having too few events, making further segmentation unsuitable for training a prediction model.

MAE $= \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$, where n is the number of predictions, y_i represents the actual observed values, and $\hat{y}_i$ denotes the predicted values. Lower MAE values signify better predictive performance.

Results. Table 5.3 provides the MAE values obtained for all traces in an event log, denoted as Σ_L, as well as those obtained for traces belonging to steady states, denoted as Σ_L^S. To better clarify the findings, we calculate the average performance change, APC, that represents the change in the MAE between Σ_L and Σ_L^S across all approaches. Additionally, we include the steady-state ratio, SSR, defined as the percentage of steady-state traces relative to the total number of traces in each event log, i.e., $SSR = |\Sigma_L^S|/|\Sigma_L|$.

First, we observe that for most event logs, the APC is negative, ranging from -5.7% to -27.1%. This indicates that the MAE for traces executed in a steady state (columns Σ_L^S) is, on average, lower than when predictions are made using the entire dataset (columns Σ_L). This is expected, as processes in steady states have shorter lead times with less fluctuation, allowing for more accurate predictions of the remaining time for ongoing cases compared to non-steady states. In some event logs, such as BPIC15 Municipalities 2, 3, and 5, this trend holds consistently across different remaining time prediction approaches. For the remaining time prediction task, this finding highlights the importance of training separate models: one tailored for steady states and another optimized for non-steady states. This strategy is likely to provide more accurate predictions in terms of MAE compared to using a single model that is trained on the entire event log.

Table 5.3: Mean Absolute Error for remaining time prediction.

Event log	**DUMMY**		**DALSTM**		**PGTNet**		**APC**	**SSR**
	Σ_L	Σ_L^S	Σ_L	Σ_L^S	Σ_L	Σ_L^S		
Hospital	47.9	66.8	36.7	35.9	24.2	59.0	60.4	8.3
Sepsis	32.7	43.4	15.7	22.2	16.4	24.9	41.8	41.8
Helpdesk	12.3	10.6	12.9	10.9	5.4	6.1	-5.7	16.3
BPIC12	7.6	7.5	8.0	4.8	5.5	5.9	-11.3	43.5
BPIC15-1	38.2	40.9	29.3	37.9	20.4	27.3	23.6	56.9
BPIC15-2	72.9	61.4	47.0	43.8	68.8	59.2	-12.1	37.4
BPIC15-3	24.2	22.5	15.1	10.6	15.0	10.3	-22.6	37.0
BPIC15-4	77.2	86.5	72.7	45.4	82.7	36.6	-27.1	64.3
BPIC15-5	48.3	45.4	43.6	32.9	36.3	32.6	-13.6	44.0

Note: The average performance change (APC) and steady-state ratio (SSR) are in %.

However, in some event logs, the APC is positive, meaning the MAE has increased. This can be attributed to the specific characteristics of the recorded processes. In the Sepsis event log, the APC is 40%, due to the process's long warm-up and cool-down phases, which together account for over 50% of the total recorded time. As a result, steady states are detected too early, misclassifying the warm-up period and causing inaccurate detection. This highlights a challenge in business process mining, where SSD techniques from other fields may struggle to accurately detect and differentiate steady states from warm-up and cool-down phases. In the Hospital event log, the APC is also positive. However, the SSR is only 8%, indicating that the proportion of traces belonging to a steady state is very low. Consequently, the steady-state sublog may be too small to yield reliable results. Finally, for the BPIC15-1 event log, the APC is approximately 24%. A closer analysis of the detected steady states reveals that the event log may contain multiple steady states with different properties. The detected steady state at the beginning of the event log occurs when the number of active cases is high, while the second part features a lower number of active cases, leading to a qualitatively different steady state. In this case, a more appropriate approach would be to consider these two steady states independently.

Overall, this experiment demonstrates that our SSD framework can notably impact the insights for a downstream process mining task, making it a valuable preprocessing step, such as bucketing in the case of remaining time prediction. While the applicability of our framework may be influenced by certain specific characteristics of the recorded process behavior, it remains a highly effective approach for many business processes.

5.5 Related Work

In this section, we position the SSD problem in relation to other related problems in process mining.

Concept drift detection. The problem of SSD is related to concept drift detection in process mining, but they have different focuses. Concept drift detection focuses on identifying changes in the process [106], whereas SSD aims to determine when a process reaches a steady state. However, not every drift leads to a change in the steady state. For example, introducing a new business activity results in a drift in the control-flow of the process, yet the process may continue to operate at the same steady state.

Business process simulation. In business process simulation, SSD can be used to address the initialization bias (or startup issue) of simulation models [107]. Many simulations begin from an empty state, causing early fluctuations that distort results and limit analysis. The primary goal of SSD in process simulation is to identify when a process reaches a steady state, which is essential for predicting reliable long-term insights. Despite the similar terminology, the SSD problem discussed in this chapter is distinct, focusing on detecting the steady state of a business process based on past recorded behavior in an event log, and serving as a crucial preprocessing step for various offline process mining techniques.

Anomaly detection. Anomaly detection seeks to identify outliers or unusual patterns at the case level that deviate from expected process behavior [108]. In contrast, SSD focuses on identifying periods of stable, consistent process behavior across all active cases for a given period. However, SSD can provide a baseline for anomaly detection, making it easier to identify and explain unexpected behaviors. Once a steady state is reached, significant deviations can signal potential irregularities, while anomalies during non-steady states can often be explained by the process's inherent instability during that period.

Statistical quality control. The problem of SSD is closely related to statistical quality control (SQC) [109], both aiming to monitor process stability over time. However, SSD focuses on identifying when a process has reached a steady state, where its characteristics remain relatively stable. In contrast, SQC emphasizes detecting deviations from a desired range, typically defined by specific process characteristics that reflect the process's quality or efficiency. Moreover, it is important to note that reaching a steady state does not necessarily mean the process is operating within the optimal performance range that SQC seeks to maintain. A process can be stable but still fall outside the desired limits.

5.6 Conclusion

This chapter addresses the problem of SSD in business processes, emphasizing its importance in process mining and examining the applicability of existing SSD solutions within this domain. We propose a framework designed to identify steady states in a data-driven manner using information recorded in event logs. The framework first generates time series to represent key process characteristics and applies established SSD techniques to identify steady states in the time series and process levels, producing a sublog that captures the process behavior during these periods. The evaluation demonstrates that the framework effectively detects steady states in business processes and can enhance the accuracy and reliability of insights derived from a downstream process mining task.

In future work, we aim to enhance the proposed framework by addressing its limitations and extending its capabilities. First, as demonstrated in one of our experiments, the framework cannot be directly applied to event logs exhibiting atypical behavior, such as extended warm-up periods or periods without any business activities. This can be addressed by developing a preprocessing step that automatically verifies whether the event log meets the necessary conditions for SSD. Second, the experiment revealed that the overall accuracy of SSD in business processes can be improved, as these processes often exhibit unique behavioral characteristics that SSD techniques from other domains may not easily capture. Therefore, future work will focus on developing new SSD techniques tailored specifically to process mining. Finally, the framework could be extended to include an additional step that groups detected steady states based on similar characteristics. This would be particularly useful when the process records multiple steady states that exhibit different qualitative properties.

Chapter 6

Business Process Resilience Assessment

The study of business process dynamics opens opportunities to address problems that characterize a business process as a whole. In this chapter, we focus on the problem of assessing business process resilience, a characteristic that cannot be fully understood by analyzing isolated process instances. Instead, it requires a comprehensive perspective that considers the process as a whole. We present a data-driven approach that automatically assesses the resilience of a business process based on historical execution data recorded in an event log using system-level process mining. This chapter builds on our paper "Data-driven Assessment of Business Process Resilience" by Alexander Kraus, Jana-Rebecca Rehse, and Han van der Aa [11].

The remainder of this chapter is structured as follows. Section 6.1 introduces the topic. Section 6.2 reviews relevant background on process resilience and its assessment. Section 6.3 defines the problem under study, and Section 6.4 presents our proposed approach, which is evaluated in Section 6.5. Advantages and limitations are discussed in Section 6.6. Finally, Section 6.7 reviews related work, and Section 6.8 concludes the chapter.

6.1 Introduction

Organizations operate in dynamic environments that are subject to frequent changes. This includes the occurrence of disruptions, such as peaks in case arrivals, equipment failure, or absences in the workforce. Such disruptions can have a (temporary) negative impact on process performance. As shown in Figure 6.1, a process disruption can cause performance to initially worsen, e.g., resulting in an increased lead time, before slowly returning to its pre-disruption level. *Process resilience* refers to the ability of an organizational process to deal with such a disruption and recover from it [110]. As a means to ensure consistent performance despite the occurrence of disruptions, being resilient is thus a core competence for organizations [111]. In this regard, the first key step for organizations towards achieving resilience is to understand how resilient their processes actually are [112], which involves assessing how they respond to different disruptions.

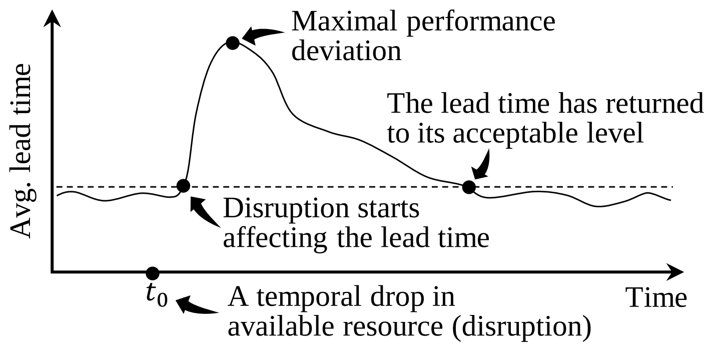


Figure 6.1: The impact of a disruption on process performance.

Despite the recognized importance of process resilience and its measurement [111–113], little research actually targets this measurement task, especially not in a data-driven manner. Existing works in this area either provide conceptual characterizations of process resilience [99, 114] and resilient business process management [113], or focus on achieving resilience in a design-time manner [115]. The one exception is a case study by Zahoransky et al. [116], where the authors describe how they assessed process resilience in a specific scenario, based on event data and a known process model. However, they do not propose a generalizable approach that is applicable to other processes. In addition, they strongly depend on domain knowledge (such as having an accurate process model) and manual analysis (for the estimation of key probabilities), which makes their resilience assessment impractical. Hence, the problem of how to properly and conveniently measure the resilience of organizational processes remains unaddressed.

The contribution of this chapter is an approach that assesses process resilience in a data-driven and automated manner, based on an event log and a resilience scenario of interest. For a potential disruption, such as a sudden absence of a resource or a peak in cases, our approach quantifies the disruption's impact on process performance according to four resilience measures, which jointly characterize the duration and severity of the disruption's impact. To do this, we establish a statistical model that captures the interrelations between different process characteristics over time. We then use this model to measure the impact of a potential future disruption on process performance. We evaluate our approach by assessing its accuracy through comparison with a "what-if" analysis by means of a simulation model. In addition, we demonstrate its effectiveness by assessing the resilience of the same process to diverse disruptions across different organizations.

6.2 Background

In this section, we explore the concept of resilience. We begin by examining resilience in general, considering its definition across various disciplines and existing assessment methods. Next, we delve into the definitions and assessment methods specific to organizational resilience. Finally, we focus on the resilience of business processes as a key layer of organizational resilience, highlighting the current lack of data-driven assessment methods and the need for their development.

Resilience as a concept. The concept of resilience has been discussed across various disciplines [117–119], ranging from psychology to seismology and to material science, as depicted in Table 6.1. However, a universal definition of resilience has not been established because different disciplines use specific terminology to address their unique needs and challenges. Despite these diverse perspectives, an overarching understanding of resilience can be summarized as the ability of a system to withstand a disruption within acceptable degradation and to recover within a suitable time and reasonable costs [120].

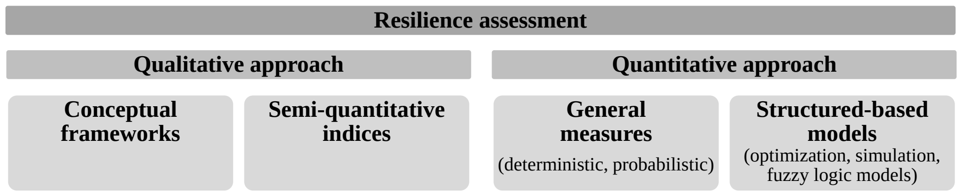


Figure 6.2: Resilience assessment approaches (adapted from [118]).

Resilience assessment methods can be grouped into two categories [118], as visualized in Figure 6.2. Qualitative approaches involve methods for assessing system resilience that do not rely on numerical calculations (cf. [121, 122]). These methods include conceptual frameworks that establish best practices, which are the primary qualitative approaches, along with semiquantitative indices that offer expert assessments of different qualitative aspects of resilience. Quantitative methods are categorized into two groups: general resilience approaches, which use deterministic or probabilistic measures to quantify resilience that are independent of

Table 6.1: Definitions of resilience in different disciplines.

Discipline	**Definition**
Psychology	The ability of individuals to recover from adversity. Positive ability of individuals to cope with stress and catastrophic events.
Seismology	The ability of the system to reduce the chances of shock, to absorb a shock if it occurs, and to recover quickly after a shock (re-establish normal performance).
Ecology	The magnitude of disturbance that a system can absorb before its structure is redefined by changing the variables and processes that control behavior.
Infrastructure	Ability of infrastructure to reduce the probability of failure, the consequences of such failure, and the response and recovery time.
Material Science	A material's tendency to return to its original form after applying a force or stress that has produced elastic deformation.
Engineering	The ability to sense, recognize, adapt, and absorb variations, changes, disturbances, disruptions, and surprises.
Tourism	Ability of communities (ecosystems) to withstand the impacts of external forces while retaining their integrity and ability to continue functioning.
Networks	The ability of a network to defend against and maintain an acceptable level of service in the presence of challenges.
Society	Capability of a system to maintain its functions and structure in the face of internal and external change and to degrade gracefully when it must.
Economics	Ability to reduce efficiently both the magnitude and duration of deviation from targeted system performance levels given the occurrence of a particular disruptive event.
Sociology	Ability to recover from adversity and become stronger than before.

The table summary is based on [117–119].

specific domains; and structural-based modeling approaches, which employ optimization, simulation, or fuzzy logic models to analyze how a system's structure affects its resilience in specific domains.

Organizational resilience. Over the past decade, research on organizational resilience has gained popularity, leading to the development of numerous approaches, indicators, and methodologies [117, 123, 124]. As shown in Table 6.2, the underlying conceptual definition generally considers a few key features that characterize an organization's response to a disruption. In the context of an organization, a disruption is an incident that causes an unplanned, negative deviation from the expected delivery of products and services according to the organization's objectives [125].

The importance of measuring organizational resilience is a key requirement

Table 6.2: Definitions of organizational resilience.

Key features	**Definition**
Absorption & Flexibility	Enterprise capacity to absorb changes and ruptures through flexibility without affecting its profitability.
Absorption & Transformation	The capacity to absorb shocks effectively, develop situation-specific responses to, and engage in transformative activities.
Anticipation & Adaptability	Ability to anticipate and adapt to key events related to emerging trends and to recover quickly after disasters and crises.
Robustness & Flexibility	Ability of an organization to strengthen the creation of robust and flexible processes in a proactive way.
Recovery & Avoidance	Ability not only to recover from disruptions but to avoid them completely.
Resistance & Recovery	Resistance to shocks, renewal, and recovery or bounce back from shocks.
Resistance & Adaptation	The ability to resist systematic discontinuities and the capability to adapt to new risk environments.
Vulnerability & Adaptation	The ability to manage vulnerabilities and adaptive response in a turbulent environment.
Withstanding & Anticipation	Reactive ability of the company to withstand an external event and active ability to anticipate events.
Withstanding & Adaptation	The ability to withstand systematic discontinuities as well as the capability to adapt to new risky environments.

The table summary is based on [117, 123].

to achieving resilience within an organization [114]. Consequently, various approaches have been proposed to measure organizational resilience, which can be categorized into six main groups [126]:

- *The systems view on measuring resilience* emphasizes understanding organizations as complex and dynamic entities, advocating a holistic approach that considers interconnected components, such as stakeholders and environment, to accurately assess resilience.
- *Resilience as an emergent feature of the system* underscores the importance of identifying inherent enterprise attributes that contribute to its resilience.
- *Inherent and adaptive characteristics of resilience* consider resilience under normal operating conditions and the deployment of resourcefulness and extra effort in crisis situations.
- *Resilience as a continuous process* views resilience as the outcome of continuous processes, including planning, responding to threats, and taking adap-

tive actions to recover.

- *Measuring resilience against disruptions* focuses on preventing and recovering from disruptive events.
- *Measuring resilience using adaptive capacity and time dimension* highlights the importance of the time taken for a system to respond and recover in understanding its resilience.

Next, we delve into existing research on the conceptualization and assessment of business process resilience, a crucial layer of organizational resilience [99].

Table 6.3: Definitions of process characteristics.

Characteristic	**Definition**
Absorption	The ability to dampen the impact of disruptive events.
Adaptability	The capability to respond to and adapt to the changing environment.
Agility	The ability to rapidly respond to changing conditions.
Flexibility	The ability to change and to adapt to new or complex situations.
Redundancy	The extra capacity to withstand potentially high-impact disruptions.
Robustness	The capability to withstand stress without significant loss.
Recovery	The ability to quickly resume operations at a desired performance.
Resourcefulness	The ability to diagnose problems and to initiate solutions.
Rapidity	The ability to react fast to changes in its environment.

The table summary is based on [123, 127].

Business process resilience. In general, *process resilience* can be summarized as the ability of a process to withstand and recover from a disruption. Similar to organizational resilience, the resilience of a business process can be defined by various characteristics outlined in Table 6.3, as enhancements in any of these characteristics ultimately strengthen the process's resilience.

As highlighted in Table 6.4, there has been little research conducted to assess resilience at the process level. Furthermore, the existing solutions—discussed in more detail in Section 6.7.1—predominantly rely on conceptual (qualitative) approaches, such as business continuity analysis [128] or value tree frameworks [129]. There are a few techniques that attempt to quantitatively assess process resilience with actual implementation and validation [112], such as methods that assess the resilience of business process architectures [116] or estimate various levels of process model resilience using a collaboration-oriented modeling language for processes [115, 130]. However, these approaches are structured-based methods primarily utilized to assess process resilience at *design-time*, i.e., during

Table 6.4: Overview of existing research on resilience assessment, highlighting the gap in quantitative approaches (general measures) for process resilience assessment.

Resilience assessment level	**Resilience assessment method**			
	Qualitative approach		**Quantitative approaches**	
	Conceptual frameworks	**Semi-quantitative indices**	**Structured-based models**	**General measures**
Across disciplines	✷✷✷	✷✷	✷✷✷	✷✷✷
➥ Organizations	✷✷	✷	✷✷	✷✷
➥ Processes	✷	(✷)	✷	ϟ

Legend: "✷✷✷" - Abundant research, "✷✷" - Moderate research, "✷" - Little research, "(✷)" - Indirect research (applicable from other levels), "ϟ" - Missing research (addressed gap).

the phase when a process is developed and planned before implementation. They do not evaluate the resilience of a process at *run-time*, i.e., when it is actually being performed. Run-time resilience assessment is crucial because it evaluates how well a process can respond disruptions that may not have been anticipated during the design phase. Furthermore, to avoid the need to rely on expectations about how a process is executed, run-time resilience assessment should be performed in a data-driven manner, which is not done by existing works. Therefore, there is a need for data-driven resilience assessment approaches that evaluate process resilience at run-time using process behavior recorded in event logs.

In the next section, we operationalize the problem of data-driven assessment of process resilience addressed in our work.

6.3 Problem Illustration

In this work, we propose an approach for the data-driven assessment of business process resilience. Our approach assesses such resilience in terms of the ability of a process to withstand and recover from disruptions. Specifically, it analyzes historic data about a process to estimate how its performance will be affected by future (previously unseen) disruptions. It thereby allows us to draw conclusions about the process's overall resilience based on an excerpt of its execution data, as captured in an event log. Next, we operationalize the problem tackled by our approach.

As input, our approach takes an *event log*, as specified in Section 2.1.2. The event log might contain additional information that can be relevant for assessing business process resilience, such as resource details for each event or trace attributes indicating different *case types* (such as regular or premium customers).

Our approach captures the state and progress of a process through *process features*. We define a process feature as follows:

Definition 11 (Process feature) *A process feature is a characteristic of a business process that is relevant for resilience assessment and measurable as a real-valued numerical metric over time, using information recorded in an event log L.*

We distinguish between *input or in-system features*, such as the numbers of case arrivals or active resources during a specific period, and *output features*, such as the number of completed cases or average lead time during a period. These latter features also encompass *Process Performance Indicators* (PPIs), which are quantifiable metrics that capture a process's effectiveness or efficiency [131]. They can relate to various dimensions, such as time, cost, and quality.

Our work assesses the resilience of a process by quantifying how a PPI will react to a *process disruption*, which we define as follows:

Definition 12 (Process disruption) *A process disruption D is a change in the value of an input or in-system process feature that is significant, exceeding the range of feature values that are normally observed for that business process, and temporary, meaning that the feature values return to the normally observed range after a given period of time.*

Examples of process disruptions are a drop in available resources or a rapid increase in case arrivals during a week.[1]

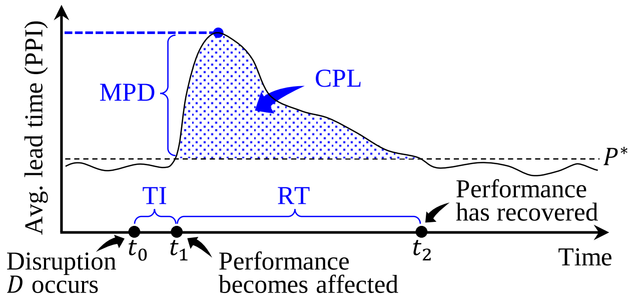


Figure 6.3: Four measures of process resilience.

We quantify the impact of a disruption D on a PPI P in terms of four measures derived from established concepts [118], as visualized in Figure 6.3, assuming an *acceptable performance level* $P^* \in \mathbb{R}$ as the threshold that PPI P must not exceed:

[1]Although an increase in case arrivals can be positive from a business perspective, it may still have negative effects on performance, such as the lead time of a process.

1. *Time-to-impact* ($\texttt{TI}_{[D,P]}$) captures the time between a disruption D, occurring at time t_0, and the next moment t_1 at which process performance P first goes beyond its acceptable performance level P^*.
2. *Recovery time* ($\texttt{RT}_{[D,P]}$) captures the time it takes for PPI P to return to its acceptable performance level P^* (at time t_2) after the initial impact at t_1.
3. *Maximal performance deviation* ($\texttt{MPD}_{[D,P]}$) captures the largest (negative) deflection of PPI P from its acceptable performance level P^* during the recovery period, i.e., between t_1 and t_2.
4. *Cumulative performance loss* ($\texttt{CPL}_{[D,P]}$) aggregates the total performance loss of P incurred during the recovery period, i.e., between t_1 and t_2.

Together, these measures provide in-depth insights into process resilience. Time-to-impact and maximal performance deviation quantify the ability of a process to withstand a disruption. Recovery time assesses the ability of a process to recover from a disruption. Finally, cumulative performance loss summarizes these two abilities by providing an aggregated value.

Given the above, we then define the resilience assessment problem that our work addresses as follows:

Definition 13 (Business Process Resilience Assessment Problem) *Business process resilience assessment uses historic process data in an event log L to estimate the impact of a possible future disruption D on a PPI P under normal process operations, quantified in terms of the time-to-impact, recovery time, maximal performance deviation, and cumulative performance loss.*

In the context of this problem, it is important to note that resilience assessment is not limited to past disruptions recorded in an event log, instead, it considers the impact of possible future process disruptions that can occur. In the next section, we present an approach that can assess process resilience for possible future process disruptions.

6.4 Approach

In this section, we provide a detailed description of our resilience assessment approach. As depicted in Figure 6.4, our approach takes as input an event log and a user-defined resilience scenario that specifies the PPIs, process disruptions, and further relevant process characteristics. Based on this scenario, our approach first generates process features as time series to capture the progression of each of the process characteristics over time. Then, we use these time series to establish a VAR model, which captures the linear interrelations between the different process characteristics. In the final step, we use the obtained VAR model to conduct impulse-response analysis to compute the four resilience measures depicted in Figure 6.3.

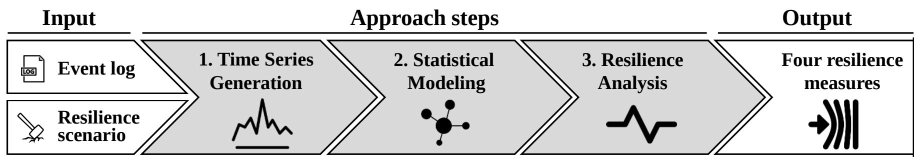


Figure 6.4: Overview of the main steps of our resilience assessment approach.

In the remainder, we describe our approach's input (Section 6.4.1) and three steps (Sections 6.4.2–6.4.4) in detail.

6.4.1 Resilience Scenario

Our approach takes as input a user-defined *resilience scenario* S. The resilience scenario captures the process features that are crucial for assessing process resilience from the user's perspective. Such a scenario is defined in terms of three sets of process features, $S := (F_P, F_D, F_A)$:

- *Performance measures (F_P):* This set contains the output features that are used as PPIs, such as the *average lead time of completed cases* or the *number of rejected orders*. F_P must contain at least one PPI, though a user can also specify multiple ones, so that the resilience of a process is considered from the perspective of different performance measures.
- *Disruption types (F_D):* This set defines process disruptions as input or in-system features that increase or decrease to cause disruptions, such as an increase in the *number of case arrivals* or a decrease in the *number of available resources*. F_D must contain at least one disruption type, though multiple disruptions can be considered in isolation or simultaneously.
- *Additional features (F_A):* Finally, a user can optionally define additional process features that may help to capture the interrelations between features in F_D and F_P (akin to *mediating variables*). For example, when determining the impact of deviations in the number of case arrivals (in F_D) on the average lead time (in F_P), F_A might include *the number of available resources* as an additional factor.

As a running example in this section, we will use as input the 5th log from the BPI Challenge 2015 [91] and a default resilience scenario S_d, with the *average lead time* as a PPI, i.e., $F_P = \{l\_t\}$, disruption types involving increases in the numbers of *case arrivals*, and *active cases*, and decreases in the *available resources*, i.e., $F_D = \{\uparrow arr\_c, \uparrow act\_c, \downarrow avl\_r\}$, and $F_A = \emptyset$. However, we stress that our approach can work with any process features measurable for an event log. For instance, a resilience scenario can include log-specific features, such as the *number of active cases by premium customers* or the *number of available specialists*.

6.4.2 Time Series Generation

In the first step, we generate time series that represent the evolution of process features within a resilience scenario. Time series generation is commonly used in process mining to address problems such as concept drift detection and explanation [51], or to assess process complexity and its impact on performance [93]. In the following, we detail the three steps of time series generation used in our approach: windowing, time series construction, and warm-up and cool-down phase detection.

Windowing. First, we split the period of an event log L using time-based tumbling windows of fixed length l [22]. This gives a series of non-overlapping time windows $W_l := \langle w_1, \dots, w_n \rangle$ of equal length, such that each event $e \in L$ belongs to exactly one window w_t for $t \in \{1, \dots, n\}$. The first window w_1 starts at the earliest event in L, whereas the last event is in w_n. Our approach can consider different options for window lengths, e.g., l_1, l_2, etc., resulting in a set of window sequences $\mathcal{W} := \{W_{l_1}, W_{l_2}, ...\}$. Based on the modeling results in the next step, our approach automatically selects a window length l that best describes the evolution of the process features over time.

Time series construction. Following Definition 10, we define a time series as a set of observations $\{y_t \mid t \in \{t_0, \dots, t_n\}\}$, where each $y_t \in \mathbb{R}$ represents an observation made at a time t ($\{y_t\}$ in short). We particularly consider series over discrete, equally-spaced time intervals, which means that each y_t captures an observation made for a specific period, e.g., a day or a week.

For all window sequences $W_l \in \mathcal{W}$, we construct time series for each process feature $f_k \in F := F_P \cup F_D \cup F_A$, with $k \in \{1, ..., |F|\}$ as an index. For this, we first calculate the feature value $y_{k,t} \in \mathbb{R}$ for the feature f_k for each window $w_t \in W_l$. Then, we combine these sequential values into a time series $\{y_{k,t}\}$, which captures the evolution of f_k over the windows in W_l. These two steps are repeated for each feature $f_k \in F$, resulting in a set $Y_l := \{\{y_{k,t}\}\}$. Repeating this for each window sequence, we obtain a set $Y := \{Y_l \mid W_l \in \mathcal{W}\}$. Figure 6.5 shows the week-based time series Y_{week} for our running example.

Warm-up and cool-down detection. Finally, we check if the beginning and end of the time series should be truncated. This can be necessary because event logs are data snapshots, consisting of cases associated with a particular time frame. For instance, the running example's log appears to contain all cases that were active

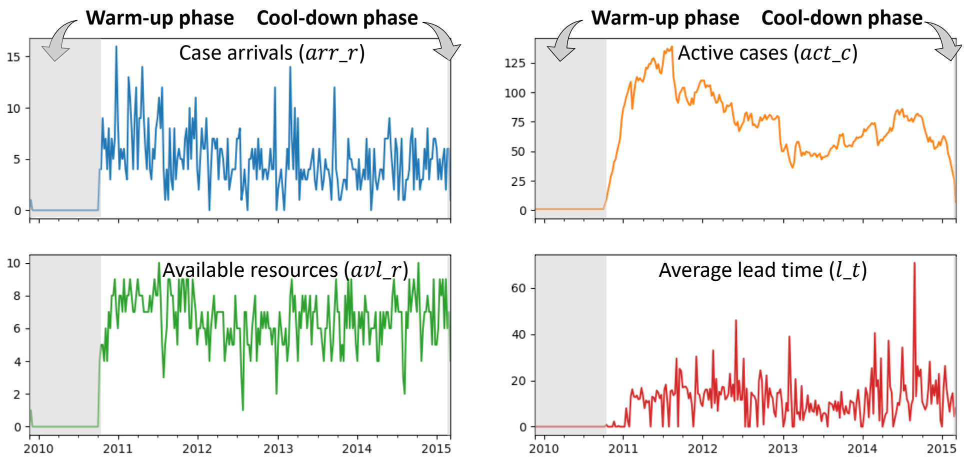


Figure 6.5: Process features derived for the running example with indicated warm-up and cool-down phases.

between early 2011 and March 2015. However, since some of these cases started much earlier than 2011, the event log has a lengthy *warm-up phase*, in which there are few active cases in the system, as shown in Figure 6.5. Similarly, if a log consists of cases that started in a certain period, there will be a *cool-down phase* at the end, in which only a few to-be-completed cases remain in the log.

To detect warm-up and cool-down phases, we use the time series $\{y_{k,t}\}$ that describes the evolution of the number of active cases over time (act_c).[2] We use the corresponding frequency distribution of the values $y_{k,t}$ and a percentile δ to define a threshold. We then use this threshold to detect a sequence of windows w_t at the beginning (warm-up) and at the end (cool-down) of W_l, where the feature values $y_{k,t}$ are all below this threshold. Finally, we remove feature values that belong to the detected windows from all features in the set Y_l, resulting in truncated time series Y_l^*. Repeating this procedure for each window sequence W_l, we get $Y^* := \{Y_l^*\}$. In Figure 6.5, the detected warm-up and cool-down phases are highlighted in gray. For the 1st percentile, $\delta = 1$, we removed 48 windows at the beginning and two at the end.

6.4.3 Statistical Modeling

In the second step, our approach uses the obtained time series to build a collection of VAR models, from which an optimal model is then selected. In this section, we

[2]We use this feature even if it is not part of the resilience scenario of interest.

first introduce VAR models and then explain the model generation and selection steps.

Vector autoregressive models. VAR models are among the most successful and flexible statistical modeling techniques for analyzing multivariate time series. Initially developed by Sims [132], they have proven to be helpful in describing and forecasting process dynamics in many domains. VAR models relate current observations of a variable to past observations of the same and other variables (called *lags*) via a linear combination [133].

Model definition. Given a set of K features, we define a K-dimensional vector $\mathbf{y}_t := (y_{1,t}, \dots, y_{K,t}) \in \mathbb{R}^K$, which captures the time series values of the features for a given window w_t. Using this vector notation, the standard VAR model is defined as a system of K linear equations:

$$\mathbf{y}_t = A_0\mathbf{d} + \sum_{i=1}^{p} A_i\mathbf{y}_{t-i} + \mathbf{e}_t. \tag{6.1}$$

The vector $\mathbf{d} \in \mathbb{R}^{m\times 1}$ holds m deterministic parameters, which can be used to model, e.g., an intercept ($\mathbf{d} = 1 \in \mathbb{R}$) or a linear trend ($\mathbf{d} = (1, t) \in \mathbb{R}^2$). The matrix $A_0 \in \mathbb{R}^{K\times m}$ holds the parameters used in $\mathbf{d}$. The value $p \in \mathbb{N}$ is the model order, defining the number of lags that are considered by the model via the vectors $\mathbf{y}_{t-i}$, for $i \in \{1, \dots, p\}$. Matrices $A_i \in \mathbb{R}^{K\times K}$ hold model coefficients that show the per-lag impact of the features on one another. Finally, $\mathbf{e}_t := (e_{1,t}, \dots, e_{K,t}) \in \mathbb{R}^K$ is a vector of error terms.

As an illustration, we consider a first-order VAR model ($p = 1$) with three time series ($K = 3$): $y_{1,t}$, $y_{2,t}$, and $y_{3,t}$, assuming that each model equation includes a linear trend, i.e., $\mathbf{d} = (1, t) \in \mathbb{R}^2$. Then, Equation 6.1 takes the following form:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \\ y_{3,t} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{bmatrix} \begin{bmatrix} 1 \\ t \end{bmatrix} + \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} \end{bmatrix} \begin{bmatrix} y_{1,t\text{-}1} \\ y_{2,t\text{-}1} \\ y_{3,t\text{-}1} \end{bmatrix} + \begin{bmatrix} e_{1,t} \\ e_{2,t} \\ e_{3,t} \end{bmatrix}. \tag{6.2}$$

Model assumptions. An adequate VAR model should fulfill two assumptions:

1. A VAR model's error terms $\mathbf{e}_t$ should be *white noise*, which is the case when the expectation of the error terms is zero, $E[\mathbf{e}_t] = 0$, the contemporaneous covariance matrix of error terms is nonsingular, $E[\mathbf{e}_t\mathbf{e}_t^T] = \Sigma_\mathbf{e}$, the error terms are uncorrelated, $E[\mathbf{e}_t\mathbf{e}_s^T] = 0$ for $t \neq s$, and all fourth moments $E[\mathbf{e}_t^4]$ exist and are bounded (large outliers are unlikely) [134].
2. A VAR model should be stable, which holds when its reverse characteristic polynomial has no roots in and on the complex unit circle [134]. Stability of

a VAR model also implies stationarity [134, p. 25], which is necessary for impulse-response analysis (Section 6.4.4).

In general, VAR models typically assume that all time series are stationary, meaning their statistical properties such as mean and variance remain stable over time. However, this assumption can be overly constraining in practical scenarios, especially within business process analysis. Unlike domains such as finance, where significant trends or abrupt changes are common, features capturing process characteristics often exhibit less variation. Nonetheless, business processes can still undergo subtle or seasonal drifts. In this scenario, concept drift detection could be regarded as an additional preprocessing step to identify different versions of the process and independently assess the resilience of each version before extracting the relevant time series. Alternatively, if seasonality is a permanent characteristic of a business process, standardized techniques for seasonality removal, such as seasonal-trend decomposition based on loess [135], can be applied after time series are extracted.

Model estimation. In situations where these assumptions are fulfilled, the model parameters in Equation 6.1 can be obtained using multivariate least squares (LS) estimation [134, p. 74]. The resulting LS estimates then have two key asymptotic properties: consistency and normality, which are important to obtain reliable estimation for model parameters [136].

Model generation and selection. Since the optimal configuration of a VAR model cannot be known a priori for a given event log, our approach first generates a collection of candidate VAR models using various parameter settings. Then, it checks the validity of these candidate models regarding the aforementioned model assumptions and, from the valid models, selects the one that has the best fit according to an established information criterion.

Candidate-model generation. We first generate a set of candidate VAR models $\mathcal{M}$, so that each model $M \in \mathcal{M}$ has a unique combination of four parameters:

1. A window sequence $W_l \in \mathcal{W}$.
2. A choice for the deterministic parameter **d** from a set of options, which commonly contains $\mathbf{d} = 0$ (no trend or intercept), $\mathbf{d} = 1 \in \mathbb{R}$ (an intercept), $\mathbf{d} = (1, t) \in \mathbb{R}^2$ (a linear trend).
3. A model order $p \in \{1, ..., p_{max}\}$, where p_{max} is the highest possible model order, which can be heuristically estimated[3] by the total number of observations $n = |W_l|$ as $12\sqrt[4]{n/100}$.
4. A set of features F_M to be included in the model. F_M contains all PPIs from F_P and disruption types from F_D, plus a (possibly empty) subset of

[3]This heuristic is applied by the Python library we employ in our implementation [137].

additional features from F_A. By including different subsets of F_A, we test whether including extra features leads to a better-fitting model.

Assumption checking. Next, for each of the candidate VAR models in $\mathcal{M}$, we check if it meets the assumptions of noise whiteness and model stability. Given a candidate model $M \in \mathcal{M}$ of order p, we check whether the error terms fulfill the white noise assumption using the multivariate Ljung–Box portmanteau test [138]. The test's null hypothesis is that there is no overall significance in the auto-correlations of the error terms. If we cannot reject the null hypothesis at the significance level $\alpha = 0.05$ and for $p+1$ lags, then we conclude that there is no evidence to reject the white noise assumption [134, p. 169]. We determine the stability of a model using the eigenvalues of its companion-form representation of order 1, which exists for any model M and provides a more compact form [134]. If all absolute eigenvalues of the companion-form matrix are less than one, the model M is stable [134, p. 15]. We add the models that meet both assumptions to a set of accepted models, $\mathcal{M}_a \subseteq \mathcal{M}$. If no models are found that fulfill the assumptions, we extend the search space and consider models with any set of features F_M that contains at least one performance measure and one disruption type.

Optimal model selection. Finally, we select the optimal model M_a^* as the model in $\mathcal{M}_a$ with the best (i.e., lowest) Akaike Information Criterion (AIC) [139] score. AIC is a widely-employed metric that evaluates a statistical model's ability to fit the data at hand, while accounting for the model's complexity [134]. Given the objective of the model to capture as much information as possible from various factors to better describe dependencies during the impulse-response analysis, AIC is a better choice compared to other information criteria, such as the Bayesian Information Criterion (BIC) or the Hannan-Quinn Information Criterion (HQIC), which impose heavier penalties for model complexity. Moreover, AIC tends to perform well with limited data points, a common situation when working with time series data extracted from event logs [140].

In general, optimal model selection is a challenging task. However, the use of AIC provides a simplification that has proven useful and practical for the addressed problem. Nevertheless, the proposed model selection can be improved by incorporating additional measures or procedures, such as cross-validation [141], or more complex techniques [142, 143].

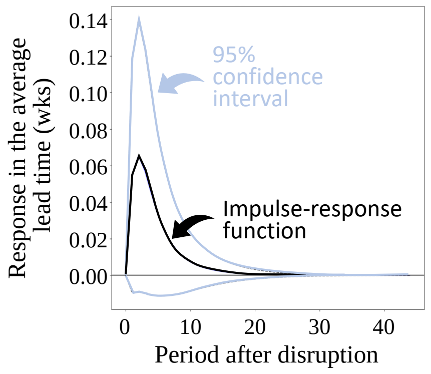
6.4.4 Resilience Analysis

In the third step, we assess process resilience using impulse-response analysis, which estimates the expected impact of disruptions on process performance. This corresponds to a form of "what-if" analysis, which can be derived directly from the obtained optimal VAR model. We first introduce its concept and then explain how we estimate the four resilience measures (time-to-impact, recovery time, maximal performance deviation, and cumulative performance loss) for a given disruption and the corresponding response in a PPI.

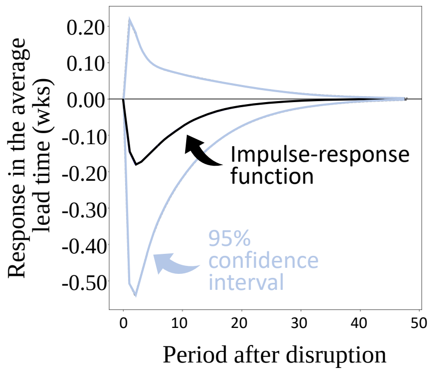
Impulse-response analysis. A VAR model's properties are typically assessed through structural analysis such as Granger causality, impulse responses, and forecast error variance decompositions, since examining individual model coefficients alone does not offer a comprehensive understanding of the interactions among the variables in a model. For the purpose of assessing business process resilience, impulse-response analysis serves as the right instrument.

Intuition. Impulse response analysis is a technique for interpreting VAR models. It is based on a *impulse-response function* that predicts how one variable responds to a sudden change in another variable [134], given the identified interrelations among model variables in Equation 6.1. To ensure a clean assessment of the response in one variable to a change in another, the analysis is conducted after centering each variable around 0, by subtracting its expected value. Next, a one-unit change is introduced in one variable, and its impact on another variable is observed over subsequent periods, accounting for detected inter-dependencies between the model variables. A one-unit change can always be considered since all time series in a VAR model are numerical and real-valued. Given that the introduced impulse is a one-time change and the VAR model is stable (an assumption checked during the search for the optimal model), the model will return to its centered state after the initial impulse has propagated through the system. This way, impulse-response analysis facilitates a deeper understanding of the dynamic interactions among variables over time, proving valuable in assessing process resilience.

We illustrate the concept of impulse-response analysis using two exemplary impulse-response functions, depicted in Figure 6.6. The first impulse-response function (Figure 6.6a) demonstrates the expected increase (black line) in the average lead time following a disruption in the total number of active cases during the periods $h \in H := 1, 2, ..., h_{max}$. The disruption occurs in period $h = 0$ and corresponds to a one-time, one-unit increase in the total number of active cases. The maximum expected deviation in the lead time is observed in the second period after the impulse, elevating the average lead time above its average value by about 0.06 units. The impulse-response analysis also provides confidence intervals (blue lines) indicating where the response in the lead time can be expected with a 95% probability. The second impulse-response function (Figure 6.6b) illustrates the anticipated change in the average lead time after a one-unit increase in the number of available resources. In this scenario, the expected lead time decreases since more resources are available, making it more likely for the lead time to decrease.



(a) Impulse corresponds to a one-unit increase in the total number of active cases.



(b) Impulse corresponds to a one-unit increase in the total number of available resources.

Figure 6.6: Two examples of impulse-response functions (incl. 95% confidence interval), showing the expected response in the average lead time (in weeks) to a one-unit increase in the impulse.

Mathematical derivation. For each feature combination $f_d \in F_D$ and $f_p \in F_P$, we obtain the corresponding impulse-response function $\texttt{IRF}[f_d, f_p](h)$, showing the impact of a potential disruption in feature f_d on performance indicator f_p at the moment h after the disruption, by transforming the (stable) VAR model into its infinite *moving average representation* [134]. To illustrate this transformation, we consider a VAR model of order 1 since any VAR model of order p can be rewritten to that, using the following matrix form [134]:

$$\underbrace{\begin{bmatrix} \mathbf{y}_t \\ \mathbf{y}_{t-1} \\ \mathbf{y}_{t-2} \\ \vdots \\ \mathbf{y}_{t-p+1} \end{bmatrix}}_{=:Z_t} = \underbrace{\begin{bmatrix} \mu \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{=:\Gamma_0} + \underbrace{\begin{bmatrix} A_1 & A_2 & \cdots & A_{p-1} & A_p \\ I & 0 & \cdots & 0 & 0 \\ 0 & I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I & 0 \end{bmatrix}}_{=:\Gamma_1} \underbrace{\begin{bmatrix} \mathbf{y}_{t-1} \\ \mathbf{y}_{t-2} \\ \mathbf{y}_{t-3} \\ \vdots \\ \mathbf{y}_{t-p} \end{bmatrix}}_{=:Z_{t-1}} + \underbrace{\begin{bmatrix} \mathbf{e}_t \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{=:\Upsilon_t}.$$

This matrix form of any VAR model of order p can be simplified using $Z_t, \Gamma_0, \Gamma_1, \Upsilon_t$:

$$Z_t = \Gamma_0 + \Gamma_1 Z_{t-1} + \Upsilon_t.$$

Using the initial notation used in Equation 6.1 for $p = 1$, we can further simplify the obtained equation:

$$\mathbf{y}_t = \mu + A_1\mathbf{y}_{t-1} + \mathbf{e}_t,$$

with $\mu := A_0\mathbf{d}$. Now, we can derive its infinite moving average representation by using recursive substitution [134]:

$$\begin{aligned}\mathbf{y}_t &= \big(1 + A_1 + A_1^2 + \ldots + A_1^j\big)\mu + A_1^{j+1}\mathbf{y}_{t-(j+1)} \\ &\quad + \big(\mathbf{e}_t + A_1\mathbf{e}_{t-1} + A_1^2\mathbf{e}_{t-2} + \ldots + A_1^j\mathbf{e}_{t-j}\big).\end{aligned} \tag{6.3}$$

For a stable VAR process, the following results hold [134]:

$$(I + A_1 + \ldots + A_1^j)\mu \rightarrow (I - A_1)^{-1}\mu \text{ and } A_1^{j+1}\mathbf{y}_{t-j-1} \rightarrow 0, \text{ as } j \rightarrow \infty.$$

Hence, Equation 6.3 reduces to:

$$\mathbf{y}_t = \varphi + \sum_{j=0}^{\infty} A_1^j\mathbf{e}_{t-j},$$

where $\varphi := (I - A_1)^{-1}\mu$. Now, by rewriting $B_j := A_1^j$ and $B_0 := I$, we get:

$$\mathbf{y}_t = \varphi + \sum_{j=0}^{\infty} B_j\mathbf{e}_{t-j}. \tag{6.4}$$

Finally, the obtained representation is used to track the impact of a disturbance in one variable on the other. This is done by initiating the impulse vector $\mathbf{e^t} := (\mathbf{0}, \ldots, \mathbf{0}, \mathbf{1}, \mathbf{0}, \ldots \mathbf{0})$, where the 1 in $\mathbf{e^t}$ corresponds to the position of the feature f_d in the VAR model. Afterward, the impact on any variable within the system for a given future time period t can be assessed by examining the outcome of the computations in Equation 6.4, given by the right-hand side of the equation. The 95% confidence interval for each period t and pair of impulse and response variables is derived using the associated covariance matrix of $\varphi \sum_{j=0}^{\infty} B_j$, where j ends at t.

Resilience assessment. To assess process resilience, we consider the resilience scenario $S = (F_P, F_D, F_A)$ and the selected optimal VAR model M_a^*. For each feature combination $f_d \in F_D$ and $f_p \in F_P$, we use the corresponding impulse-response function $\texttt{IRF}[f_d, f_p](h)$, which estimates the expected impact of a potential disruption in feature f_d on performance indicator f_p.

Resilience aspects. We account for the following aspects when using an impulse-response function to assess business process resilience:

1. *Disruption size*: By default, an impulse-response function considers the impact of one additional unit, which may be insufficient for modeling an actual disruption. However, as a VAR model is linear, the initial impulse can be rescaled by multiplication with a scalar of interest. A common choice is one, two, or three standard deviations of the impulse variable. This rescaling is widely employed as it facilitates the comparison of disruptions' impacts across variables with different scaling [134]. Consequently, the values of the impulse-response function are rescaled as well, maintaining the same shape. Using the impulse-response function in Figure 6.6a as an example, if the standard deviation of the number of active cases is 10, and the average lead time is 3 weeks, then a disruption, represented by 2 standard deviations, will increase the lead time by $10 \times 2 \times 0.06 = 1.2$ weeks in the second period after the disruption. In other words, the maximum expected increase in the average lead time is approximately 40% (1.2 wks / 3 wks).
2. *Disruption direction*: Depending on the resilience scenario, we may be interested in disruptions that increase or decrease the impulse variable. For example, when assessing resilience to disruptions in the number of active cases, we want to evaluate the impact of increases, while in the case of available resources, we want to observe the effect of decreases. To obtain the impact of a decreasing disruption, we multiply the values of the impulse-response function by -1. In the context of the impulse-response function in Figure 6.6b, where we aim to observe the response to a decrease in the number of available resources, this action will mirror the obtained curves along the x-axis.
3. *PPI's negative deviation*: When examining the response in a PPI, our focus lies on negative deviations, which we denote as $\texttt{IRF}[f_d, f_p](h)^-$. Depending on the PPI, a negative impact can be linked to either an increase or decrease in the PPI (or both if the PPI needs to remain within a specified range). In the scenario of average lead time, an increase is deemed negative as it results in prolonged execution time, commonly viewed as an undesirable outcome. Moreover, deviations in a PPI are commonly tolerated up to a certain level. For this reason, we consider a specific *tolerated deviation*, P_{tol}. This defines the accepted deviation of a PPI from its average (target) value, i.e., $P^* = \bar{f}_p + P_{tol}$ (or $-P_{tol}$ for PPIs where higher is better). If the response to a disruption is less than P_{tol}, the disruption is not considered to impact the PPI.
4. *PPI's deviation confidence*: When assessing the expected PPI deviation after a disruption, we aim to measure process resilience in terms of the worst-case scenario. In other words, we want to ensure that the performance deviation does not exceed a specific threshold, like 2 weeks for average lead time, with

a 95% probability. For this, we are more interested in the deviations given by the upper and lower confidence bounds (blue lines) in Figure 6.6 around the expected response given by the black curve. Depending on the resilience scenario, we denote $\mathtt{IRF}^*[f_d, f_p](h)^-$ as the relevant negative extreme bound for the PPI's deviation.

Next, we explain how we define resilience measures using the relevant impulse-response function.

Resilience measures. For a given resilience scenario $S = (F_P, F_D, F_A)$ and the tolerated deviation P_{tol}, we define the following four resilience measures:

- *Time-to-impact*: We define time-to-impact as the first period after a disruption where the PPI's negative deviation exceeds P_{tol}:

  $$\mathtt{TI}_{[f_d,f_p]} := \inf\{h \in H : |\mathtt{IRF}^*[f_d, f_p](h)^-| > P_{tol}\}.$$

  If the deviation does not exceed P_{tol}, all remaining resilience measures are 0, since there is no impact on the PPI.

- *Recovery time*: The recovery time is given by the difference between the last time at which the predicted negative deviation is greater than the tolerated deviation and the time-to-impact value:

  $$\mathtt{RT}_{[f_d,f_p]} := \sup\{h \in H : |\mathtt{IRF}^*[f_d, f_p](h)^-| > P_{tol}\} - \mathtt{TI}_{[f_d,f_p]}.$$

  Both time-to-impact and recovery time take integer between 0 and h_{max}.

- *Maximal performance deviation*: The maximal performance deviation is given as the greatest absolute negative deviation of the impulse-response function during the recovery period:

  $$\mathtt{MPD}_{[f_d,f_p]} := \sup\{|\mathtt{IRF}^*[f_d, f_p](h)^-|,\ h \in \{\mathtt{TI}_{[f_d,f_p]}, \ldots, \mathtt{TI}_{[f_d,f_p]} + \mathtt{RT}_{[f_d,f_p]}]\}\}.$$

- *Cumulative performance loss*: To quantify the cumulative performance loss, we sum up the gains and losses incurred between initial impact and recovery:

  $$\mathtt{CPL}_{[f_d,f_p]} := \sum_h \mathtt{IRF}^*[f_d, f_p](h),\ h \in \{\mathtt{TI}_{[f_d,f_p]}, \ldots, \mathtt{TI}_{[f_d,f_p]} + \mathtt{RT}_{[f_d,f_p]}\}.$$

  Both $\mathtt{MPD}_{[f_d,f_p]}$ and $\mathtt{CPL}_{[f_d,f_p]}$ takes a value in $\mathbb{R}^+$.

Example. To illustrate the output obtained in this manner, Figure 6.7 depicts an impulse-response function $\mathtt{IRF}^*[f_d, f_p](h)$ for the running example. It shows the response of the average lead time (l_t, in weeks) to a disruption of two standard

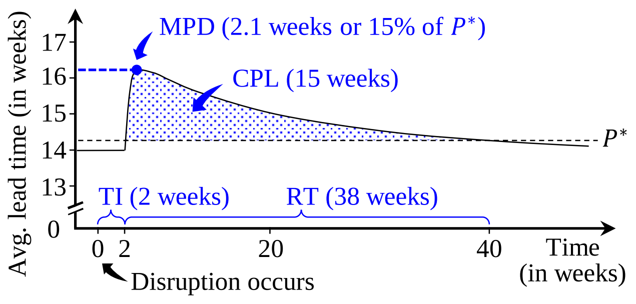


Figure 6.7: Process resilience in terms of the average lead time in response to a disruption in the number of active cases for the running example.

deviations in the number of active cases ($\uparrow act\_c$), using a tolerated deviation P_{tol} of 1 percent (about 0.15 week). With a probability of 95%, the actual increase in the lead time is not going to exceed the predicted response $\texttt{IRF}^*[f_d, f_p](h)$ (black curve). As shown, the disruption impact occurs quickly (time-to-impact of just two weeks) and increases the average lead time from about 14 weeks to a maximum of 16.1 (an increase of 15%). After this initial spike, the process starts recovering gradually, though it takes a total of 40 weeks before it returns to its acceptable performance level. Overall, the cumulative performance loss is 15 weeks, which can be used when comparing the impact of this disruption to other types or when comparing the process' resilience to other versions. We demonstrate this, among others, in the evaluation experiments performed next.

6.5 Evaluation

This section reports on two experiments conducted to evaluate our approach. In the first experiment, we assess the validity of our approach through a controlled process simulation. In the second experiment, we evaluate the effectiveness of our approach by applying it to real-life event logs.

Our experiments can be replicated using our repository[4], which contains a Python prototype of our approach, input data, relevant experimental details, and raw results. Our prototype employs functions from the *PM4Py* [144] and *statsmodels* [137] libraries.

[4]Project repository: `https://gitlab.uni-mannheim.de/processanalytics/resilience`

6.5.1 Experiment 1: Approach validity

In the first experiment, we demonstrate the validity of our approach by assessing the accuracy of estimated resilience measures. To accomplish this, we conduct multiple simulations of a business process under a predetermined resilience scenario and compare the actual resilience measures—from the simulation—against the estimations obtained using our approach.

Setup

In the following, we discuss the relevant experiment settings, i.e., the simulation configurations, resilience scenario, approach configurations, and evaluation metric.

Simulation scenario. As a basis for this experiment, we consider a process model and its simulation parameters that were used in prior work on business process resilience assessment [116]. The model captures an order-to-cash process of a medium-sized company, depicted in Figure 6.8.

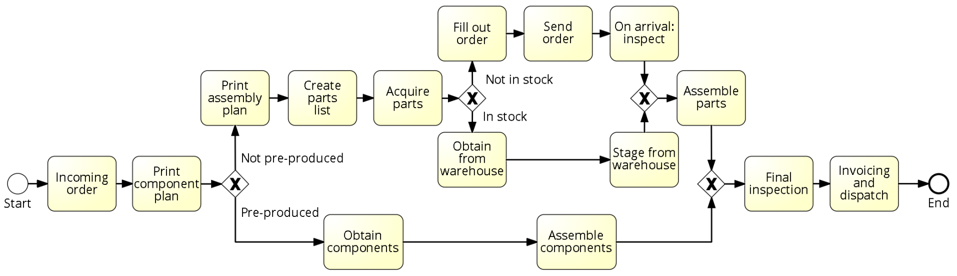


Figure 6.8: Simulated order-to-cash process (adapted from [116]).

Control-flow, processing times, and resources. The process model consists of 15 activities and two decision points. Following the parameters of the original scenario [116], each decision point has an equal branching probability when simulating the process. Furthermore, the execution times of the activities are given by the distributions shown in Table 6.5. Finally, each task is managed by one dedicated resource, operating 24/7.

Arrival process. The original paper does not specify how case arrivals were simulated, thus we utilize the Poisson distribution $P(\lambda)$ for the arrival process. We use an arrival rate of $\lambda_n = 0.45$ cases per hour, which leads to stable process utilization without causing queues to grow.

Process disruption. We perform simulation runs that last 10,000 (simulated) hours, with each run involving a disruption in the case arrival rate λ after 70% of the

Task	**Distribution**	**Parameters (hours)**	
Incoming order	Log-normal	$\mu = 0.2$	$\sigma = 0.4$
Print component plan	Normal	$\mu = 0.5$	$\sigma = 0.1$
Print assembly plan	Normal	$\mu = 0.6$	$\sigma = 0.15$
Create part list	Gamma	$\alpha = 0.9$	$\beta = 0.7$
Acquire parts	Gamma	$\alpha = 0.8$	$\beta = 0.8$
Fill out order	Log-normal	$\mu = 0.1$	$\sigma = 0.5$
Send order	Gamma	$\alpha = 1.0$	$\beta = 0.5$
Arrival and inspection	Log-normal	$\mu = 0.25$	$\sigma = 0.5$
Obtain from warehouse	Log-normal	$\mu = 0.07$	$\sigma = 0.3$
Stage from warehouse	Gamma	$\alpha = 0.8$	$\beta = 0.3$
Assemble parts	Log-normal	$\mu = 1.3$	$\sigma = 0.4$
Obtain components*	Log-normal	$\mu = 0.4$	$\sigma = 0.4$
Assemble components	Log-normal	$\mu = 0.4$	$\sigma = 0.4$
Final inspection	Normal	$\mu = 1.0$	$\sigma = 0.4$
Invoicing and dispatch	Normal	$\mu = 0.8$	$\sigma = 0.3$

* No specifications were provided for the activity “Obtain components", therefore, we used the same setup as for the activity “Assemble components".

Table 6.5: Simulation setup for the process depicted in Figure 6.8.

time. This timing ensures the model stabilizes before the disruption, and allows sufficient time afterward to observe changes in the PPI. The increase in the arrival rate persists for exactly one window, returning then to its normal level of $\lambda_n = 0.45$ cases per hour.

We simulate disruptions by increasing the number of case arrivals by up to 20 standard deviations. This is achieved by multiplying the arrival rate by a factor ranging between 2 and 4 times the normal arrival rate. For each tested increase in the arrival rate, we conduct 1000 simulations, resulting in 10000 simulation runs in total.

Implementation. To simulate the resilience scenario, we built a simulation model using the CIW library [100], an open-source library for conducting discrete event simulations[5]. This library allows us to simulate the process as a network of queues, providing the functionality and simulation logic required to model the desired simulation. The total computation time for his experiment took about 3 hours on 50 CPUs[6].

[5]Available online: `https://ciw.readthedocs.io/en/latest/index.html`

[6]Intel Xeon CPU E5-2698 v4 @ 2.20GHz.

Resilience scenario. We consider a resilience scenario $S = (F_P, F_D, F_A)$, where the disruption type F_D is given by an increase in the number of case arrivals. The performance measure F_P is given by the average lead time derived from the lead time of all cases that finished during the window. Finally, the additional feature F_A is given by the number of active cases determined by the number of cases that were in progress during the window[7].

Approach configuration. We use the following settings when applying our approach. In Step 1, we use a fixed window length l of 70 hours (appr. 3 days). In this controlled experiment, this window size leads to a balanced representation of the process features as time series over the total simulation duration. It is neither too large, ensuring a sufficient length of the time series before the disruption to fit VAR models, nor too small, preventing any of the derived process features from having zero values. We test five percentiles for the warm-up and cool-down phase detection parameter δ, i.e., $\delta \in \{1\%, 5\%, 10\%, 15\%, 20\%\}$. In Step 2, we fit VAR models without an intercept, with an intercept, and with an intercept and a linear trend, i.e., $\mathbf{d} \in \{0, 1, (1, t)\}$ and use a significance level $\alpha = 0.05$ for the multivariate Ljung–Box portmanteau test. In Step 3, we consider a response horizon h_{max} of 10 windows after the disruption moment, which corresponds to about 10% of the simulated time before the disruption. This response horizon covers enough time to capture the response in the post-disruption PPI. We consider a tolerated deviation P_{tol} of 30% of the standard deviation of the average lead time observed before the disruption.

Accuracy assessment. To evaluate our approach, we assess how often each resilience measure falls within the interval estimated by our approach for each simulation run. To derive the interval estimated by our approach, we apply our approach to the portion of the event log that occurs before the disruption. This way our approach does not observe the disruption in the data nor does it learn how the process reacts to it. Then, we derive lower R_l^{est} and upper R_u^{est} bounds using the impulse-response functions that denote the corresponding 99% confidence interval, resulting in the interval $I := [R_l^{est}, R_u^{est}]$. To derive the actual resilience measure R^{act} from the simulation, we consider the response in the PPI following the disruption.

Given R^{act} and I for each resilience measure, we evaluate our approach by considering its accuracy, denoted Acc, as the proportion of simulations where the actual value R^{act} falls within an interval of interest $I = [R_l^{est}, R_u^{est}]$ (correct detections) divided by the total number of conducted simulations N (all detections)

[7]In this simulated experiment, we do not consider the number of available resources. This is because each activity has its own dedicated resource, and all resources are consistently available to process tasks once they are completed with the current one.

for a given disruption size. It is defined as follows:

$$Acc := \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{I_i}(R_i^{act}), \quad \text{where} \quad \mathbf{1}_{I_i}(R_i^{act}) := \begin{cases} 1, & \text{if } R_i^{act} \in I_i, \\ 0, & \text{otherwise.} \end{cases}$$

The accuracy ranges between 0 and 1. The higher the value, the more accurate our approach is.

Results

Table 6.6 shows the results of this experiment, depicting the accuracy of our approach obtained for different disruption sizes in the number of case arrivals.

Table 6.6: Results of Experiment 1: accuracy for different disruption sizes.

Disruption size	**Moderate (5-10 std.)**	**Severe (10-15 std.)**	**Extreme (15-20 std.)**	**Average**
Time-to-impact	0.75	0.83	0.68	0.76
Recovery time	0.86	0.88	0.89	0.88
Maximal deviation	0.65	0.78	0.88	0.77
Cumulative perf. loss	0.70	0.72	0.72	0.71
Average	**0.74**	**0.80**	**0.79**	**0.78**

Overall, our approach demonstrates a satisfactory level of accuracy in estimating resilience measures, achieving an average accuracy across all disruption sizes and resilience measures of 0.78. With respect to different disruption sizes, the average accuracy ranges from 0.74 to 0.80 (without a clear trend), indicating consistency in accuracy largely regardless of the disruption's severity.

Concerning different resilience measures, the accuracy varies between 0.71 and 0.88. Notably, the approach achieves its highest accuracy of 0.88 for recovery time estimation, maintaining consistency across various disruption sizes. While the accuracy for cumulative performance loss also demonstrates consistency, it is the lowest among the resilience measures with an average accuracy of 0.71. However, this aligns with expectations, as the accuracy of cumulative performance loss is influenced by the combined accuracy of other resilience measures. The accuracy for the time-to-impact measure spans from 0.68 to 0.83, without any notable trend relative to disruption size. Lastly, maximal performance deviation demonstrates good accuracy, surpassing 0.78 for severe and extreme disruption sizes. However, it tends to slightly underestimate the severity of the impact in cases of severe disruption.

The results of this experiment thus show that our approach is able to accurately estimate the disruption size in a complex scenario. After this confirmation of the validity of our approach, we conduct the next experiment to show the effectiveness of our approach using real-life data.

6.5.2 Experiment 2: Approach effectiveness

After confirming the validity of our approach, we aim to showcase the effectiveness of our approach in evaluating business process resilience using the introduced four resilience measures. Since we do not have access to the actual resilience ground truth of the processes recorded in an event log, our evaluation centers on real-life event logs describing the execution of the same process across various organizations. This allows us to compare findings about process resilience, providing relative insights into strengths and weaknesses across different organizations and demonstrating the utility of our analysis.

Setup

Next, we discuss the setup and obtained results of the conducted experiment.

Data collection. To perform the experiment, we used data from the BPI Challenge 2015 [91], which consists of five real-world event logs (M1-M5), each capturing a permit application process at a Dutch municipality. The nature of these logs makes them highly suitable for our work because they allow us to assess and compare the resilience of the same process across different organizations, accounting for differences in its implementation. The event logs are comparable in terms of the covered period (about 5 years) and size, including numbers of events, event classes, and traces, as shown previously shown in Table 6.7. However, we also observe considerable differences in their lead times (averages ranging approximately from 9 to 23 weeks), which strongly fluctuate, as evidenced by the high standard deviation (from 14 to 24 weeks).

Table 6.7: Properties of the considered real-life event logs.

Event log	Period (wks)	N. of events	Event classes	N. of traces	Trace variants	Lead time Avg./SD (wks)
BPIC15 M1	252	52 217	398	1199	1170	14/17
BPIC15 M2	244	44 354	410	832	828	23/24
BPIC15 M3	270	59 681	383	1409	1349	9/14
BPIC15 M4	276	47 293	356	1053	1049	17/15
BPIC15 M5	275	59 083	389	1156	1153	14/15

Approach configuration. We use the following settings when applying our approach. In Step 1, we use a fixed window length to achieve comparability of resilience insights between municipalities. We select semi-monthly time windows because they were found to be the overall best window size across the logs, given their length and fluctuations in the process features. We test the same five percentiles for the warm-up and cool-down phase detection parameter δ, as in the first experiment. In Step 2, we fit VAR models with the same configurations for trend options, keeping again the significance level $\alpha = 0.05$ for the multivariate Ljung–Box portmanteau test. In Step 3, we use a tolerated deviation P_{tol} of 10% of the standard deviation of the PPI and consider a response horizon h_{max} equal to 50% of the window sequence length. Finally, we let each disruption correspond to an increase or decrease of two standard deviations of the respective feature (using the standard deviation across the time-series values).

Using our implementation with the specified configuration, the total computation time for this experiment on an Intel i7-9750H CPU (2.60GHz) with 16 GB RAM took about 40 minutes.

Experiments. We conducted two experiments to assess process resilience[8]:

Experiment 2.1: Overall resilience. We assess the overall resilience per municipality using the default resilience scenario described in Section 6.4.1. Specifically, we consider the average lead time as the PPI, i.e., $F_P = \{l\_t\}$, and disruptions in the numbers of case arrivals, active cases, and available resources, i.e., $F_D = \{\uparrow arr\_c, \uparrow act\_c, \downarrow avl\_r\}$. We do not consider additional features, i.e., $F_A = \emptyset$.

Experiment 2.2: Resilience per case type. Given that the process under investigation covers a broad range of permit applications, we use this experiment to examine if the process is specifically prone to increases in certain case types. To do this, we consider a resilience scenario with disruptions in terms of the three most common case types per municipality (based on a case's "parts" attribute), $F_D = \{\uparrow act\_c\_type_x, \uparrow act\_c\_type_y, \uparrow act\_c\_type_z\}$, where x, y, and z differ per municipality, due to varying commonality of the case types. Other scenario options are left the same as in Experiment 2.1, i.e., $F_P = \{l_t\}$ and $F_A = \emptyset$.

Results

In this section, we report on the results obtained from our experiments.

[8]In the second experiment with real-life data, we determine the average lead time by considering the lead time for each window of the last 3% of all cases completed before the end of that window to prevent situations where the lead time is inflated by a few lengthy cases completed within a specific window.

Experiment 2.1: Overall resilience. To contextualize resilience insights, Table 6.8 presents the main characteristics of diverse time series in terms of the average values, standard deviations, and coefficients of variation, obtained through semi-monthly windows. Here, we find that the processes are rather stable in terms of the available resources (avg. coefficient of variation of 19%) and active cases (22%) but fluctuate in terms of case arrivals (46%). Per municipality, we see considerable differences, with, e.g., M3 having much less fluctuation in its active cases (just 14%) than others (up to 32%).

Table 6.8: Statistics of the time series per each process feature from the default resilience scenario on a semi-monthly windowing.

Event log	**Case arrival**			**Active cases**			**Avl. resources**			**Lead time**		
	Avg	**Std**	**CV**	**Avg**	**Std**	**CV**	**Avg**	**Std**	**CV**	**Avg**	**Std**	**CV**
M1	11.3	5.6	49%	88.2	12.4	14%	7.4	1.7	22%	6.5	2.1	33%
M2	8.0	4.8	60%	94.9	21.0	22%	5.3	1.1	21%	9.7	3.4	35%
M3	13.2	4.9	37%	69.1	9.7	14%	6.7	0.8	12%	3.7	1.2	32%
M4	9.9	4.2	42%	91.4	25.3	28%	5.2	1.2	22%	7.4	2.3	31%
M5	10.9	4.7	43%	82.3	26.1	32%	8.1	1.3	16%	6.1	2.4	40%

Table 6.9 provides an overview of the overall resilience results. We observe that the processes have the worst resilience to increases in the active cases, with the longest average recovery time across all municipalities and the greatest average maximum increase in the average lead time, causing a notable cumulative performance loss. Although resilience to the other two disruptions is generally comparable in terms of maximal impact, the average recovery time after decreases in the number of available resources takes 40% longer compared to increases in the case arrival, which doubles the observed average cumulative performance loss.

Table 6.9: Results of Experiment 2.1, showing the resilience for the three disruption types (semi-monthly windows).

Disruption	**Increase in case arrivals**					**Increase in active cases**					**Decrease in avl. resources**				
Municipality	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5	M1	M2	M3	M4	M5
Time-to-impact	1	1	1	1	9	1	1	1	1	1	0	1	1	1	1
Recovery time	4	17	3	1	11	8	28	8	22	29	0	28	8	11	5
Max. deviation	0.5	0.6	0.7	0.4	0.2	0.3	2.5	0.4	1.2	0.9	0	0.9	0.4	0.8	0.5
Cum. perf. loss	1.4	5.7	1.3	0.4	2.6	2.2	25.8	2.3	15.9	16.4	0	13.5	2.2	5.6	1.8

When comparing the resilience among the municipalities, we also gain several interesting insights. Municipality M1 exhibits the highest overall resilience. It demonstrates strong resilience to disruptions in case arrivals and active cases. Furthermore, it is not anticipated to experience any performance drops from disruptions in available resources. In other words, the expected increase in the average lead time does not exceed the tolerated deviation P_{tol}. Municipality M3 also displays strong resilience; however, a potential disruption in available resources affects the lead time, compared to M1. In contrast, municipality M2 shows the worst resilience in all disruption scenarios, showing the highest cumulative performance loss across all municipalities. Finally, municipalities M4 and M5 exhibit relatively low resilience to an increase in the number of active cases, remaining competitive with other municipalities in other disruptions. Based on such insights, municipality M2 could conduct further analyses to understand why their resilience on the same process is so different compared to M1 and M3, as a basis for future improvement.

Experiment 2.2: Resilience per case type. Table 6.10 shows the resilience insights obtained for disruptions in the three common application types per municipality. To contextualize the resilience insights, we include the corresponding frequency and average lead time for each case type. We observe that the three most common application types account for 55 to 70 % of all cases. Interestingly, type *A* is most common for all municipalities, accounting for 40 to 50% of the total cases, and type *B* always is the second or third most common type. The lead times differ remarkably between case types and municipalities. For M1, for instance, type *A* takes about 11 weeks on average (5 semi-monthly windows), whereas cases of type *C* take 40 weeks (19 semi-monthly windows).

Based on the computed resilience measures, we observe several differences in resilience to the three most common case types compared to the overall resilience. First of all, we see that municipalities M1 and M3 show the highest resilience to all frequent case types. This aligns with insights from an overall analysis of resilience to disruptions in the number of active cases, depicted in Table 6.6. The resilience measures for common case types at M2 are comparable to the overall resilience in terms of recovery time and cumulative performance loss. However, municipalities M4 and M5 show remarkably lower resilience to increases in the three most common case types compared to the overall resilience. In both cases, the maximum impact doubles, and with a long recovery time reaching the maximum considered horizon, the cumulative loss is significant.

We also observe notable differences in resilience for the same case type across different municipalities. For example, resilience to the most common case type *A* is high at M1 and M3 but worse at other municipalities, especially M4. A similar trend is observed for type *B*, with lower resilience at M5. Looking at type *C* at M1 and M2, there's a notable difference in resilience despite the same case frequency

Table 6.10: Results of Experiment 2.2, showing process resilience (semi-monthly windows) per municipality to disruptions in the three most common application types and the corresponding statistics, obtained based on the initial event logs.

Municipality	**M1**			**M2**			**M3**			**M4**			**M5**		
Increase in case type	*A*	*B*	*C*	*A*	*B*	*C*	*A*	*B*	*D*	*A*	*B*	*E*	*A*	*B*	*D*
Case frequency (%)	50	8	5	39	11	7	39	19	6	44	15	9	50	6	10
Avg. lead time	5	5	19	9	4	20	4	2	6	8	6	7	5	6	7
Time-to-impact	1	1	2	2	1	5	3	3	1	1	4	1	1	1	1
Recovery time	8	12	14	36	31	45	10	4	26	52	49	52	50	50	50
Max. deviation	0.5	0.9	0.3	1.3	1.6	0.8	0.5	0.2	0.6	3.2	0.9	3.4	2.0	2.3	1.6
Cum. perf. loss	2	6	3	29	27	29	3	0.8	8	147	30	151	82	102	73

Case type descriptions: A: “Bouw”(“Construction"), B: “Kap” (“Felling (Cutting down trees)"), C: “Milieu (vergunning)” (“Environment (permit)"), D: “Bouw, Handelen in strijd met regels RO” (“Construction, Acting against spatial planning rules"), E: “Handelen in strijd met regels RO” (“Acting against spatial planning rules")

and average lead time. A similar trend is noted for type *D* in municipalities M3 and M5.

Overall, when considering resilience per case type and municipality, we see the importance of examining resilience by case type. The resilience insights obtained from this experiment expand upon the findings from the overall resilience analysis and can guide further investigation into improving process resilience in municipalities. For instance, it is worth investigating why the resilience to the most common case types at M4 and M5 differs significantly from their overall resilience or whether batching causes low resilience at M4.

6.6 Discussion

In this section, we first discuss the advantages of our approach for business process resilience assessment in comparison to the alternative solution. Then, we examine the limitations of our approach, identifying potential areas for improvement.

Advantages of the proposed solution. The proposed approach for resilience assessment offers several advantages that make it particularly well-suited for the task at hand. First, the VAR model estimates how process performance responds to disruptions using impulse-response analysis, which closely aligns with the idea of measuring resilience against disruptions (see Section 6.2). Moreover, the impulse-response analysis provides insights into the magnitude and duration of disruption impacts, enabling direct identification of all four resilience measures. Second, the

VAR model assumes linear relationships between relevant process features, which have been proven to provide reliable and accurate estimates of the resilience of a business process under the short-term impact of disruptions (see Section 6.5.1), even when process disruptions have not been previously observed in the data. Finally, our approach is automated, requiring only an event log and a user-defined resilience scenario as input, and can be used out-of-the-box.

Comparison with alternative solution. Our approach offers significant advantages over the alternative solution based on (auto-mined) process simulations. Concerning the latter, we note that constructing a simulation model that adequately replicates process behavior is highly complex and time-consuming [45, 145]. Simulation models discovered in an automated manner [146, 147] lack the functionality needed to simulate disruption scenarios essential for resilience assessment, requiring additional customization and hindering automated resilience assessment. In contrast, our approach eliminates the need to create simulation models from scratch or to modify automatically generated process simulations to include specific functionalities.

Limitations. The use of a VAR model to assess the resilience of a business process from event data also has limitations, which we aim to address in the future.

The linearity of the VAR model allows for robust estimation of the short-term impact of a disruption on process performance. However, it might lead to an underestimation of the impact if the disruption size is extremely large. This occurs because a process might behave non-linearly under extreme disruptions, making rescaling the detected response in process performance ineffective.

The first step in our approach generates process features as time series, which describe characteristics relevant for resilience assessment. Although these process features are typically stationary, maintaining a consistent mean and variance, they can exhibit significant changes due to factors such as concept drift. In such cases, preprocessing steps are necessary. For instance, concept drift detection can be applied to an event log to identify these changes before assessing the resilience of the process version of interest. Furthermore, if a process is highly seasonal, the time series may need to be de-seasonalized to eliminate the impact of seasonality on the resilience assessment results.

Finally, our approach employs the Akaike information criterion for selecting optimal model settings. Despite its acceptable results, this is a relatively simplistic strategy, whereas selecting (truly) optimal models is a complex task.

6.7 Related Work

Our work relates to resilience assessment approaches specific to business processes and some well-established problems in process mining. It also relates to broader research areas in other fields.

6.7.1 Process Resilience Assessment

In the following, we discuss the two main research directions in process resilience assessment and explore related problems within process mining.

Main research directions. The assessment of business process resilience diverges into two directions: resilience assessment at design-time and at run-time.

Resilience assessment at design-time. Resilience assessment at design-time refers to the evaluation of a process' resilience capabilities during the initial design phase. A recently proposed method introduces four levels of process model resilience based on a data-centric, collaboration-oriented modeling language for processes [130]. Based on this method, a resilience-aware maturity model for modeling multi-party business processes allows precise quantification of model compliance concerning the different resilience levels [115]: no resilience awareness, failure awareness, data resilience, milestone resilience, and process resilience.

Resilience assessment at run-time. Resilience assessment at run-time involves evaluating a process's ability to withstand disruptions and maintain performance during its operational phase, with a focus on monitoring its response to unexpected events. Our chapter focuses on data-driven resilience assessment at run-time, for which little research in business process management and process mining has been conducted so far. The only research that addresses this problem is the work by Zahoransky et al., who have presented a decision support framework [99]. This framework collects and quantifies metrics and indicators for assessing the run-time resilience of a process based on the ex-post analysis of event logs. They also conducted a case study [116], investigating the temporal aspect of process resilience, done by applying process mining to create probability distributions on the time behavior of business processes, using historic information in an event log. However, their approach is not generalizable because it depends on domain knowledge and manual intervention and requires a known process model for building a simulation model, making process-centered and data-driven resilience assessment impractical.

Connection to other problems in process mining. In the following discussion, we position the problem of assessing business process resilience alongside other well-established problems in process mining, highlighting the unique aspects of resilience assessment.

Anomaly detection. Anomalies can be defined as deviations in process behavior from normal or expected behavior according to an established process model [148]. Numerous supervised and unsupervised techniques have been proposed to detect these deviations within event logs, as highlighted in recent surveys [149, 150]. However, anomaly detection cannot be used to address the problem of process resilience assessment due to its different scope and approach level. Specifically, process resilience assessment aims to estimate the potential impact of future process disruptions, which indicate changes characteristics of a process at the system level. In contrast, anomaly detection focuses on identifying past deviations recorded in an event log, either at the trace level or within segments of traces.

Concept drift detection. Concept drift in process mining occurs when a process changes while being analyzed [106]. The objective of concept drift detection is to identify and describe these changes using data from an event log. A temporary disruption, such as a sudden increase in case arrivals, should not be mistaken for concept drift. Unlike concept drift, which indicates a shift to a new version of the process model, disruptions are temporal and are more linked to anomalies. Furthermore, concept drift detection focuses on detecting changes in the recorded data rather than predicting how a process might react to future disruptions. Therefore, resilience assessment presents a distinct challenge compared to concept drift detection.

Causal process mining. Causal process mining involves identifying and understanding the cause-and-effect relationships within different features of a business process. Causal analysis has been successfully used to automate the discovery of factors that impact process performance [151] or to mine process dependencies for control-flow decisions taken during the execution of process models [152].

The problem of resilience assessment can be viewed as a problem of causal analysis, where the disruption is the cause and the decline in process performance is the effect. However, in this context, identifying the cause and effect is redundant since they are already known, making the quantification of the impact the primary concern. Existing methods for causality detection, such as those based on Granger causality, can quantify the impact of a disruption on process performance. However, these methods do not consider the evolution of this impact over time or its relation to other process features that are also affected and may contribute to the performance decline. In contrast, a vector autoregressive model can address these challenges more effectively. It can analyze the dependencies between relevant process features in a single run, considering all features and their interrelations simultaneously. This allows for a comprehensive prediction of the response in process performance under various disruption scenarios, taking into account the complex interactions between different process features.

6.7.2 Resilience assessment in other research fields

The problem of resilience assessment is linked to various established research fields. In the following discussion, we consider these fields and discuss their differences in comparison to business process resilience assessment.

Risk assessment. Risk assessment is a mature discipline that has been developed in the past 40 years to help understand and control the risk of accident events [153]. Many principles and methods are developed for how to conceptualize, assess, and manage risk [154], i.e., the potential occurrence of undesirable consequences for certain entities or situations.

Risk assessment and resilience assessment complement each other [153]. Risk assessment focuses on identifying and managing potential risks, while resilience assessment focuses on the ability to withstand and recover from disruptions. Together, they provide a comprehensive approach to understanding and mitigating threats, ensuring both the prevention of issues and the capability to recover when they occur. However, existing data-driven techniques for risk assessment cannot describe the ability of a process to withstand and recover from possible disruptions, but process resilience assessment can enhance awareness of potential risks in business process management.

Business continuity management. The continuity of a business process and process resilience are closely related concepts that complement each other, though they differ slightly in scope and focus. Business continuity is defined as the ability of an organization to continue delivering products or services at acceptable levels after a disruption [125]. It primarily aims at ensuring immediate operational stability by developing specific, tactical plans and procedures to prevent disruptive events from causing unexpected, unwanted interruptions in production or service activities [153]. Business continuity focuses on building processes and procedures to navigate a single disturbance by establishing alternative suppliers, maintaining backup inventory, and creating detailed plans for switching suppliers if the primary supplier is unavailable. In contrast, process resilience emphasizes the process itself, aiming for long-term strength and the ability to handle any number of disruptions, even unforeseen ones, by integrating flexible manufacturing practices that enable process activities to be executed under various conditions.

Statistical quality control. Statistical quality control is a collection of tools and techniques useful in achieving quality improvement by establishing process stability [109]. The enhancement of process quality stems from the reduction of variability in its outcomes, implying that a decrease in variability in the critical characteristics of a product results in an increase in product quality.

Process resilience assessment and statistical process control share some similarities but differ in scope. Statistical process control aims to maintain process performance within acceptable levels and alert management when deviations occur, typically utilizing control charts that display upper and lower bounds for a given performance indicator. In contrast, process resilience not only addresses deviations from desired levels but also seeks to predict the expected impact and recovery time in the event of disruptions.

Sensitivity analysis. Sensitivity analysis evaluates how variations in input parameters affect a model or system's output [155, 156]. It is used to identify critical factors influencing outcomes, quantify their influence, and inform decision-making by highlighting areas where improvements or interventions may be most effective.

Still, sensitivity analysis provides an incomplete view of the resilience assessment problem. Existing sensitivity analysis methods, e.g., Sobol sensitivity analysis [157] and PRIM (Patient Rule Induction Method) analysis [158], allow quantification of the impact of changes in one variable on another. In resilience assessment, this can be used to assess the maximal expected impact on process performance based on the initial size of the disruption. However, they can only be used to estimate the anticipated maximum impact of a disruption, whereas they do not provide insights into the impact of a disruption over time, such as the time-to-impact or recovery time.

6.8 Conclusion

This chapter presented an approach for assessing process resilience using event log data. We measure process resilience in terms of the expected deviation of the process performance to disruptions in different process characteristics, such as arrival rate, active cases, and available resources. Our approach represents process characteristics as time series, constructs a vector autoregressive model that captures the statistical relationship between them, and conducts impulse-response analysis. Evaluation experiments demonstrate the accuracy and effectiveness of our approach in quantifying overall process resilience and its weak and strong points concerning different disruption types. The obtained insights help to understand the resilience of business processes, which is the first critical step towards achieving better resilience.

In future research, we plan to improve and further develop our work in several manners. First, our employed VAR models assume a linear relationship between the process features, which is not always the case. In our evaluation, we saw that the linear models generally performed well. Nevertheless, future research should investigate other VAR models, for instance, with time-varying coefficients or non-linearity. Second, our approach employs the Akaike information criterion for selecting optimal model settings. Despite its acceptable results, this is a relatively simplistic strategy, whereas selecting (truly) optimal models is a complex task. Therefore, in the future, we aim to investigate state-of-the-art selection strategies that best suit the objective of resilience assessment using event data. Third, we aim to improve the statistical properties of the generated time series by accounting for, e.g., trends with drifts and seasonal effects, which may yield more accurate models. Finally, we aim to study how countermeasures can mitigate the expected negative impact of a disruption on a PPI. We seek to determine the appropriate timing and amount of these countermeasures to keep the negative impact within defined boundaries.

Chapter 7

Conclusion

This chapter concludes the doctoral thesis. Section 7.1 summarizes the main results, while Section 7.2 discusses their implications for research and practice. Finally, Section 7.3 outlines possible directions for future research that build on the presented work.

7.1 Summary of the Results

In this thesis, we applied system-level process mining to gain an understanding of business process dynamics that we then used to address specific problems in process mining. To this end, we explored different forms of system-level process mining to uncover aspects of process dynamics that are relevant for the process mining tasks at hand. Given our contributions, we can summarize the main results presented in this thesis as follows:

1. *Comprehensive Concept Drift Characterization*: Business processes evolve in time, leading to concept drift in event logs. Detecting such drift is crucial for many process mining tasks to obtain reliable insights. Current techniques, however, mainly identify isolated changes and therefore fail to provide a complete understanding of process evolution. In Chapter 3, we showed how studying business process dynamics through system-level process mining enables a more comprehensive characterization of concept drifts. Specifically, we addressed the problem of comprehensive drift characterization by introducing a new taxonomy that emphasizes process changes and their interconnections. Building on this taxonomy, we presented a three-step framework that automatically characterizes concept drift from event logs using algorithms that employ behavioral similarity analysis of process

dynamics across different periods, thus supporting the characterization of both isolated changes and more complex drift types.

2. *Concept Drift Detection Using Computer Vision*: Business process dynamics can be used not only to enhance the characterization capability of existing concept drift detection techniques but also to enable more accurate detection of concept drifts. In Chapter 4, we proposed a novel approach that leverages process dynamics and machine learning to address the problem of concept drift detection in a new way. By introducing an image-based visualization of process dynamics over time, our approach applies supervised ML to learn how drifts manifest in event logs. Using a fine-tuned state-of-the-art computer vision model, the approach detects four drift types and outperforms existing solutions in accuracy and robustness to noise, demonstrating the potential of this new paradigm that combines process dynamics with machine learning.

3. *Steady-State Detection*: A business process is a complex system that continuously evolves over time. As a result, event logs often contain data from both steady and non-steady states. Distinguishing between these states is a crucial task in process mining, as ignoring them can distort results such as performance analysis and remaining time prediction. The analysis of process dynamics over time makes it possible to study steady and non-steady states in business processes. In Chapter 5, we highlighted the importance of SSD in the process mining domain and proposed a two-step framework for detecting steady states. This framework allowed us to investigate the applicability of existing SSD solutions in the context of process mining. The findings emphasize the importance and potential of SSD for obtaining more accurate process mining insights.

4. *Process Resilience Assessment*: Process resilience refers to the ability of a process to return to an acceptable performance level after disturbances and is a critical competence for organizations facing disruptions such as sudden workload increases or workforce absences. Despite its importance, few studies assess resilience in a data-driven manner, limiting the ability of organizations to understand how strongly their processes are affected by disruptions and how long recovery takes. In Chapter 6, we presented an approach for automated resilience assessment. It builds on different characteristics that capture process dynamics over time and applies a statistical model to analyze their interrelations and evaluate how process performance responds to both observed and unseen disruptions. We validated the approach against

simulation-based “what-if” analyses and demonstrate its effectiveness in assessing process resilience under diverse disruptions across organizations.

When developing and evaluating our approaches, we followed established practices from algorithm-engineering research (Section 1.3). By ensuring ontological clarity, maintaining epistemological precision, and addressing threats to methodological validity, we provided sound contributions that substantially advance the body of knowledge in the field of process mining.

7.2 Implications

This section discusses the implications of the work presented in this thesis, with Section 7.2.1 focusing on practice and Section 7.2.2 on research.

7.2.1 Implications for Practice

The work presented in this thesis carries several implications for practitioners in organizations that apply process mining to analyze their processes. The primary implication is that our research supports the generation of more accurate insights across a wide range of process mining activities. The following sections outline the specific implications of our research for the key process mining tasks most commonly applied in practice:

- *Achieving more accurate process discovery.* Process discovery is one of the main process mining activities in practice. However, the insights obtained can be highly misleading if event logs contain data from different versions of the same process. Our work on concept drift detection and characterization, presented in Chapter 3 and Chapter 4, enables more precise identification of different process versions by enhancing both accuracy and robustness to noise. As a result, practitioners can gain more reliable insights into different process versions and their evolution over time by applying process discovery only to periods that cover the process version of interest.
- *Proposing a new bucketing strategy.* Our work on steady-state detection (Chapter 5) introduces a new bucketing strategy in process mining. For practitioners, it is applicable to many use cases where it is essential to divide event data into meaningful segments, such as:
  - *Process performance analysis.* Performance analysis is a core process mining activity frequently applied in practice. Similar to process discovery, it can lead to misleading insights if proper pre-analysis is not

conducted. In particular, performance analysis may become unreliable when practitioners ignore the distinction between steady and non-steady states of a business process, because process performance can vary significantly between these states, making such a distinction essential for accurate performance analysis. With our proposed approach for steady-state detection, it becomes possible to differentiate performance between periods of steady states, when a process operates under normal conditions, and non-steady states, when it is subject to continuous disruptions.

  – *Predictive process monitoring.* Predictive process monitoring often involves bucketing before training predictive models in order to obtain more accurate predictions. By applying our framework, event logs can be split into sublogs that represent steady and non-steady states of a process. These sublogs can then be used in predictive process monitoring, where separate models are trained to predict the remaining time of ongoing cases under different conditions. The resulting models can then be applied to ongoing cases by selecting the model that corresponds to the current state of the business process, thereby improving the accuracy of predictions.

- *Enabling the measurement of business process resilience.* Organizations aim to be resilient in order to survive turbulent times, and organizational resilience begins with the resilience of core business processes. To understand business process resilience, the first step is to measure it. Our approach (Chapter 6) presents the first automated approach to assess business process resilience in a data-driven manner. Using historical event data, organizations can evaluate the resilience of their processes through four proposed resilience measures. These measures help organizations assess the resilience of their processes, determine which processes require improvement, and identify the resilience aspects that should be addressed first through the design of interventions that strengthen process resilience.

7.2.2 Implications for Research

Our work enables and informs several research directions in the field of process mining. In particular, our contributions demonstrate the usefulness and added value of studying process dynamics through system-level process mining, with implications for research on:

- *Concept drift detection and characterization.* Concept drift detection is a well-established problem in process mining, and numerous techniques have been proposed to address it. However, in recent years, research in this area has slowed down, with limited progress in developing new approaches to overcome the persistent trade-off between accuracy and robustness to noise. Our research on concept drift detection (Chapter 3 and Chapter 4) gives this problem new momentum. We introduced a new taxonomy and proposed the first supervised machine learning–based approach that directly learns how drifts manifest in event logs. Our approach outperforms existing solutions by improving both accuracy and robustness to noise, thereby introducing a new use case for machine learning in process mining and extending the state of the art.

- *Steady-state detection.* The problem of steady-state detection has received no attention in process mining so far, even though it can significantly affect the accuracy of many process mining activities. In our work (Chapter 5), we demonstrated the importance of SSD in process mining and proposed a framework to detect steady and non-steady periods in business processes. This opens a new research direction in process mining with a novel problem definition and a baseline solution.

- *Process resilience assessment.* Research on process resilience has received little attention in process mining and is still in its early stages. The only tangible approach so far has been to manually build business process simulation models and evaluate disruption scenarios to estimate process resilience. This is typically time-consuming, and although recent years have seen progress in automating simulation modeling, existing solutions still lack the necessary functionality for resilience assessment. In our work, we formalized the concept of resilience for business processes and proposed an automated approach to assess process resilience directly from event logs without the need to build simulation models and conduct extensive "what-if" analyses. In this way, our research highlights the importance of process resilience and provides a baseline solution that can serve as a starting point for more advanced approaches in the future.

7.3 Future Research

This work opens several promising directions for future research. They offer opportunities to study business process dynamics more deeply and to apply them to various process mining tasks in order to improve accuracy and broaden the range and quality of insights. In particular, we see the greatest potential in the following directions:

- *Extending the concept of process dynamics.* This thesis presents an initial attempt to formalize the idea of process dynamics within process mining and should be regarded as a starting point. Future research should aim to further refine this concept and clarify its relevant aspects to establish a stronger conceptual and methodological grounding. This may involve developing formal definitions, models, taxonomies, and categorizations of process dynamics. It is also important to investigate how process dynamics emerge, evolve, and interact, and to distinguish between different types of dynamics, such as structural, behavioral, and performance-related. A deeper theoretical understanding would not only sharpen the notion of process dynamics but also open new opportunities for systematically exploring its applicability across diverse process mining tasks.

- *Exploring data sources and forms for system-level process mining.* In this thesis, system-level process mining was applied to capture process dynamics in the form of time series and images derived from event logs. Future research should investigate opportunities beyond these representation forms and data types. For example, alternative formats such as graphs or multidimensional data structures (e.g., 3D cubes) may provide richer ways to represent the temporal and structural aspects of high-level process dynamics. Moreover, combining event logs with complementary data sources, such as resource calendars, sensor data, weather information, or geolocation, could yield a more comprehensive view of process dynamics within their operational context. Finally, system-level process mining should be explored for other data types, such as object-centric event logs, to analyze how different objects contribute to overall process dynamics.

- *Broadening applications of process dynamics and system-level process mining.* In this thesis, we demonstrate how process dynamics can support three distinct tasks, i.e., concept drift detection and characterization, steady-state detection, and resilience assessment, which extend the limited set of existing applications discussed in Section 2.2.3. We believe that many more tasks in process mining could benefit from, or even require, system-level process mining and the study of process dynamics. For example, beyond traditional tasks such as process discovery, conformance checking, and predictive process monitoring, the study of process dynamics through system-level process mining can also provide deeper insights into interconnected processes within the same organization. It can enable the analysis of how multiple processes interact (e.g., supply chains) and supports the identification of dependencies across organizations.

Bibliography

[1] Marlon Dumas, Marcello La Rosa, Jan Mendling, Hajo A. Reijers, et al. *Fundamentals of business process management*. Vol. 2. Springer, 2018.

[2] Wil van der Aalst. *Process Mining: Data Science in Action*. Springer, 2016.

[3] Zahra Toosinezhad, Dirk Fahland, Özge Köroğlu, and Wil van der Aalst. "Detecting System-Level Behavior Leading To Dynamic Bottlenecks". In: *International Conference on Process Mining (ICPM)*. IEEE, 2020, pp. 17–24.

[4] Bianka Bakullari and Wil van der Aalst. "High-level event mining: A framework". In: *International Conference on Process Mining (ICPM)*. IEEE. 2022, pp. 136–143.

[5] Arik Senderovich, Chiara Di Francescomarino, Chiara Ghidini, Kerwin Jorbina, and Fabrizio Maria Maggi. "Intra and inter-case features in predictive process monitoring: A tale of two dimensions". In: *International Conference on Business Process Management (BPM)*. Springer. 2017, pp. 306–323.

[6] Mahsa Pourbafrani, Sebastiaan van Zelst, and Wil van der Aalst. "Supporting automatic system dynamics model generation for simulation in the context of process mining". In: *International Conference on Business Information Systems (BIR)*. Springer. 2020, pp. 249–263.

[7] Alexander Kraus and Han van der Aa. "Looking for Change: A Computer Vision Approach for Concept Drift Detection in Process Mining". In: *International Conference on Business Process Management (BPM)*. Springer, 2024, pp. 273–290.

[8] Alexander Kraus and Han Van der Aa. "Machine Learning-based Detection of Concept Drifts in Business Processes." In: *Process Science* 2, 5 (2025).

[9] Alexander Kraus and Han van der Aa. "Comprehensive characterization of concept drifts in process mining". In: *Information Systems* 135, 102584 (2026).

[10] Alexander Kraus, Keyvan Amiri Elyasi, and Han van der Aa. "On the Use of Steady-State Detection for Process Mining: Achieving More Accurate Insights". In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 2025, pp. 204–220.

[11] Alexander Kraus, Jana-Rebecca Rehse, and Han van der Aa. "Data-driven assessment of business process resilience". In: *Process Science* 1, 4 (2024).

[12] Justus Grimm, Alexander Kraus, and Han van der Aa. "CDLG: A Tool for the Generation of Event Logs with Concept Drifts". In: *Workshop Proceedings of the International Conference on Business Process Management (BPM)*. Vol. 3216. CEUR-WS. 2022, pp. 92–96.

[13] Jan Mendling, Henrik Leopold, Henning Meyerhenke, and Benoît Depaire. "Methodology of algorithm engineering". In: *Preprint arXiv:2310.18979* (2023).

[14] Gijs A. Holleman, Ignace T.C. Hooge, Chantal Kemner, and Roy S. Hessels. "The 'real-world approach'and its problems: A critique of the term ecological validity". In: *Frontiers in Psychology* 11, 721 (2020).

[15] Jana-Rebecca Rehse, Sander Leemans, Peter Fettke, and Jan Martijn E. M. van der Werf. "On Process Discovery Experimentation: Addressing the Need for Research Methodology in Process Discovery". In: *ACM Transactions on Software Engineering and Methodology* 34.1 (2024).

[16] Allen S. Lee and Richard L. Baskerville. "Generalizing generalizability in information systems research". In: *Information Systems Research* 14.3 (2003), pp. 221–243.

[17] Scott W. O'Leary-Kelly and Robert J. Vokurka. "The empirical assessment of construct validity". In: *Journal of Operations Management* 16.4 (1998), pp. 387–405.

[18] Norman Fenton and James Bieman. *Software Metrics: A Rigorous and Practical Approach*. 3rd ed. CRC Press, 2014.

[19] Thomas D. Cook, Donald Thomas Campbell, and William Shadish. *Experimental and quasi-experimental designs for generalized causal inference*. Vol. 1195. Houghton, Mifflin and Company, 2002.

[20] Paul C. Cozby. *Methods in behavioral research*. 10th ed. McGraw-Hill Education, 2009.

[21] José Hernández-Orallo. "Evaluation in artificial intelligence: from task-oriented to ability-oriented measurement". In: *Artificial Intelligence Review* 48.3 (2017), pp. 397–447.

[22] Wil van der Aalst and Josep Carmona. *Process mining handbook*. Springer, 2022.

[23] Project Management Institute. *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. 7th ed. Project Management Institute, 2021.

[24] Eric Almquist, John Senior, and Nicolas Bloch. “The elements of value”. In: *Harvard business review* 94.9 (2016), pp. 47–53.

[25] Wil Van der Aalst. “Data scientist: The engineer of the future”. In: *Enterprise interoperability VI: Interoperability for agility, resilience and plasticity of collaborations*. Springer, 2014, pp. 13–26.

[26] Jan vom Brocke, Wil van der Aalst, et al. “Process science: the interdisciplinary study of socio-technical change”. In: *Process Science* 1, 1 (2024).

[27] Michael Rosemann, Jan vom Brocke, Amy Van Looy, and Flavia Santoro. “Business process management in the age of AI–three essential drifts”. In: *Information Systems and e-Business Management* 22.3 (2024), pp. 415–429.

[28] Mathias Weske. *Business Process Management: Concepts, Languages, Architectures*. 4th ed. Springer, 2024.

[29] Sergey Smirnov, Matthias Weidlich, and Jan Mendling. “Business process model abstraction based on behavioral profiles”. In: *International Conference on Service-Oriented Computing (ICSOC)*. Springer. 2010, pp. 1–16.

[30] Mark Von Rosing, Stephen White, Fred Cummins, and Henk De Man. *Business Process Model and Notation - BPMN*. OMG, 2015.

[31] R.P. Jagadeesh Chandra Bose, Wil Van Der Aalst, Indrė Žliobaitė, and Mykola Pechenizkiy. “Dealing with concept drifts in process mining”. In: *Transactions on Neural Networks and Learning Systems* 25.1 (2013), pp. 154–171.

[32] IEEE Task Force on Process Mining. *XES Standard Definition*. `http://www.xes-standard.org/`. 2016.

[33] Anahita Farhang Ghahfarokhi, Gyunam Park, Alessandro Berti, and Wil van der Aalst. “OCEL: a standard for object-centric event logs”. In: *European Conference on Advances in Databases and Information Systems (ADBIS)*. Springer. 2021, pp. 169–175.

[34] Wil Van der Aalst, Ton Weijters, and Laura Maruster. “Workflow mining: Discovering process models from event logs”. In: *IEEE transactions on knowledge and data engineering* 16.9 (2004), pp. 1128–1142.

[35] Sergey Smirnov, Matthias Weidlich, and Jan Mendling. “Business process model abstraction based on synthesis from well-structured behavioral profiles”. In: *International Journal of Cooperative Information Systems* 21.1 (2012), pp. 55–83.

[36] Wil van Der Aalst, Maja Pesic, and Helen Schonenberg. “Declarative workflows: Balancing between flexibility and support”. In: *Computer Science-Research and Development* 23 (2009), pp. 99–113.

[37] Ludwig von Bertalanffy. *General System Theory: Foundations, Development, Applications*. 1st ed. George Braziller, 1968.

[38] Peter Checkland. “Systems thinking, systems practice”. In: *European Journal of Operational Research* 11.4 (1982), pp. 405–407.

[39] Diana Wright and Donella H. Meadows. *Thinking in systems: a primer*. Taylor and Francis, 2012.

[40] John Sterman. “System Dynamics: systems thinking and modeling for a complex world”. In: *Engineering Systems Division (ESD) Working Paper Series* (2002).

[41] Jiale Zheng, Chunhui Zhao, and Furong Gao. “Retrospective comparison of several typical linear dynamic latent variable models for industrial process monitoring”. In: *Computers & Chemical Engineering* 157, 107587 (2022).

[42] Philippe Esling and Carlos Agon. “Time-series data mining”. In: *ACM Computing Surveys (CSUR)* 45.1 (2012), pp. 1–34.

[43] Arik Senderovich, Chiara Di Francescomarino, and Fabrizio Maria Maggi. “From knowledge-driven to data-driven inter-case feature encoding in predictive process monitoring”. In: *Information Systems* 84 (2019), pp. 255–264.

[44] Marlon Dumas. “Constructing digital twins for accurate and reliable what-if business process analysis.” In: *Workshop Proceedings of the International Conference on Business Process Management (BPM)*. Vol. 2938. CEUR-WS. 2021, pp. 23–27.

[45] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo Reijers. *Fundamentals of business process management*. Vol. 2. Springer, 2018.

[46] Anne Rozinat, Ronny S. Mans, Minseok Song, and Wil van der Aalst. “Discovering simulation models”. In: *Information systems* 34.3 (2009), pp. 305–327.

[47] Lukas Kirchdorfer, Robert Blümel, Timotheus Kampik, Han van der Aa, and Heiner Stuckenschmidt. “Discovering multi-agent systems for resource-centric business process simulation”. In: *Process Science* 2, 4 (2025).

[48] Manuel Camargo, Marlon Dumas, and Oscar González-Rojas. “Learning accurate business process simulation models from event logs via automated process discovery and deep learning”. In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer. 2022, pp. 55–71.

[49] Orlenys López-Pintado and Marlon Dumas. “Business process simulation with differentiated resources: Does it make a difference?” In: *International Conference on Business Process Management (BPM)*. Springer. 2022, pp. 361–378.

[50] Wil Van Der Aalst, Arya Adriansyah, Ana Karla Alves De Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, R.P. Jagadeesh Chandra Bose, Peter Van Den Brand, Ronald Brandtjen, Joos Buijs, et al. “Process mining manifesto”. In: *Workshop Proceedings of the International Conference on Business Process Management (BPM)*. Springer. 2011, pp. 169–194.

[51] Jan Niklas Adams, Sebastiaan van Zelst, Lara Quack, Kathrin Hausmann, Wil van der Aalst, and Thomas Rose. “A framework for explainable concept drift detection in process mining”. In: *International Conference on Business Process Management (BPM)*. Springer. 2021, pp. 400–416.

[52] Alexander Seeliger, Timo Nolle, and Max Mühlhäuser. “Detecting concept drift in processes using graph metrics on process graphs”. In: *Conference on Subject-Oriented Business Process Management (S-BPM)*. Vol. 9. 6. ACM, 2017, pp. 1–10.

[53] Eva L. Klijn, Felix Mannhardt, and Dirk Fahland. “Multi-perspective concept drift detection: Including the actor perspective”. In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer. 2024, pp. 141–157.

[54] J. Martjushev, R.P. Jagadeesh Chandra Bose, and Wil Van Der Aalst. “Change point detection and dealing with gradual and multi-order dynamics in process mining”. In: *International Conference on Business Informatics Research (BIR)*. Springer. 2015, pp. 161–178.

[55] Abderrahmane Maaradji, Marlon Dumas, Marcello La Rosa, and Alireza Ostovar. “Detecting sudden and gradual drifts in business processes from execution traces”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.10 (2017), pp. 2140–2154.

[56] Anton Yeshchenko, Claudio Di Ciccio, Jan Mendling, and Artem Polyvyanyy. “Visual drift detection for event sequence data of business processes”. In: *IEEE Transactions on Visualization and Computer Graphics* 28.8 (2021), pp. 3050–3068.

[57] Jan Niklas Adams, Cameron Pitsch, Tobias Brockhoff, and Wil van der Aalst. “An Experimental Evaluation of Process Concept Drift Detection”. In: *Proceedings of the VLDB Endowment* 16.8 (2023), pp. 1856–1869.

[58] Bianka Bakullari, Jules van Thoor, Dirk Fahland, and Wil van der Aalst. “The interplay between high-level problems and the process instances that give rise to them”. In: *International Conference on Business Process Management (BPM)*. Springer. 2023, pp. 145–162.

[59] Ghada Elkhawaga, Mervat Abuelkheir, Sherif I. Barakat, Alaa M. Riad, and Manfred Reichert. “CONDA-PM: a systematic review and framework for concept drift analysis in process mining”. In: *Algorithms* 13.7, 161 (2020).

[60] Denise Maria Vecino Sato, Sheila Cristiana De Freitas, Jean Paul Barddal, and Edson Emilio Scalabrin. “A Survey on Concept Drift in Process Mining”. In: *ACM Computing Surveys* 54.9 (2021), pp. 1–38.

[61] Tobias Brockhoff, Merih Seran Uysal, and Wil van der Aalst. “Time-aware concept drift detection using the earth mover’s distance”. In: *International Conference on Process Mining (ICPM)*. IEEE. 2020, pp. 33–40.

[62] Canbin Zheng, Lijie Wen, and Jianmin Wang. “Detecting process concept drifts from event logs”. In: *OTM Confederated International Conferences “On the Move to Meaningful Internet Systems"*. Springer. 2017, pp. 524–542.

[63] Leilei Lin, Lijie Wen, Li Lin, Jisheng Pei, and Hedong Yang. “LCDD: Detecting business process drifts based on local completeness”. In: *IEEE Transactions on Services Computing* 15.4 (2020), pp. 2086–2099.

[64] Hoang Nguyen, Marlon Dumas, Marcello La Rosa, and Arthur H.M. ter Hofstede. “Multi-perspective comparison of business process variants based on event logs”. In: *International Conference “Conceptual Modeling” (ER)*. Springer. 2018, pp. 449–459.

[65] Bart F.A. Hompes, Joos Buijs, Wil van der Aalst, Prabhakar M. Dixit, and Johannes Buurman. “Detecting changes in process behavior using comparative case clustering”. In: *Workshop Proceedings of the International Symposium on Data-driven Process Discovery and Analysis (SIMPDA)*. Springer. 2015, pp. 54–75.

[66] Alfredo Bolt, Wil van der Aalst, and Massimiliano De Leoni. “Finding Process Variants in Event Logs”. In: *OTM Confederated International Conferences “On the Move to Meaningful Internet Systems"*. Springer. 2017, pp. 45–52.

[67] Xixi Lu, Dirk Fahland, Frank J.H.M. van den Biggelaar, and Wil van der Aalst. “Detecting deviating behaviors without models”. In: *Workshop Proceedings of the International Conference on Business Process Management (BPM)*. Springer. 2016, pp. 126–139.

[68] Douglas Curran-Everett and Calvin L Williams. “Explorations in statistics: the analysis of change”. In: *Advances in Physiology Education* 39.2 (2015), pp. 49–54.

[69] Fabian Rösel, Stephan A. Fahrenkog-Petersen, Han van der Aa, and Matthias Weidlich. “A distance measure for privacy-preserving process mining based on feature learning”. In: *International Conference on Business Process Management (BPM)*. Springer. 2021, pp. 73–85.

[70] Toon Jouck and Benoît Depaire. “PTandLogGenerator: a Generator for Artificial Event Data”. In: *Demonstration Proceedings of the International Conference on Business Process Management (BPM)*. Vol. 1789. 2016, pp. 23–27.

[71] Han van der Aa, Adrian Rebmann, and Henrik Leopold. “Natural language-based detection of semantic execution anomalies in event logs”. In: *Information Systems* 102, 101824 (2021).

[72] Alessandro Berti, Sebastiaan van Zelst, and Daniel Schuster. “PM4Py: a process mining library for Python”. In: *Software Impacts* 17, 100556 (2023).

[73] Neo Christopher Chung, Błażej Miasojedow, Michał Startek, and Anna Gambin. “Jaccard/Tanimoto similarity test and estimation methods for biological presence-absence data”. In: *BMC Bioinformatics* 20.15, 644 (2019).

[74] J. Bijsterbosch and A. Volgenant. “Solving the rectangular assignment problem and applications”. In: *Annals of Operations Research* 181 (2010), pp. 443–462.

[75] David F. Crouse. “On implementing 2D rectangular assignment algorithms”. In: *IEEE Transactions on Aerospace and Electronic Systems* 52.4 (2016), pp. 1679–1696.

[76] Claudio Di Ciccio and Massimo Mecella. “On the discovery of declarative control flows for artful processes”. In: *Transactions on Management Information Systems (TMIS)* 5.4 (2015), pp. 1–37.

[77] Vincenzo Pasquadibisceglie, Annalisa Appice, Giovanna Castellano, Donato Malerba, and Giuseppe Modugno. “ORANGE: outcome-oriented predictive process monitoring based on image encoding and CNNs”. In: *IEEE Access* 8 (2020), pp. 184073–184086.

[78] Peter Pfeiffer, Johannes Lahann, and Peter Fettke. “Multivariate business process representation learning utilizing gramian angular fields and convolutional neural networks”. In: *International Conference on Business Process Management (BPM)*. Springer. 2021, pp. 327–344.

[79] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. “Deep learning for generic object detection: A survey”. In: *International Journal of Computer Vision* 128 (2020), pp. 261–318.

[80] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. “Focal loss for dense object detection”. In: *International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2980–2988.

[81] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. “Microsoft COCO: Common objects in context”. In: *European Conference on Computer Vision (ECCV)*. Vol. 8693. Springer. 2014, pp. 740–755.

[82] Alireza Ostovar, Abderrahmane Maaradji, Marcello La Rosa, Arthur H. M. ter Hofstede, and Boudewijn van Dongen. “Detecting Drift from Event Streams of Unpredictable Business Processes”. In: *International Conference “Conceptual Modeling” (ER)*. Springer, 2016, pp. 330–346.

[83] Paolo Ceravolo, Gabriel Marques Tavares, Sylvio Barbon Junior, and Ernesto Damiani. “Evaluation Goals for Online Process Mining: A Concept Drift Perspective”. In: *IEEE Transactions on Services Computing* 15.4 (2022), pp. 2473–2489.

[84] Fremont E. Kast and James E. Rosenzweig. “General systems theory: Applications for organization and management”. In: *Academy of Management Journal* 15.4 (1972), pp. 447–465.

[85] Lang Dai, Tianyu Liu, Zhongyong Liu, Lisa Jackson, Paul Goodall, Changqing Shen, and Lei Mao. "An improved deep learning model for online tool condition monitoring using output power signals". In: *Shock and Vibration* 2020.1, 8843314 (2020).

[86] Zoltan Derzsi. "Optimal approach for signal detection in steady-state visual evoked potentials in humans using single-channel eeg and stereoscopic stimuli". In: *Frontiers in Neuroscience* 15, 600543 (2021).

[87] Nicholas W. Tschoegl. *Fundamentals of equilibrium and steady-state thermodynamics*. Elsevier, 2000.

[88] David L. Nelson and Michael M. Cox. *Lehninger Principles of Biochemistry*. 5th ed. W. H. Freeman, 2008. ISBN: 978-0-7167-7108-1. URL: https://www.amazon.ca/Lehninger-Principles-Biochemistry-Albert/dp/071677108X.

[89] Esa Ranta, Per Lundberg, and Veijo Kaitala. *Ecology of Populations*. Cambridge University Press, 2005.

[90] Herman E. Daly. "The economics of the steady state". In: *The American Economic Review* 64.2 (1974), pp. 15–21.

[91] Boudewijn van Dongen. "BPI Challenge". In: *Workshop Proceedings on Business Process Intelligence (BPI)*. 4TU ResearchData. Dataset. 2015.

[92] Mahsa Pourbafrani and Wil van der Aalst. "Discovering system dynamics simulation models using process mining". In: *IEEE Access* 10 (2022), pp. 78527–78547.

[93] Maxim Vidgof, Bastian Wurm, and Jan Mendling. "The Impact of Process Complexity on Process Performance: A Study using Event Log Data". In: *International Conference on Business Process Management (BPM)*. Springer. 2023, pp. 413–429.

[94] Eric Zivot and Jiahui Wang. *Modeling financial time series with S-PLUS*. Vol. 2. Springer, 2006.

[95] Fredrik Gustafsson. *Adaptive filtering and change detection*. Vol. 1. John Wiley & Sons, 2000.

[96] R. Russell Rhinehart. "Automated steady and transient state identification in noisy processes". In: *American Control Conference*. IEEE. 2013, pp. 4477–4493.

[97] Kaylea Haynes, Paul Fearnhead, and Idris A. Eckley. "A computationally efficient nonparametric approach for changepoint detection". In: *Statistics and Computing* 27 (2017), pp. 1293–1305.

[98] Alan V. Oppenheim. *Discrete-time signal processing*. Pearson Education India, 1999.

[99] Richard M. Zahoransky, Christian Brenig, and Thomas Koslowski. "Towards a process-centered resilience framework". In: *International Conference on Availability, Reliability and Security (ARES)*. IEEE. 2015, pp. 266–273.

[100] Geraint I. Palmer, Vincent A. Knight, Paul R. Harper, and Asyl L. Hawa. "CIW: An open-source discrete event simulation library". In: *Journal of Simulation* 13.1 (2019), pp. 68–82.

[101] Mattan S. Ben-Shachar, Indrajeet Patil, Rémi Thériault, Brenton M. Wiernik, and Daniel Lüdecke. "Phi, Fei, Fo, Fum: Effect Sizes for Categorical Data That Use the Chi-Squared Statistic". In: *Mathematics* 11.9 (2023).

[102] Evren Mert Turan and Johannes Jäschke. "A simple two-parameter steady-state detection algorithm: Concept and experimental validation". In: *Computer Aided Chemical Engineering*. Vol. 52. Elsevier, 2023, pp. 1765–1770.

[103] Niek Tax, Ilya Verenich, Marcello La Rosa, and Marlon Dumas. "Predictive Business Process Monitoring with LSTM Neural Networks". In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer, 2017, pp. 477–492.

[104] Efrén Rama-Maneiro, Juan C. Vidal, and Manuel Lama. "Deep Learning for Predictive Business Process Monitoring: Review and Benchmark". In: *IEEE Transactions on Services Computing* 16.1 (2023), pp. 739–756.

[105] Keyvan Amiri Elyasi, Han van der Aa, and Heiner Stuckenschmidt. "PGTNet: A Process Graph Transformer Network for Remaining Time Prediction of Business Process Instances". In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer. 2024, pp. 124–140.

[106] R.P. Jagadeesh Chandra Bose, Wil van der Aalst, Indrė Žliobaitė, and Mykola Pechenizkiy. "Handling concept drift in process mining". In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer. 2011, pp. 391–405.

[107] Mahsa Pourbafrani, Niels Lücking, Matthieu Lucke, and Wil van der Aalst. "Steady State Estimation for Business Process Simulations". In: *International Conference on Business Process Management (BPM)*. Springer. 2023, pp. 178–195.

[108] Jonghyeon Ko and Marco Comuzzi. “A Systematic Review of Anomaly Detection for Business Process Event Logs”. In: *Business & Information Systems Engineering* 65.4 (2023), pp. 441–462.

[109] Douglas C. Montgomery. *Introduction to statistical quality control*. 8th ed. John Wiley & Sons, 2019.

[110] Yossi Sheffi. “Resilience reduces risk”. In: *Logistics Quarterly* 12.1 (2006), pp. 12–14.

[111] Alessandro Annarelli and Fabio Nonino. “Strategic and operational management of organizational resilience: Current state of research and future directions”. In: *Omega* 62 (2016), pp. 1–18.

[112] Günter Müller, Thomas Koslowski, and Rafael Accorsi. “Resilience-a new research field in business information systems?” In: *International Conference on Business Information Systems (BIS)*. Springer. 2013, pp. 3–14.

[113] Pedro Antunes and Hernâni Mourão. “Resilient business process management: framework and services”. In: *Expert Systems with Applications* 38.2 (2011), pp. 1241–1254.

[114] E. P. Dalziell and S. T. McManus. “Resilience, Vulnerability, and Adaptive Capacity: Implications for System Performance”. In: *Proceedings of the International Forum for Engineering Decision Making (IFED)*. International Forum for Engineering Decision Making. 2004.

[115] Andrea Marrella, Massimo Mecella, Barbara Pernici, and Pierluigi Plebani. “A design-time data-centric maturity model for assessing resilience in multi-party business processes”. In: *Information Systems* 86 (2019), pp. 62–78.

[116] Richard M. Zahoransky, Thomas Koslowski, and Rafael Accorsi. “Toward resilience assessment in business process architectures”. In: *International Conference on Computer Safety, Reliability, and Security (SafeComp)*. Springer. 2014, pp. 360–370.

[117] Raquel Sanchis, Luca Canetta, and Raúl Poler. “A conceptual reference framework for enterprise resilience enhancement”. In: *Sustainability* 12.4, 1464 (2020).

[118] Seyedmohsen Hosseini, Kash Barker, and Jose E. Ramirez-Marquez. “A review of definitions and measures of system resilience”. In: *Reliability Engineering & System Safety* 145 (2016), pp. 47–61.

[119] Ran Bhamra, Samir Dani, and Kevin Burnard. “Resilience: the concept, a literature review and future directions”. In: *International Journal of Production Research* 49.18 (2011), pp. 5375–5393.

[120] Yacov Y. Haimes. “On the definition of resilience in systems”. In: *Risk Analysis: An International Journal* 29.4 (2009), pp. 498–501.

[121] Michael Ungar. “Qualitative contributions to resilience research”. In: *Qualitative social work* 2.1 (2003), pp. 85–102.

[122] Sophie Sarre, Cara Redlich, Anthea Tinker, Euan Sadler, Ajay Bhalla, and Christopher McKevitt. “A systematic review of qualitative studies on adjusting after stroke: lessons for the study of resilience”. In: *Disability and rehabilitation* 36.9 (2014), pp. 716–726.

[123] Muhammedamin Hussen Saad, Geoffrey Hagelaar, Gerben Van Der Velde, and SWF Omta. “Conceptualization of SMEs’ business resilience: A systematic literature review”. In: *Cogent Business & Management* 8.1 (2021), p. 1938347.

[124] Elisa Conz and Giovanna Magnani. “A dynamic perspective on the resilience of firms: A systematic literature review and a framework for future research”. In: *European Management Journal* 38.3 (2020), pp. 400–412.

[125] International Organization for Standardization. *ISO 22301:2019: Security and resilience - Business continuity management systems.* (Visited on 05/28/2024).

[126] Ozgur Erol, Devanandham Henry, Brian Sauser, and Mo Mansouri. “Perspectives on measuring enterprise resilience”. In: *International Systems Conference (SYSCON)*. IEEE. 2010, pp. 587–592.

[127] Ozgur Erol, Brian J. Sauser, and Mo Mansouri. “A framework for investigation into extended enterprise resilience”. In: *Enterprise Information Systems* 4.2 (2010), pp. 111–136.

[128] Ulrich Winkler, Wasif Gilani, Alex Guitman, and Alan Marshall. “Models and methodology for automated business continuity analysis”. In: *International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE. 2012, pp. 57–64.

[129] R.J.M. Stolker, D.M. Karydas, and J.L. Rouvroye. “A comprehensive approach to assess operational resilience”. In: *Proceedings of the Resilience Engineering Symposium*. Vol. 1. 2008, pp. 247–253.

[130] Pierluigi Plebani, Andrea Marrella, Massimo Mecella, Marouan Mizmizi, and Barbara Pernici. “Multi-party business process resilience by-design: a data-centric perspective”. In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer. 2017, pp. 110–124.

[131] Adela del-Río-Ortega, Manuel Resinas, Cristina Cabanillas, and Antonio Ruiz-Cortés. "On the definition and design-time analysis of process performance indicators". In: *Information Systems* 38.4 (2013), pp. 470–490.

[132] Christopher Sims. "Macroeconomics and reality". In: *Econometrica* 48.1 (1980), pp. 1–48.

[133] Colm Kearney and Mehdi Monadjemi. "Fiscal policy and current account performance: International evidence on the twin deficits". In: *Journal of Macroeconomics* 12.2 (1990), pp. 197–219.

[134] Helmut Lütkepohl. *New Introduction to Multiple Time Series Analysis*. Springer, 2005.

[135] Tim Krake, Daniel Klötzl, David Hägele, and Daniel Weiskopf. "Uncertainty-Aware Seasonal-Trend Decomposition Based on Loess". In: *IEEE Transactions on Visualization and Computer Graphics (TVCG)* 31.2 (2024), pp. 1496–1512.

[136] Larry Wasserman. "Bayesian Inference". In: *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2004, pp. 175–192.

[137] Skipper Seabold, Josef Perktold, et al. "Statsmodels: econometric and statistical modeling with Python." In: *SciPy* 7.1 (2010), pp. 92–96.

[138] Greta Ljung and George Box. "On a measure of lack of fit in time series models". In: *Biometrika* 65.2 (1978), pp. 297–212.

[139] Peter Brockwell and Richard Davis. *Time series: theory and methods*. Springer, 2009.

[140] Paulo C. Emiliano, Mário J.F. Vivanco, and Fortunato S. de Menezes. "Information criteria: How do they behave in different models?" In: *Computational Statistics & Data Analysis* 69 (2014), pp. 141–153.

[141] Stephen Bates, Trevor Hastie, and Robert Tibshirani. "Cross-validation: what does it estimate and how well does it do it?" In: *Journal of the American Statistical Association* 119.546 (2024), pp. 1434–1445.

[142] Dimitris Korobilis. "VAR forecasting using Bayesian variable selection". In: *Journal of Applied Econometrics* 28.2 (2013), pp. 204–230.

[143] Ralf Brüggemann, Hans-Martin Krolzig, and Helmut Lütkepohl. *Comparison of Model Reduction Methods for VAR Processes*. SFB 373 Discussion Paper. Humboldt University of Berlin, 2002.

[144] Alessandro Berti, Sebastiaan van Zelst, and Wil van der Aalst. *Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science*. 2019. arXiv: `1905.06169`.

[145] Laura Oberle and Han van der Aa. “DDPS: A Project Methodology for Data-Driven Process Simulation”. In: *Proceedings of Americas’ Conference on Information Systems (AMCIS)*. Vol. 13. 2023, 1767.

[146] Manuel Camargo, Daniel Báron, Marlon Dumas, and Oscar González-Rojas. “Learning business process simulation models: a hybrid process mining and deep learning approach”. In: *Information Systems* 117, 102248 (2023).

[147] Orlenys López-Pintado, Iryna Halenok, and Marlon Dumas. “Prosimos: Discovering and Simulating Business Processes with Differentiated Resources”. In: *International Conference on Enterprise Design, Operations, and Computing (EDOC)*. Springer. 2022, pp. 346–352.

[148] Fábio Bezerra, Jacques Wainer, and Wil van der Aalst. “Anomaly detection using process mining”. In: *International Workshop on Business Process Modeling, Development and Support (BPMDS)*. Springer. 2009, pp. 149–161.

[149] Jonghyeon Ko and Marco Comuzzi. “A systematic review of anomaly detection for business process event logs”. In: *Business & Information Systems Engineering (BISE)* 65.4 (2023), pp. 441–462.

[150] Riyanarto Sarno, Fernandes Sinaga, and Kelly Rossa Sungkono. “Anomaly detection in business processes using process mining and fuzzy association rule learning”. In: *Journal of Big Data* 7.1, 5 (2020).

[151] Bart F.A. Hompes, Abderrahmane Maaradji, Marcello La Rosa, Marlon Dumas, Joos Buijs, and Wil van der Aalst. “Discovering causal factors explaining business process performance variation”. In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer. 2017, pp. 177–192.

[152] Sander Leemans and Niek Tax. “Causal reasoning over control-flow decisions in process models”. In: *International Conference on Advanced Information Systems Engineering (CAiSE)*. Springer. 2022, pp. 183–200.

[153] Enrico Zio. “The future of risk assessment”. In: *Reliability Engineering & System Safety* 177 (2018), pp. 176–190.

[154] Terje Aven. “Risk assessment and risk management: Review of recent advances on their foundation”. In: *European Journal of Operational Research* 253.1 (2016), pp. 1–13.

[155] Andrea Saltelli, Stefano Tarantola, Francesca Campolongo, Marco Ratto, et al. *Sensitivity analysis in practice: a guide to assessing scientific models*. Vol. 1. Wiley Online Library, 2004.

[156] Federico Ferretti, Andrea Saltelli, and Stefano Tarantola. “Trends in sensitivity analysis practice in the last decade”. In: *Science of the Total Environment* 568 (2016), pp. 666–670.

[157] Ilya M Sobol. “Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates”. In: *Mathematics and Computers in Simulation* 55, 1-3 (2001), pp. 271–280.

[158] Jerome H. Friedman and Nicholas I. Fisher. “Bump hunting in high-dimensional data”. In: *Statistics and Computing* 9.2 (1999), pp. 123–143.