Towards Optimization of Anomaly Detection Using Autonomous Monitors in DevOps

Hrusto, Adha

2022

Document Version:
Publisher's PDF, also known as Version of record

Towards Optimization of Anomaly Detection Using Autonomous Monitors in DevOps

Adha Hrusto

Licentiate Thesis, 2022
Department of Computer Science
Lund University

ISBN 978-91-8039-213-6 (printed version)
ISBN 978-91-8039-214-3 (electronic version)
Licentiate Thesis 3, 2022
ISSN: **1652-4691**

Department of Computer Science
Lund University
Box 118
SE-221 00 Lund
Sweden

Email: adha.hrusto@cs.lth.se
WWW: `http://cs.lth.se//adha-hrusto/`

Printed in Sweden by Tryckeriet i E-huset, Lund, 2022

ABSTRACT

Continuous practices including continuous integration, continuous testing, and continuous deployment are foundations of many software development initiatives. Another very popular industrial concept, *DevOps*, promotes automation, collaboration, and monitoring, to even more empower development processes. The scope of this thesis is on continuous monitoring and the data collected through continuous measurement in operations as it may carry very valuable details on the health of the software system.

Aim: We aim to explore and improve existing solutions for managing monitoring data in operations, instantiated in the specific industry context. Specifically, we collaborated with a Swedish company responsible for ticket management and sales in public transportation to identify challenges in the information flow from operations to development and explore approaches for improved data management inspired by state-of-the-art machine learning (ML) solutions.

Research approach: Our research activities span from practice to theory and from problem to solution domain, including problem conceptualization, solution design, instantiation, and empirical validation. This complies with the main principles of the design science paradigm mainly used to frame problem-driven studies aiming to improve specific areas of practice.

Results: We present identified problem instances in the case company considering the general goal of better incorporating feedback from operations to development and corresponding solution design for reducing information overflow, e.g. alert flooding, by introducing a new element, a smart filter, in the feedback loop. Therefore, we propose a simpler version of the solution design based on ML decision rules as well as a more advanced deep learning (DL) alternative. We have implemented and partially evaluated the former solution design while we present the plan for implementation and optimization of the DL version of the smart filter, as a kind of autonomous monitor.

Conclusion: We propose using a smart filter to tighten and improve feedback from operations to development. The smart filter utilizes operations data to discover anomalies and timely report alerts on strange and unusual system's behavior. Full-scale implementation and empirical evaluation of the smart filter based on the DL solution will be carried out in future work.

ACKNOWLEDGEMENTS

This thesis was partially supported by the Wallenberg Artificial Intelligence, Autonomous Systems and Software Program (WASP) funded by Knut and Alice Wallenberg Foundation.

I would like to express my gratitude and appreciation for Prof. Per Runeson, Dr. Emelie Engström, and Dr. Magnus C Ohlsson whose guidance, support, and encouragement have been invaluable throughout this Ph.D. journey. It has been a privilege to have you as supervisors and to collaborate with you on the studies.

I would also like to thank my colleagues from the Department of Computer Science, especially the SERG members for inspiring discussions and very pleasant lunches and fikas. Alma, Song, Sergio, Momina, and Idriss thank you for making this journey even more wonderful.

I would like to offer my special thanks to System Verification, the company where I have always been warmly welcomed, appreciated, and encouraged to pursue my personal and professional goals. In particular, I would like to thank Kadira Šubo and Henrik Sällman for initiating and encouraging application for the doctoral program, as well as to my immediate manager Andreas Axelsson and HR manager Joanna Doweyko for continuous support and patience in managing all relocation formalities.

I have had an opportunity to collaborate with the development and test teams from an anonymous case company and System Verification. I am thankful for their willingness to share insights and respond to all questions. I would like to specially thank Tobias Anderson for the helpful advice and unlimited assistance at any time.

I am extremely grateful to my family and friends for their unconditional love, unwavering support, and belief in me.

Finally, all praises and thanks are due to God, the Almighty, for His blessings and knowledge granted to me.

Tack så mycket!
Adha Hrusto

LIST OF PUBLICATIONS

This thesis consists of an introduction and a compilation of two papers listed below and referred to by Roman numerals.

Publications included in the thesis

I **Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations**
Adha Hrusto, Per Runeson, Emelie Engström
Springer Nature Computer Science (2021) 2:447
DOI:10.1007/s42979-021-00826-y

II **Optimization of Anomaly Detection in a Microservice System Through Continuous Feedback from Development**
Adha Hrusto, Emelie Engström, Per Runeson
In Proceedings of 10th ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems (SESoS 2022).

Contribution statement

All papers included in this thesis have been co-authored with other researchers. The authors' individual contributions to Papers I-II are as follows:

Paper I
Adha Hrusto, Prof. Per Runeson, and Dr. Emelie Engström initiated study and collaboration with the industry partner. All authors contributed in conducting interviews with the practitioners and formulating identified problem instances. Adha Hrusto has led the solution design and implementation of the solution prototype while the decisions and actions taken were critically assessed by Prof. Per Runeson and dr. Emelie Engström. Adha Hrusto was mainly responsible for writing the paper while Prof. Per Runeson and dr. Emelie Engström have also contributed to writing, reviewing, and editing.

Paper II
Adha Hrusto designed the study and as a lead author, was responsible for writing the paper. Dr. Emelie Engström initiated very important discussions about the research approach, while Prof. Per Runeson assisted in refining and formulating design science concepts of the study. Adha Hrusto proposed reviewing deep learning approaches for advancing solution design and a plan for its in-context implementation and optimization. Dr. Emelie Engström and Prof. Per Runeson have also contributed to reviewing and editing the paper.

CONTENTS

INTRODUCTION

Continuous software engineering (CSE) embraces continuous practices such as continuous integration, continuous testing, continuous deployment, continuous run-time monitoring, to support and enhance the entire software life-cycle all the way from the business strategy to the final software product. Another term, closely related to continuous practices and more often used in industry, *DevOps*, stands for cross-functional collaboration between development (Dev) and operations (Ops) and complements continuous practices by stressing the importance of *collaboration* and *culture* in software development environments. *Automation* and *monitoring* are additional dimensions of DevOps [44], also highly important within CSE. Thus, automation infrastructure is a means of achieving *continuity* of CSE activities while run-time monitoring in operations monitors the health and performance of the deployed software through continuous *measurements*. Accumulated monitoring data may provide particular insights to development e.g. timely exposing unusual discrepancies in performance metrics. Accordingly, analyzing and managing operational data is of high importance for providing fast and valuable feedback to development, which is the focus of our ongoing research.

One of the benefits of implementing continuous run-time monitoring is improved release reliability due to shorter testing cycles and more frequent releases. Hence, extensive testing prior to deployment might be less favored due to shorter time-to-market demands. However, some development teams have started relying on production monitoring for discovering bugs, glitches, or performance degradation as long as they can be easily and quickly fixed. This is enabled thanks to automated continuous integration/continuous deployment (CI/CD) infrastructure. On the other hand, monitoring highly complex software systems, e.g. consisting of dozens of microservices might be challenging as the volume and complexity of monitoring data increases over time with the software system evolution. Therefore, collected monitoring data in such systems may burden operations and cause information overflow e.g. alert flooding, in the feedback loop to development.

A microservice system is an example of a distributed software system where every component is separately monitored and potential issues are individually reported for each of the services even though they might be dependent. Therefore, in order to capture the health status of a system as a whole, it is important to consider

operational data across multiple microservices. For such systems, selecting proper algorithms for the analysis and detection of specific events in multidimensional operational data is highly significant. Even more challenging are the industrial contexts with unknown expectations and alerting events, where each reported alert needs to be manually checked in order to be resolved.

In our research, we specifically focus on the detection of anomalous system's behavior in monitoring data, e.g. performance metrics, and timely reporting alerts to development, providing all needed details for fast debugging and root cause analysis. Anomaly detection in operations data is an important activity for discovering faulty system states and performance degradation that could evolve into severe problems if not properly addressed and resolved on time. For addressing anomaly detection tasks, we focus on machine learning (ML) inspired approaches as they have been recognized as very suitable and reliable for discovering deviations in high-volume data [1, 16, 28, 57, 75].

We collaborated with a Swedish company responsible for ticket management and sales in public transportation, to be able to further explore the usage of runtime monitoring data for the detection of anomalies. Since we only collaborated with one industry partner, in further text we refer to it as the *case company*. The research work was divided into the following three areas:

- *Exploration and identification* of main challenges in the case company with respect to feedback from operations to development, focusing on the available operations data and its utilization for discovering alerting events.
- *Reading* relevant state-of-the-art literature, *analyzing* the findings and suitable ML approaches and accordingly *designing* a solution for identified challenging problem instances.
- *Implementation and validation* of the proposed solution design in the case company using tools applicable for the studied industrial context.

The system under study, developed and operated by the case company, is an example of a microservice system with distributed monitoring in operations. As a consequence, all detected performance degradation and glitches, namely *anomalies*, across individual services are reported as *alerts* to the single notification platform (in this case a Slack channel). Those are the facts we started with after collaboration with the case company was initiated. Further, the focus was on exploring ML approaches for anomaly detection and utilizing a monitoring platform and corresponding monitoring data for developing an autonomous monitor for detecting anomalies and reporting alerts.

Conducting the work in the aforementioned research areas has led to the following threefold contributions:

- **C1:** Identification of problem instances on the borderline between operations and development, in particular, instances of problems related to the management of the operations data and data complexity.

- **C2:** A solution design inspired by ML, preventing a bottleneck in the information flow from operations to development, and timely reporting of alerts on identified anomalous system's behavior.
- **C3:** Partial empirical evaluation of the simpler version of the solution design based on ML decision rules and a plan for implementation and evaluation of the deep learning (DL) alternative.

1 Background and Related Work

In this section, the main aspects of continuous software engineering are discussed. The presented aspects are related to current research done in this thesis, focusing on the continuous activities in development and operations. Further, an overview of continuous monitoring and monitoring data management is presented. Moreover, the overview extends to deep learning approaches for anomaly detection in monitoring operations data in the previously introduced studied context.

1.1 Continuous Software Engineering

Continuous software engineering provides a set of continuous activities needed for an iterative software development process with accelerated delivery. As shown in Figure 1 by Fitzgerald and Stol [18], CSE spans across three domains: business strategy, development and operations.

The software product creation starts with business strategy and planning, followed by technical implementation, deployment, and continuous execution in operations. This is accomplished in several cycles, were BizDev and DevOps, connecting the three domains, ensure continuous flow of information between *business strategy* and *development*, and *development* and *operations*, respectively. In this way, any discontinuities between sales/marketing expectations, development decisions, and users' satisfaction are eliminated [18].

DevOps gains more attention than BizDev as it is more frequently used within both, academic and industrial circles. A common interpretation of DevOps is that it is about culture, tooling, and processes and aims to improve collaboration and integration between team members including developers, managers, operational personnel, and everyone involved in product creation and deployment. This means that DevOps is defined by four principles: collaboration, automation, measurement, and monitoring [45].

As shown in Figure 1, the three domains are represented with continuous practices required for planning, developing, and operating software. In the conducted research work, the focus was on the linkage between development and operations, thus, the corresponding continuous practices are further explained:

- **Continuous Integration (CI)** is an automatically triggered process of interconnected steps including code compilation, execution of automated tests

on unit, integration, system and acceptance level, validating code coverage, and building deployment packages.

- **Continuous Testing (CT)** is an uninterrupted process of running automated tests on the latest stable version of the software. Its main purpose is to help reducing the time between the introduction of errors and their detection.
- **Continuous Deployment (CD)** is an automatic process of reliable delivering software to the customer environments as soon as new code is developed [54]. According to Rodríguez [54], the CD has roots in Agile Software Development (ASD) as agile methodologies are based on the same principles of delivering valuable software continuously while accelerating development processes.
- **Continuous Delivery** is a prerequisite for continuous deployment and releasing of software builds for this practice is done manually and not directly to customers, while with continuous deployments it is done automatically and directly to customer environments [18].
- **Continuous Run-Time Monitoring (CM)** is a continuous activity of monitoring running software services, which may enable tracking their health sta-

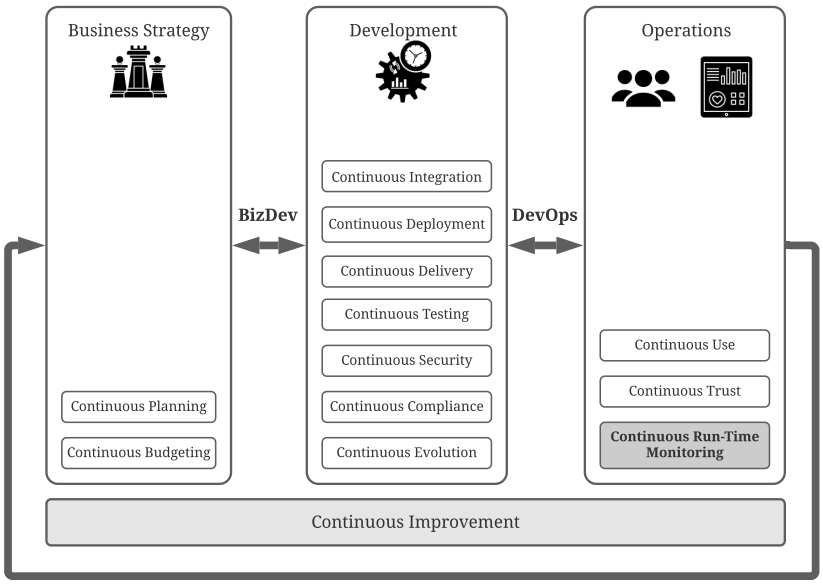


Figure 1: A holistic view on the continuous activities [18]

tus and timely detection of various malfunctions [18] while collected monitoring data may serve for improving development processes [11], generating or prioritizing test cases [52, 64], or detecting anomalies [28].

- **Continuous Improvement** is a data-driven activity for improving software quality and increasing customer satisfaction by utilizing data generated in the CI/CD pipeline to drive decision-making and eliminate waste.

Another term, mostly used among practitioners, that encompasses the aforementioned continuous activities is *continuous integration/continuous deployment (CI/CD) pipeline*. It is considered as a set of automated steps that need to be performed to be able to deliver software, while it is complemented with DevOps principles of collaboration and culture to ensure uninterrupted responsiveness within the pipeline [60].

Implementation of the continuous practices may bring various benefits to software development environments such as [54]: 1) *shorter time-to-market* due to the higher frequency of release cycles; 2) *improved release reliability* achieved with a narrower test focus since each deployment introduces only a limited amount of code while also relying on rollback mechanisms; 3) *instant feedback* from automated infrastructure that enables discovering, locating, and resolving issues more rapidly, utilizing both pre-deployment extensive testing and post-deployment runtime monitoring; 4) *increased developers' productivity* as they can only be oriented on the development of new features while relying on the automation infrastructure and feedback from operations for discovering bugs and various anomalies.

Further, we narrow our focus to continuous run-time monitoring since it has an important role in the feedback from operations to development and providing valuable alerting notifications to development in case of detected anomalies, which was recognized as crucial in addressing identified problem instances in the case company.

1.2 Continuous Monitoring

The main idea of the term *continuity* is to ensure an uninterrupted software development life-cycle, even after deployment. When a software product has been deployed, it is important to continue with post-deployment activities, such as continuous monitoring. Suonsyrja et al. [61] studied the importance of post-deployment activities and how automatically collected data from operations could be used as feedback for requirement modifications and further development processes. They reviewed the literature that relates to the evolution of software development methods and created a questionnaire to examine if companies are interested in collecting usage data and what they think the data could be used for. The results show that practitioners are interested in adopting post-deployment activities and that these activities are important for gathering data about performance, time, and the way of using the software product but also for discovering system failures.

Hence, continuous monitoring complements traditional testing, especially because the time intended for testing tends to be decreased due to short time-to-market demands or limited development resources. In this way, continuous monitoring may serve in discovering various anomalous system's behavior in case software products are being released with limited testing coverage or with incomplete functionalities. Furthermore, underlying automation infrastructure offers a lot of opportunities for the further advancements of post-deployment activities.

Types of data collected through continuous monitoring in operations may take different forms and provide diverse information about the software in operations. For instance, Cito et al. [10] report a taxonomy of types of operations data, where they differentiate between *monitoring* data and *production* data. Monitoring data includes execution performance data (e.g. service response time), load data (e.g. service requests rate), cost data (e.g. hourly costs for using specific cloud solution), and user behavior data (e.g. click-streams), while production data denotes the data produced by the software system itself such as placed orders of customer information. In a more recent study, Cito et al. [11] categorize operations data in system metrics (e.g. CPU utilization), application metrics (e.g. number of exceptions, logs/events), and application system metrics (e.g. method-level response time) where they also map identified categories with the phases of software development life-cycle and their purpose. Further, Capizzi et al. [6] investigate data management within DevOps all the way from the planning stage to the deployment stage. They have also recognized monitoring data such as logs, performance metrics, and other types of data collected from monitoring tools, as an important segment of DevOps data. However, there are also studies that specifically focus only on a certain type of operations data such as logs – printing statements from developers to trace the state of the running software [4]. Hence, the terminology used to describe specific types of operations data may differ among research circles and years of publication. However, some patterns may be noticed – performance metrics data has been recognized as an important source of information for providing valuable feedback for development.

One of the initial aims when collaborating with the DevOps team from the case company was to explore available operations data types. Identified categories partially match with aforementioned types, specifically including applications' and systems' performance metrics (e.g. CPU Time, the total number of requests, number of failed dependency calls, or number of internal server errors), logs (e.g. failed dependencies with attributes such as id, target, name, type or result code) and business data (e.g. number of ticket purchases using specific payment method). Further investigations on the management of operations data and observations on resolving alerting events indicated that we put our focus towards performance metrics as they have been usually analyzed in case some severe failures and their numerical form make them adequate to use with ML algorithms.

1.3 Monitoring Data Management

Monitoring data in operations may be leveraged for improving different aspects of the software development life-cycle. Using feedback based on the operations data aggregation, integration, and mapping for determining analytical and predictive relation to development processes has been reported by Cito et al. [10, 11]. The testing phase may also benefit from monitoring data, e.g. to improve the reliability of testing with the iterative estimation of operational profile presented by Pietrantuono et al. [52] or for automatic generation of test cases reported by Deepika et al. [64]. Moreover, security and privacy issues could be addressed utilizing monitoring data [49]. Detecting anomalous system's behavior is another very important aspect for timely reporting alerting events and fast recovery from failures [28]. Interestingly, some of the operations data has been used for decision making and detection of harmful release candidates [6], while Fu et al. presented how to use clusters of system logs to further infer dependencies that are used for failure prediction and root cause analysis [20]. Those are selected examples of operations data utilization for various purposes while there are many more use cases adapted for specific industrial contexts.

Initial insights gained from collaboration with the case company has led to prioritizing anomaly detection among other ways of utilizing monitoring data. During the problem conceptualization, various flaws has been detected in the way alerting events were being reported. Thus, a need for a new and improved way of discovering unusual and unexpected software behavior, namely anomalies, has been recognized.

1.4 Anomaly Detection in Monitoring Data

Anomaly detection is a research area with growing attention across numerous domains such as cybersecurity, healthcare, bioinformatics, industrial fault detection, genetics, and many more [55]. An observation that significantly and unexpectedly deviates from the normal pattern in the observation data is considered as an anomaly [3]. However, the concept of anomaly detection may be context-dependent and different approaches might be needed for different types of data. According to Hagemann et al. [23], the three most common approaches to anomaly detection entails classical machine learning, deep learning, and statistical methods. The selection of the approaches substantially depends on the available data, its type, and volume. Moreover, the data available for the learning of normal and anomalous dependencies may define selection between unsupervised and supervised methods. Labeled data for applications of anomaly detection are rarely available and even when there are at least partial annotations, they might be insufficient and unreliable since anomalies in most of the industrial context do not appear often and periodically [55]. Thus, they are hardly detected and require special treatment. Moreover, a careful selection of suitable approaches for their analysis is needed.

In our case, the selection of anomaly detection approaches has been defined by the software system and environment in the case company. Thus, we focused on approaches for treating *multidimensional time series data* as the monitoring data, indicating the health of the software, consists of multiple numerical observations captured within a specific time range across multiple services. This type of data, known as multivariate time series, is quite challenging for analysis especially if there are no ground truth data, which is the case. Initially, we focused on classical machine learning approaches for labeling the data and predicting anomalies but due to data complexity and untrustworthiness of the labeling process, we shifted focus towards advanced approaches for anomaly detection such as *deep learning*.

Deep Learning for Anomaly Detection

There is a tendency to use deep learning (DL) approaches for solving diverse tasks in a plethora of applications. The main idea behind those, human brain-inspired methods, lies in the pile of interconnected neural network layers capable of learning complex and non-linear dependencies in the data [55]. Deep learning approaches for anomaly detection differ in the type of neural networks layers, the way inter-correlation between observations is defined, and the way of calculating the anomaly score [8]. Hence, a wide variety of approaches can be found in the state-of-the-art literature [8, 28, 55] where most common approaches include reconstruction-based, prediction-based, and dissimilarity-based deep learning methods for anomaly detection. Choosing between the aforementioned methods and their variations is mostly defined by the industrial context since, to our knowledge, there is no all-purpose method that fits into diverse contexts.

A challenging fact in the studied problem context was the highly unstructured and unpredictable monitoring data with unknown anomalous labels. Hence, selecting and evaluating a single deep learning method that will perform optimally is quite untrustworthy. Instead, we relied on feedback from the development team to generate labels that will afterward be used for evaluating and selecting the method with the best accuracy. Thereafter, the continuous feedback from practitioners on reported alerts based on detected anomalies will further serve for updating the best candidate in several iterations. As more data becomes available over time, each update will drive the DL model towards the *optimum* and the entire solution towards an *autonomous* solution for managing monitoring data.

1.5 Alerts in DevOps

Alerts are warning or error notifications reported based on detected software malfunctions or performance degradation. They are a convenient way of notifying development teams about the state and the health of the software running in operations. The alert rules can be implemented either using a heuristic approach – manually thresholding monitored data or more advanced algorithms treating the

raw monitoring data such as anomaly detection. However, there might be some variations depending on the industry domain, such as a large-scale service system of a commercial bank [72], cyber-physical systems [41], process plants e.g. an offshore oil gas separation plant [43] or large-scale enterprise IT system [39], on how alerting events are triggered.

Alerts are useful for timely diagnosing severe failures but on the other hand they might cause information overflow and confusion within the development teams if they are not properly managed. For instance, Zhao et al. [72] report a case where the development team had wasted a time debugging non-severe alerts while severe ones were missed. Thus, they propose an effective ML-based algorithm, which utilizes both reported alerts and key performance indicators (metrics) to more accurately trigger high-priority alerts. Similarly, to improve operational efficiency and support manual alert examination, Lin et al. [39] propose an unsupervised ML approach for clustering alerts based on their content.

The studied industrial context also comprises alerts in the feedback loop from operations to development. The aforementioned approaches for improving alert management were not applicable due to the low quality of implemented alerting rules. Thus, a different approach, including raw monitoring data analysis, was needed. Instead of alert prioritization, we focused on mining monitoring data in order to discover highly relevant anomalies also considering dependencies among services and timely reporting alerts.

2 Research Approach

An overview of the research work in this thesis is presented in Figure 2. The goals of the design science studies were to explore problems in the specific industrial context, a microservice system developed and operated in the DevOps environment (Paper I and Paper II), investigate how to improve the problem context by applying state-of-the-art machine learning methods (Paper I and Paper II), and implement and evaluate the ML inspired solution instance (Paper I, Paper II and Evaluation paper). Afterwards, based on the synthesis, the steps for further work towards generalization are decided.

Throughout the collaboration with the case company and while studying challenges in managing operations data, we aimed at answering the following research questions:

- **RQ1:** What are the main challenges on the borderline between development and operations with regards to operations data flow and overflow?
- **RQ2:** What available ML approaches may be used for designing a solution for the identified problems of data overflow from operations to development?

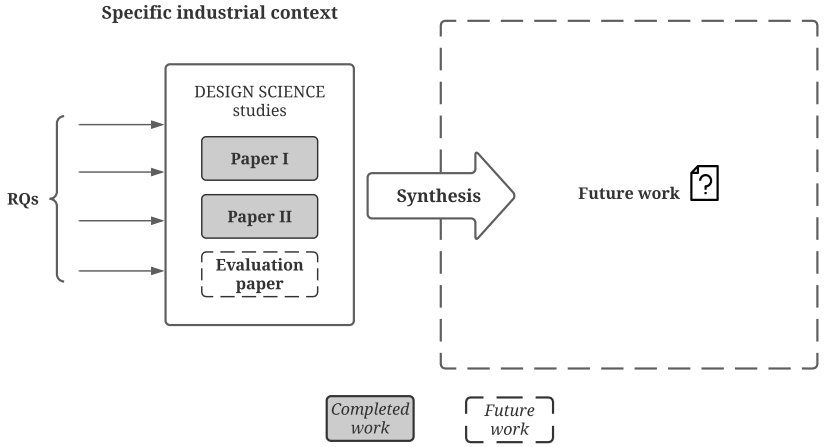


Figure 2: Research overview of the completed work

- **RQ3:** What tools and cloud platforms are needed for the deployment and evaluation of the implemented ML based solution?

- **RQ4:** How to evaluate ML based solutions without ground truth data and how to examine if selected ML approaches successfully address identified problem instances of data overflow?

The details of the future work are elaborated in Section 5.

The conducted research work, presented in the papers included in this thesis, follows the principles of the design science (DS) paradigm [56]. We aimed to explore and address challenges in managing operations data by studying the case company, which perfectly fits the design science concept of understanding and improving certain practices, in our case in an industrial DevOps context. A conceptual overview of the research approaches, for the two papers under the design science frame, is shown in Figure 3. Each of the research approaches in Paper I and Paper II include activities shown in arrows (see Figure 3) that span from problem to solution and from practice to theory domain.

The design science frame entails four main activities: problem conceptualization, solution design, instantiation, and empirical validation [56]. *Problem conceptualization* is a fundamental constituent of design science research but it is not necessarily the first step [56]. However, it may be a starting point of problem-driven research as the problem first needs to be understood in order to envision potential solutions. *Solution design* is an activity of mapping identified problem to a matching solution [56]. Furthermore, identified problem instances and matching

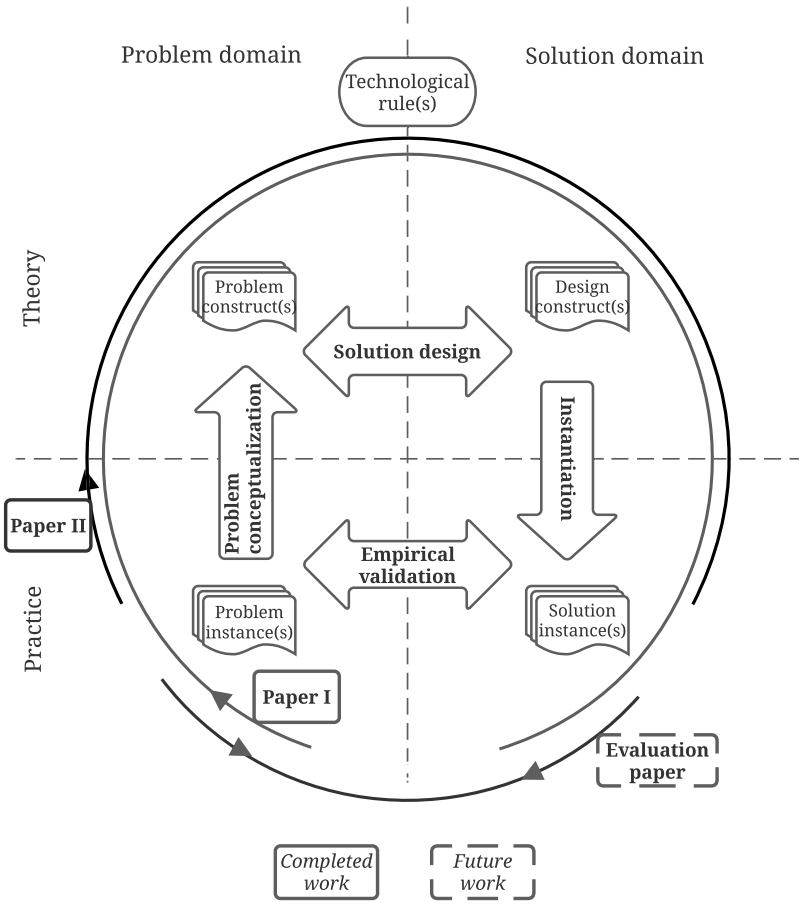


Figure 3: Research approach under the design science frame

solutions may be expressed as *technological rules*, e.g. prescriptive recommendations for practice. *Instantiation* refers to the implementation of the solution design in a specific industrial context while *empirical validation* aims at evaluating the solution instance and examining how well it addresses identified problem instances [56].

The four research questions are in line with the design science paradigm and we aim at answering them by conducting the aforementioned DS activities. Thus, in the first study **(Paper I)** we used interviews and observations to explore the problem domain in the case company to identify problem instances and formulate

a general problem construct considering related work in the field, which adds to the contribution **C1**. Further, the matching solution was designed considering previous research and available solutions to similar problems and served as a proof of concept for further work. The solution design presented in Paper I, was implemented using exiting ML solutions and Python libraries for learning decision rules. The complexity of the industrial context hindered us from a full-scale implementation. Thus, the solution instance served as a proof of concept to explore possibilities for further work and more advanced algorithms. However, it aims at answering the **RQ2** and leads to the contribution **C2**. As presented in Paper I, we managed to execute a partial empirical validation of the initial solution instance considering time and environment constraints. Thus, the circle, shown in Figure 3, is not fully complete and only partially adds to **C3**.

Research questions **RQ2, RQ3**, and **RQ4** are further explored in the second study, published in Paper II. As shown in Figure 3, the problem conceptualization step in the second study (**Paper II**) was less extensive, including brief discussions with practitioners and investigation of relevant solutions to formulate a more condensed problem construct. Moreover, in Paper II, we advance the proposed solution design, presented in Paper I, with respect to the state-of-the-art deep learning solutions in order to capture all notions of anomalousness in operations data. The goal is to provide a more robust and reliable solution than the initial one, which complements contributions from Paper I and finalizes **C2**. Furthermore, the guidelines for implementation and full-scale evaluation in a cloud environment are presented, which adds to contribution **C3**. We aim to complete the design science cycle with an evaluation paper (see Figure 3), where an actual implementation and empirical validation of the advanced solution instance will be presented.

3 Results and Contributions

The overall contribution of this thesis is related to the goal of improving feedback from operations to development in cloud-based and large-scale software systems, developed and operated in DevOps environments. In particular, it entails optimization of anomaly detection in multidimensional operations data needed for timely reporting alerts and reducing information overflow between the two domains. A more detailed overview of specific results and contributions per paper is given below.

3.1 Paper I: Closing the Feedback Loop from Operations to Development

The goal of a study, presented in Paper I, was to explore the problem context in the case company and design a solution for a conceptualized problem, which is later instantiated and will be evaluated as a proof of concept in further work.

Next, individual contributions of Paper I and their relation to the three overall contributions of this thesis are presented below.

Throughout the problem conceptualization, we identified the following three problem instances of the general alert flooding problem to answer **RQ1**, which correspond to the contribution **C1**:

- **Targeting problem:** The distribution of alerts to target recipients, teams, and individuals is not fully explicit, which causes lack of precision in resolving fired alerts.
- **Optimization problem (High priority alerts vs. noisy alerts):** The differentiation between alerts that cause failures and alerts causing temporary glitches that don't affect the system's performance, is not clear.
- **Interoperability flaws with external services:** Disruptions in external services are often discovered through customer service after end customers have been affected.

We provided a conceptual design for only one of the problem instances, alert flooding as an optimization problem since addressing this particular instance may reduce the scope of the remaining problems. As shown in Figure 4, the proposed solution design entails an additional element in the feedback loop from operations to development, a *smart filter*. The smart filter is a kind of autonomous monitor that is capable of detecting performance anomalies in monitoring data and triggering alerts in a development environment. In particular, it fetches various systems' and applications' metrics in near-real-time and outputs a triggering event if an unusual system behavior is detected based on the alert rules, created as functions of multiple input variables. In this way, we urge treating the metrics data across different services as some hidden dependencies might be discovered, which is crucial for a fast response from development on fired alerts.

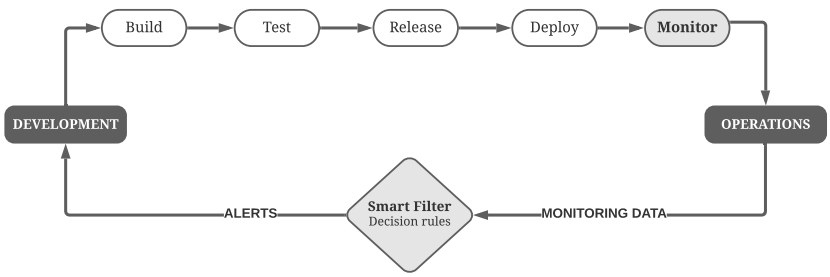


Figure 4: Smart filter (decision rules) in the feedback loop

Alongside the proposed solution design, which is contribution **C2**, we implemented and evaluated a prototype instance, which leads to contribution **C3**. The

goal of the implementation part was to explore limitations of the operations data and the corresponding context. Thus, we focused on basic ML methods, e.g. *tree-based machine learning methods* [19,21], to generate new advanced decision rules. Moreover, such rule-based ML methods require labeled data so we used the baseline ML anomaly detection method, e.g. *isolation forests* [38] to generate labels while also considering the service vulnerability and observed metrics frequency.

We evaluated the solution instance against the current solution in the case company for alert detection, as well as against the unsupervised *multivariate anomaly detection* (MAD) method for detecting anomalies that will thereafter trigger alerting events. For that purpose, a limited test data set was used to explore the effectiveness of the proposed solution in the studied context. In each of the evaluation instances, the results showed that the MAD trained model produced the same level of noisy alerts as the existing alert system in the case company, while the smart filter outperformed them by significantly reducing the alert flooding, simplifying the root cause analysis, and further resolving altering events. This was a pilot implementation and evaluation of the proposed solution in the case environment as a proof of concept for further work.

Throughout this study, we identified several challenges that impacted our further decision-making about the future work in the case company. We were struck by how much uncertainty each of the fired alerts was continuously bringing into the development environment, as they were sent through two different platforms (Slack channel and email) and there were no clear attempts at resolving them. Moreover, it was hard to understand what triggers developers to act upon fired alerts, which could further help in defining a *real alert* and evaluating any alternative approach for alert detection. The previously presented solution design included a labeling process, which is, in general, a very thorny process and highly sensitive in case of anomaly detection problems. Hence, we decided to shift our focus towards *unsupervised deep learning approaches* for anomaly detection in multivariate data, governed by the existing solutions in the state-of-the-art literature. Furthermore, we decided to explore approaches for evaluation of the ML-based solutions without any ground truth data. The final aim was to create a unique technical solution that will overcome all aforementioned challenges and enable timely reporting of the most relevant alerts to only one of the communication platforms.

3.2 Paper II: Optimization of Anomaly Detection

In Paper II, we address shortcomings of the solution design presented in Paper I and explore state-of-the-art deep learning (DL) approaches for its improvement. We were inspired by the results achieved using the unsupervised DL methods in addressing anomaly detection tasks [28, 42, 46]. Unsupervised learning does not require data labeling, which is preferable due to the ambiguity of fired alerts in the case company, while deep learning enables capturing highly complex and nonlinear dynamics in the multivariate data. Below, we present the individual contribu-

tions of Paper II and the corresponding mapping to the three overall contributions of this thesis.

An additional contribution to **C2** was made by advancing the solution design from Paper I, with deep learning approaches. For that purpose, we firstly presented a brief overview of unsupervised deep learning methods for anomaly detection in multivariate time series, which addresses **RQ2**. The methods were presented across three dimensions including inter-correlation between variables, type of neural networks and the way anomaly score was calculated. Moreover, we provided guidelines on how to select minimum feasible DL methods, in terms of simplicity and applicability, when firstly attacking the problem of anomaly detection in time series. We proposed three variations of DL methods with brief descriptions on how to use them and corresponding examples. These guidelines were also planned to be used for advancing the smart filter and further exploration of the fitting between the selected methods and the studied context.

For answering **RQ3** and **RQ4**, we presented a cloud solution for the deployment of the DL model and in-context implementation of the smart filter. Moreover, the context in the case company, characterized by a shortage of ground truth data on *real alerts*, required a special plan for evaluation of the overall solution. Thus, we introduced a concept of iterative evaluation through feedback from the development team on each reported alert. As shown in Figure 5, there is a connection between the development and the smart filter utilized for evaluating and updating the solution instance. In this way, collected feedback data, representing ground truth annotations, may be used for exploring different variations of the solution and afterward for *optimizing* the selected, best performing, DL solution. This particular contribution adds to the **C3** contribution of this thesis.

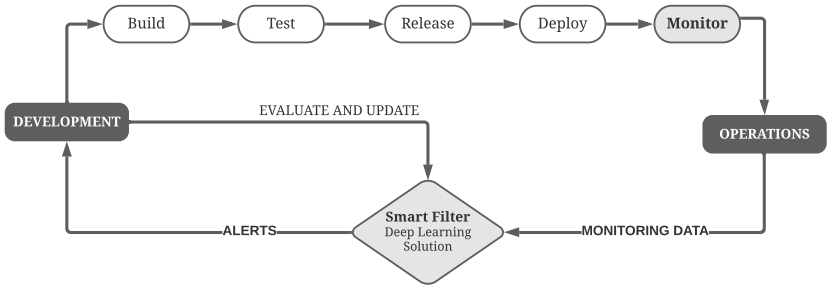


Figure 5: Smart filter (DL version) in the feedback loop

As previously mentioned, the full-scale implementation and evaluation of the advanced version of the smart filter are planned as future work and will be published as an *evaluation paper*. The details of the further future work are discussed in Section 5.

4 Limitations

As proposed by Runeson et al. [56], we assess the contributions of conducted design science research, published in Paper I and Paper II, by considering *relevance*, *rigor* and *novelty*. Below, the limitations of the thesis contributions across the aforementioned factors are discussed.

4.1 Relevance

The relevance of the design science research can be discussed ether from other practitioners or research community perspective [56]. To be useful for other practitioners than in *the case company*, the proposed solution including a kind of an autonomous monitor in the feedback loop to development, needs to be easily adapted to any other relevant industry context. Currently, we are not able to assess whether the presented research outcomes are applicable in different contexts as we only collaborated with one industry partner. However, based on knowledge gained from the related literature and previous industry experience, we may assume that the studied problem is highly relevant and that the designed solution may be used for addressing similar problems in different contexts. We plan to address this limitation in our future work where additional effort is needed to prove the previous statement.

The research community may benefit from the design knowledge with regards to the relevance of the studied problem and the degree of its generalizability. In each of the papers included in this thesis, there is a section that presents related work, which supports the relevance of the studied research phenomena. The designed solution presented in Paper I and Paper II is currently highly context-dependent and its generalizability will be further explored in future work.

4.2 Rigor

The rigor of the design science studies presented in this thesis is determined by assessing the knowledge creating activities: problem conceptualization, solution design and validation [56]. Regarding the problem conceptualization, we used interviews, observations, documentation, and informal discussion with practitioners to collect qualitative data and identify the most challenging problem instances of data flow and overflow in operations. These techniques provide direct insights into the studied phenomenon as they require quite close collaboration with the practitioners.

We assess the rigor of the design activity by estimating to which extent the proposed design builds on the prior designs from state-of-the-art solutions [56]. In Paper II, we provide an overview of the most relevant deep learning approaches which we considered for improving the initial version of the smart filter. Thus, our solution is based on the approaches used for addressing similar problems. In

Paper II we also elaborate why we focus on DL-inspired solutions instead of less complex alternative solutions.

We have the least evidence for the rigor of the validation activity. Since the full-scale implementation of the DL-inspired solution is underway, we are not able to conduct thorough validation. Moreover, we plan to execute an interactive and iterative validation where practitioners will be involved in validating reported alerting events, which will be used for measuring the accuracy of the prediction and improving the solution by retraining the unsupervised DL models or training new supervised DL models. Thus, in several iterations, we aim to find the best performing solution that includes either an individual DL model or an ensemble of more DL models (unsupervised and supervised).

4.3 Novelty

The novelty of conducted research work is expressed through a new technological rule:
To timely expose unusual software system's behavior in cloud-based environments without known ground truth data on expected anomalies, use a technical solution based on a deep learning approach with periodical updates with regards to the feedback from development.

The novelty of the contribution stated in the technological rule is in *the context* to which the proposed intervention is applied. Similar solutions have already been proposed for addressing anomaly detection problems in monitoring data. However, the difference is that the latest and most relevant publications do not explicitly address the real struggle of development teams in assessing whether the reported alerts are true or false positives [28, 48, 75]. This uncertainty caused the ignorance of reported alerts and shortage of annotated *real alerts* in the case company. With our research outcomes, we aim to address this gap in a real-world context and propose an intervention that includes a smart filter in the feedback loop from operations to development and iterative updates based on the feedback from development. In this way, the deep learning based solution may be iteratively optimized.

5 Conclusion and Future Work

Throughout the design science studies, we focused on the specific industrial context to explore the usage of operations data for addressing challenges on the borderline between operations and development. All the three studies, including published papers (Paper I and Paper II) and ongoing work planned for an evaluation paper, are related and focused on the same problem of data flow and overflow from operations to development. The studies tackle the problem from different perspectives, focusing on the specific parts of the design science frames to answer research questions, presented in Section 2.

RQ1 and partially **RQ2** were addressed in Paper I where identified problem instances and initial solution prototype were presented. **RQ2**, **RQ3** and **RQ4** were further investigated in the second study and the results were reported in Paper II. The contributions of Paper II include a review of state-of-the-art deep learning methods for anomaly detection in monitoring data (performance metrics represented by multivariate time series), an overview of a cloud platform for deployment and maintenance of deep learning solution, and an approach for evaluation and optimization of deep learning solutions without ground truth labels on expected anomalies. To complete the design science cycle, the full-scale implementation and evaluation are planned as future work and will be published in an evaluation paper. As shown in Figure 6, after the three design science studies, we synthesize the results and limitations to further decide on future work.

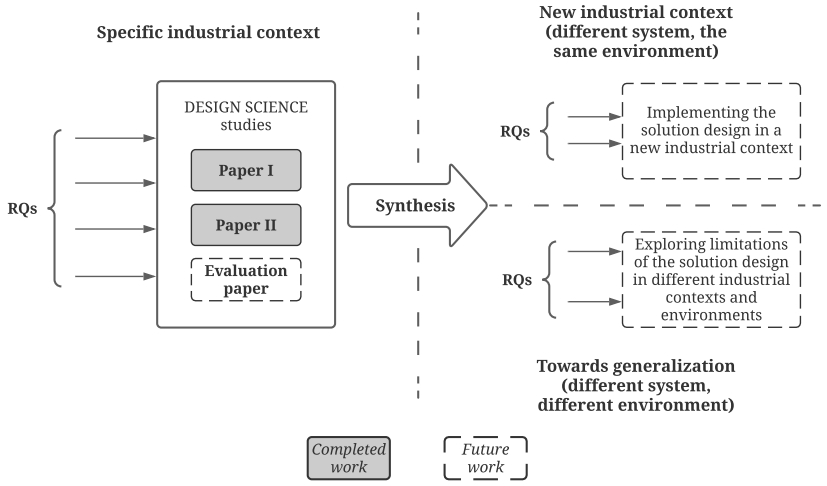


Figure 6: Research overview of the completed and future work

The accomplished results, presented in Section 3, were only significant for a specific industrial context. Moreover, considering the limitations discussed in Section 4, we decide to focus future work on increasing the *relevance* of the conducted research work. Accordingly, we divide future work into two separate studies that will explore the applicability of the solution proposed in Paper I and Paper II in different contexts. Firstly, we plan for a smooth transition to a new context by keeping the same environment (built using *Microsoft tools and services*) but shifting to a new system under study, which requires collaboration with another industry partner. In this way, we would get an opportunity to examine the functionality of the smart filter in the feedback loop of another cloud-based software system. The second planned study of the thesis will aim towards the generalization of the solution

for smarter detection and management of alerting events in operations. This may include mapping of identified Microsoft Services, needed for implementation of the smart filter, with services from other cloud solution providers. Moreover, this could be additionally supported by surveying practitioners from different software development companies on the applicability of the smart filter for the same or similar problems within their monitoring platforms. Collected data may help improve and adjust the smart filter to be less context-dependent as well as govern the creation of clear guidelines for implementation of the smart filter using tools from different cloud providers. Thus, future work will be mainly focused on addressing the limitations of current research outcomes.

To conclude, we investigated challenges in continuous monitoring and monitoring data management by collaborating with the case company. The studied research area is highly connected to the industry context, thus, we wanted to identify relevant problems from the practitioners' perspective in order to design and implement solutions that will afterward be beneficial both for software development companies and the research community in terms of designed knowledge. In our future work, we aim to demonstrate that the identified problems are common for similar software systems and that the proposed solution may be used for improving different industrial contexts.

INCLUDED PAPERS

PAPER I

CLOSING THE FEEDBACK LOOP IN DEVOPS THROUGH AUTONOMOUS MONITORS IN OPERATIONS

Adha Hrusto, Per Runeson, Emelie Engström

Abstract

DevOps represent the tight connection between development and operations. To address challenges that arise on the borderline between development and operations, we conducted a study in collaboration with a Swedish company responsible for ticket management and sales in public transportation. The aim of our study was to explore and describe the existing DevOps environment, as well as to identify how the feedback from operations can be improved, specifically with respect to the alerts sent from system operations. Our study complies with the basic principles of the design science paradigm, such as understanding and improving design solutions in the specific areas of practice. Our diagnosis, based on qualitative data collected through interviews and observations, shows that alert flooding is a challenge in the feedback loop, i.e. too much signals from operations create noise in the feedback loop. Therefore, we design a solution to improve the alert management by optimizing when to raise alerts and accordingly introducing a new element in the feedback loop, a smart filter. Moreover, we implemented a prototype of the proposed solution design and showed that a tighter relation between operations and development can be achieved, using a hybrid method which combines rule-based and unsupervised machine learning for operations data analysis.

1 Introduction

The software industry has gone through several revolutionary changes over the last decades. A major change is that software is no longer delivered as a box product. Technological advancements and availability of cloud computing platforms have enabled continuous delivery of *software systems* leveraging the flexibility and reliability of various *cloud* delivery solutions [53]. Moreover, cloud providers offer an infrastructure for developing and operating large-scale software systems empowered by continuous practices and DevOps, the latest industry concept based on principles of collaboration, automation, measurements, and monitoring [60]. However, it also comes with an abundance of data to be managed as it is considered to be the *fuel* of the DevOps process [6].

The software life cycle includes continuous integration, continuous testing, and continuous deployment practices [18]. During deployment, software systems are transitioned from development to operations, to be continuously used by end-users. The connection between development (Dev) and operations (Ops), known as DevOps, ensures faster development cycles and frequent releases. However, keeping the same level of software quality becomes challenging due to shorter testing cycles. Run-time monitoring of services in operations [17], which is the focus of this study, is of high importance for gaining confidence in a software system and providing feedback to the development.

Through the run-time monitoring system, a vast amount of data is continuously collected and saved for manual or automatic analysis. The data analysis serves as feedback to development teams and provides deep and quick insight into the status of the software system during operational execution [6]. Consequently, developers and project managers can act as soon as they are notified about anomalies. The notification is typically implemented as *alerts* sent through a messaging platform, like Slack, triggered by alert rules, which are defined as functions of the operational data. However, the abundance of data and particularly alerts from minor or major malfunctions in system components, tend to flood over the developers and create noise that drowns the important alerts.

In the literature, there are examples of various methods for the analysis of operations data but only a few are addressing real industrial needs and challenges companies are facing in relation to the feedback from operations to development [71]. Consequently, there is a limited choice of potential solutions available in the literature for designing more context-specific solution designs based on the identified industrial needs. *Thus, with our research, we aim to fill this gap by addressing challenges related to the flow – and overflow – of data from operations to development.* We intend to explore and improve existing solution designs in the context of the case company's feedback loop from operations to development. Thus our study complies with the principles of a design science paradigm [56].

We conducted a study in collaboration with a Swedish company responsible for ticket management and sales in public transportation. Their main product is

the back-end system for ticketing and payments, developed and operated in a DevOps environment using Microsoft services and tools. Following design science principles, we *explore and describe* the existing DevOps environment and identify main challenges on the borderline between operations and development, using qualitative data collected through interviews and observations. To address the identified challenges, we *design* a solution for more effective processing of data available through the monitoring system in operations by introducing a smart filter in the feedback loop. Thus our research adds to the new research and innovation discipline called AIOps, artificial intelligence for IT operations [13]. Moreover, we present a *prototype implementation and validation* of the proposed design. It includes a description of the labeling process of unlabeled operations data, using unsupervised anomaly detection and considering the service vulnerabilities, as well as learning new advanced alert rules using a supervised, decision tree-based Python module.

The contributions of our paper are threefold:

C1. **Problem conceptualization.** We identified alert *targeting*, signal to noise *optimization*, and system *interoperability* as being three important problem instances of the general alert flooding problem in the feedback from operations to development.

C2. **Solution design.** We present a unique technical solution that combines various systems' and applications' metrics for learning advanced alert rules within the new element in the feedback loop, a smart filter.

C3. **Prototype implementation.** We performed a pilot implementation of the proposed solution in the case environment as a proof of concept for further work.

The rest of the paper is structured as follows. In Section 2 we present the background and previous work in this field. In Section 3 we elaborate the research approach while in Section 4 we describe the case company. Identified problem instances are introduced in Section 5. The solution proposal is presented in Section 6. Prototype implementation of the proposed solution and empirical validation are given in Section 7, while Section 8 discusses the contributions and concludes the paper.

2 Background and Related work

Ståhl et al. [60] conclude in their systematic mapping study on continuous practices and DevOps, that the concepts of continuous software engineering practices and DevOps are ambiguous in the literature. We adhere to their proposed definition that "**Continuous deployment** is an *operations practice* where release candidates evaluated in continuous delivery are frequently and rapidly placed in a production

environment". In contrast, "**Continuous release** is a *business practice* where release candidates evaluated in continuous delivery are frequently and rapidly made generally available to users/customers." Depending on the environment, a release may be achieved through deployment, for example in most SaaS (Software as a Service) environments. On the contrary, for user installed software, continuous deployment is not an applicable concept as the user must take actions to install a new version. However, continuous releases may still be offered to the users.

Ståhl et al. [60] find DevOps be a broader term, including culture and mindset. It also comprises tools, processes, and practices. We adhere to this broad definition of DevOps, as we want to investigate "the interplay between specific continuous practices and DevOps principles, processes and methods" [60], which aligns well with Fitzgerald and Stol's scoping of continuous software engineering [18].

Despite the observed ambiguity, there are additional research summaries. Laukkanen et al. [35] presented a literature review of problems, causes and solutions, when adopting continuous delivery. They build on a previous literature review by Rodriguez et al. [54], and summarize topics related to build design, system design, integration, testing, release, human and organizations, and resources. However, the operational aspects are not included. Similarly, Shahin et al. [58] do not cover practices beyond continuous deployment in their review and Mishra and Otaiwi [47] only briefly mention operational feedback as contributing to software quality in DevOps, in their systematic mapping study.

There is, however, research related to post-deployment activities. Suonsyrjä et al. [62] studied how automatically collected data from operations could be used as feedback to the development. They reviewed the literature and surveyed practitioners' interest in such activities. They conclude that topics related to post-deployment monitoring appeared in the scientific literature during the 20th century but, not during the last two decades [62]. As an exception, Orso et al. [50] presented the GAMMA system 2002, as an approach to support monitoring software's behavior during its lifetime.

Monitoring is not only focused on the software. According to Pietrantuono et al. [51], monitoring of the software product in operation can be used for collecting usage data. The data is afterward analyzed and reused for selecting the most representative test cases, based on usage profiles, which are used in their approach to "continuous software reliability testing".

Moreover, monitoring has also been part of alarm systems used for triggering warning signals in case of unusual rises in systems' metrics. Xu et al. [66] proposed a Process-Oriented Dependability (POD)-Monitor for reducing a number of false alarms focusing on sporadic and infrequent operations. Their approach utilizes process-context information and the Support Vector Machines (SVM) algorithm for learning when to suppress alarms and reduce the overload on operators.

Alerts is another term used for denoting the same or similar events as *alarms* and according to Zhao et al. [71], they represent a key source of anomalous events in operations. Zhao et al. [71] reported an approach for handling alert storms

consisting of alert storm detection using Extreme Value Theory (EVT), alert filtering using ML Isolation Forest method, alert clustering using Similarity Matrix Construction, and representative alert selection. Furthermore, Zhao et al. [72] published another study on enhancing the quality of services by utilizing the monitoring data. Similarly, they analyzed alerts but with aim of identifying the severity level. They proposed a framework AlertRank for extracting severe alerts based on textual and temporal alert features as well as features extracted from monitoring metrics. Since there are two different terms in the literature, in the rest of the paper we use *alerts* to denote signals of unexpected systems' behaviors in operations.

Monitoring in operations can be utilized even without alert rules, thus considering raw operations data. Cito et al. [11] identified three main categories of operations data: system metrics, application metrics, and application system metrics [11]. Recently, researchers and practitioners have devoted significant effort to the analysis of aforementioned operations data considering, among others, machine learning techniques and to the development of various applications. Anomaly detection is one of the available applications for early detection of a system's abnormal behavior. It has been used for detecting deviations in software releases based on the data generated by a DevOps toolchain [5]. Further, Du et al. [15] presented DeepLog, a model based on deep learning for natural language processing, which is used for learning patterns in logs and detecting anomalies in log data. More thorough research on anomaly detection has been undertaken by He et al. [25] where they provide an overview of supervised and unsupervised machine learning techniques used for log analysis. In addition, logs have been studied for several other applications. Clustering log sequences into groups, identifying causal dependencies, and creating failure rules are the main steps in the root cause analysis and failure prediction approach proposed by Fu et. al. [20].

More attempts at problem identification by log analysis can be found in papers by He et al. [24] and Lin et al. [40] where KPI (Key Performance Indicators) are used in a combination with logs. In both papers, the authors deal with clustering-based techniques, but their solutions differ in the second phase of the proposed approaches. In the solution by He et al. [24], the second phase consists of correlation analysis of identified clusters with system KPIs, while the second phase by Lin et al. [40] includes extracting most representative logs from clusters and comparison of clusters created in test and production environment for simpler problem identification. Furthermore, feedback from operations has been used for decision making and improving feature planning [11] as well as for feedback-driven development where monitoring data has been used for improving developer's tools [10].

In summary, operations data has been studied and analyzed for different purposes but still, there is more to be explored in DevOps contexts, to improve the feedback from operations to development. State of the art solutions [5, 28, 71] address relevant challenges in managing operations data. However, situations of *alert* flooding in DevOps environments are not extensively explored. Thus, we aim to contribute to the design of solutions that better manage alerts in DevOps.

3 Research Approach

Our study, as shown in Figure 1, is a problem-driven design science approach [56]. Thus our starting point was to gain deeper insights into the specific challenges of our case company. As a first step, we explored how the general problem, of incorporating feedback from operations in the development, manifests as a *problem instance* in the industrial context under study. For that purpose, we conducted six interviews and performed observations in the case company to identify and articulate the main problem instances on which to focus further improvements.

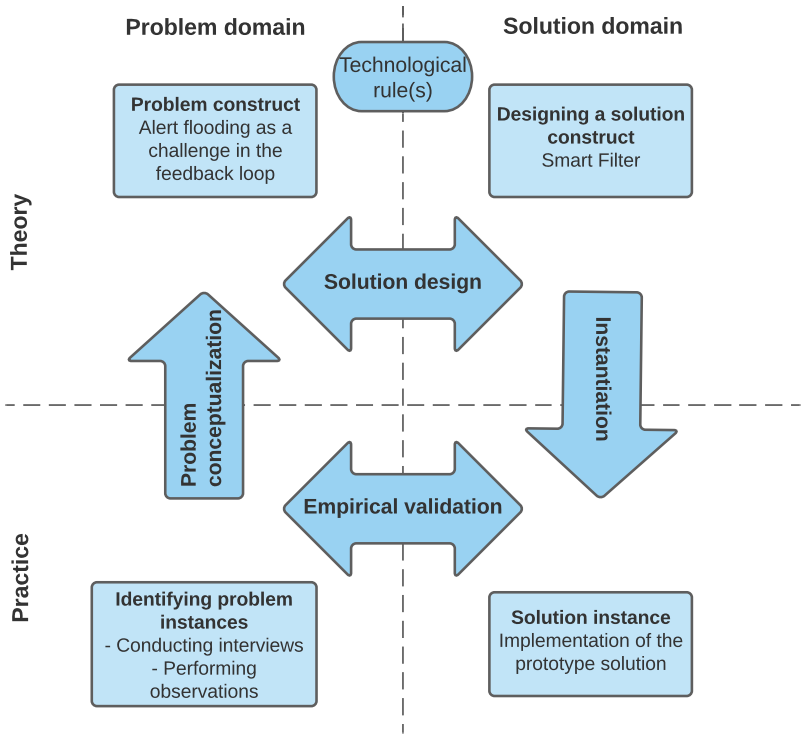


Figure 1: Overview of the design science approach

To obtain a comprehensive overview of the issues, we selected interviewees in senior positions with different responsibilities within the team including a product owner, a test manager, a test developer, a system architect, and two developers. During the interviews, we asked general as well as more specific questions related to the DevOps cycle. The interviews were semi-structured since we wanted to

flexibly explore the interviewee's opinions and let them speak about their main issues. Focus areas and examples of questions used in the interviews are shown in Table 1. All collected qualitative data, notes and video records, was analyzed using the NVivo tool. Furthermore, we observed their processes in operations and the way they were handling operations data. This enabled uncovering insights and defining problem instances.

In the *problem conceptualization* step, we described three identified problem instances (Section 5) through the lens of envisioned matching solutions, i.e. we formulated three high level technological rules. However, in this paper, we refined only one of them in the conceptual *solution design*. Hence, we improve the feedback loop from operations to development by introducing a new element, a smart filter, for optimization of alert to noise ratio. In the design process, we considered the insights gained through interviews, results of the intensive discussions with the development team, and state of the art solutions for alert management [71, 72].

Table 1: Topic areas and examples of questions used in the semi-structured interviews

Focus area	**Examples of questions**
CI/CD pipeline	- Could you describe the CI/CD pipeline? - What are the shortcomings and how can they be addressed?
Continuous monitoring	- Which parts of the system are monitored? - Which signals are the most critical and good candidates for monitoring?
Alerts	- How does the current alert system look like? - In which periods you experience the highest number of alerts?
Accessibility of operations data	- Which types of operations data are available for analysis? - Which types of operations data are used for setting the alert rules?
Potential improvements	- How/what would you improve in your current monitoring system?

Moreover, alongside the proposed solution design, we implemented a prototype *instance* to get a better understanding of the opportunities of the available operations data, its type and characteristics as well as the constraints of the context. In the implementation of the prototype solution, we used unsupervised anomaly detection throughout the labeling process of unlabeled operations data while also considering the service vulnerability and observed metrics frequency. Further, for generating new advanced alert rules, a supervised tree-based machine learning technique was used. Regarding the *empirical validation*, there were time and en-

vironment constraints that hindered a full evaluation of the implemented solution. However, we were able to perform a partial evaluation using limited data set for implementation of the multivariate anomaly detection in a prototype environment. In this way, we were able to compare the results obtained by using the smart filter in the feedback loop with the results of using the pure unsupervised ML technique for predicting alerts based on multivariate unlabeled data set.

4 Case Description

The system under study is a backend system of an application for ticketing and payments used in public transportation. It is a cloud-based system developed and operated in a DevOps environment, using Microsoft tools and services. The system architecture is leaning towards a microservice architecture which consists of 20 services that are highly maintainable, testable, and independently deployable.

Throughout the entire CI/CD cycle, shown in Figure 2, new features or updates of each service are tested on: 1) unit level, every time the build process of the system under test with its dependencies is triggered; 2) API and UI level, every time the master branch is updated as well as every night on the latest build version from the master branch. Moreover, the candidate version for the release is used as a reference version by other teams in the company for a week, which is called the "hardening process". If necessary, the latest version is tested in the acceptance-test environment which serves as a production-like environment. The release cycle is weekly and ends by deploying to three production environments. Hence, the existence of several independent environments enables smooth development, testing, and deployment activities but also multiplies the complexity of the entire system.

The health status of each service is monitored using the Microsoft data platform, Azure Monitor. Azure Monitor collects the data from several sources such as applications or Azure resources into a common platform to be used for analysis, alerting, and visualization. Within this data platform, two types of data are available, metrics and logs. Metrics are numerical values denoting specific system's observations captured within a defined timestamp. Logs are represented by both, numerical and textual values and they describe specific events that happened at a particular moment in time. Both metrics and logs can be used for setting alert rules that signalize that something unexpected is detected in the observations of the targeted resources. The case company has implemented simple rules for detecting failed requests with error 500 and unexpected raises of dependency calls and failed Http requests, as shown in Table 2. When these rules are satisfied, then alerts are triggered and alert notifications are sent either to a dedicated Slack channel or via email.

Operations data shown in Table 2, represent only a small portion of all available data in Azure Monitor but in this paper, we focus on the selected logs and metrics. Among all accessible observations of different system components, we

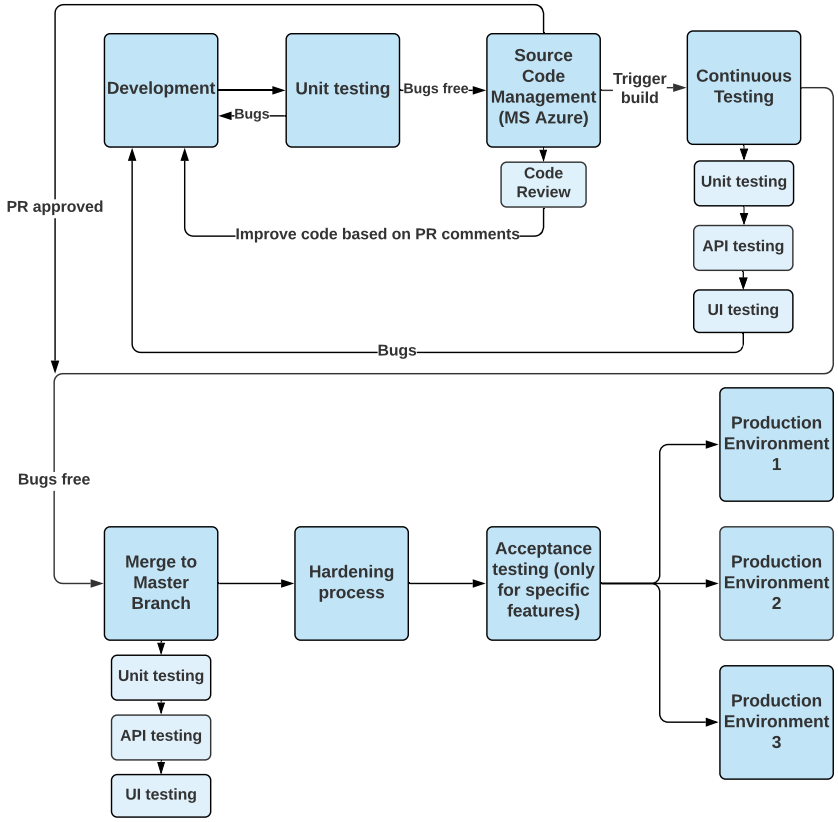


Figure 2: CI/CD pipeline

chose metrics and logs related to the data types used for setting current alert rules and the ones used in debugging in case of detected anomalies. Alert rules, shown in Table 2, are configured for all 20 services, and notifications about raised alerts are sent on two different platforms. Alerts that detect internal server error 500 are sent to the Slack channel, while unusual rises in the rate of dependency failures and failed requests are sent via email.

The development team has already reported various challenges in managing and responding to fired alerts with this configuration. Moreover, their every day development tasks are filled with the uncertainty that every alert brings into their development environment due to overload of non relevant alerts. Consequently, this might cause a bottleneck in the information flow from operations to development. The flaws, identified within the monitoring and alert system, are elaborated in the next section.

Table 2: Types of operations data mapped with configured alerts

Operations data		Configured alerts
Logs	Exceptions	/
	Traces	/
	Requests	/
Application metrics	Dependency Failures	An unusual rise in the rate of dependency failures
	Exceptions	/
	Failed Requests	/
	Server Exceptions	/
System Metrics	CPU Time	/
	Errors Http 4xx	An unusual rise in the rate of failed HTTP requests
	Server Errors 5xx	Whenever there is a server error 500
	Response Time	/
	Requests	/

5 Problem Conceptualization

In this section, we present three main problem instances, identified in the problem conceptualization step, with respect to the general goal of better incorporating feedback from operations into development. Based on observations made in the case company, *alert flooding* is identified as the main cause of all three problems. Alert flooding is a phenomenon that appears in a case of a high number of alerts that are not properly managed. In this paper, we focus on the specific aspects of this phenomenon namely, *targeting, optimization*, and *interoperability* problems.

5.1 Alert flooding as targeting problem

The first problem is defined as a targeting problem. This means that the distribution of alerts to target recipients, between the teams and individual assignment of a single or group of alerts within the team, is not fully transparent. Moreover, a lot of time is spent on discussions on how to resolve alerts and who is going to take the responsibility. Currently, there are three teams that can be assigned when an alert is fired. Each team consists of four or five members, mainly developers, and every team is responsible for one of the domains which consist of multiple services. Alert notifications are sent to a dedicated Slack channel, but no one is tagged or directly assigned to the raised alerts. Individual responsibilities within the team are not clear and team members usually discuss specific alerts in the same Slack channel. Sometimes they tag each other and ask if that person has already looked into raised alerts. As acknowledgment, they usually write that they will

look at it right away or later. If they agree that an action should be taken, a ticket is created and added to a backlog of the board in Azure DevOps. Hence, two different platforms for communicating alerts are used but the information is not synchronized.

While observing the team and their current practices, we noticed that some team members showed more interest than others in resolving alerts and that some look into alerts that are related only to services they are developing or they are familiar with. Consequently, there is an increasing number of alert notifications because no one takes full responsibility for looking into alerts that frequently appear every day. After talking to some team members, it was clear that they would like to see some structured way of alert management and assignment but they also pointed out that acting on every alert would take too much time since their main focus is development. Because of that, designing a solution for the targeting problem becomes even more challenging.

5.2 Alert flooding as optimization problem

The second problem instance represents an optimization problem, which addresses optimization of a signal to noise ratio. In this case, the signal consists of high priority alerts while the noise represents low priority alerts, which frequently appear every day. Hence, the main question is how to differentiate between alerts that cause failures and alerts that cause temporary glitches that don't affect the system's performance.

While observing the current practices in alert management, we noticed that all alert notifications come to the Slack channel with the same priority. Over time, developers learned which alerts are reoccurring occasionally, and they consider them as "normal alerts". Normal alerts are mostly caused by glitches in an external or internal service or represent a consequence of a failure related to the central service. The central service represents the heart of the system and all alerts related to this service have the highest priority. This priority is not specified as a part of an alert notification, but is something that developers know since they developed the system and they know how vulnerable each of the services is. "Normal alerts" are not normal since they signalize that something might be wrong in the specific service, but they are normal as they occur frequently, and the team got used to them. They also produce noise in the channel used for communicating alerts and because of that some critical things may pass unnoticed. The team raised concerns about this and agreed that addressing and solving this particular problem might help in faster and better response to other more important alerts. One more reason to do so is because they currently do not act upon normal alerts unless there is a high number of occurrences.

The majority of current alert rules aim at discovering internal server errors with error code 500 while a significantly higher number of logs still remain unexplored, Table 2. Hence, there is a need for adding more alert rules. However, the

team decided to stick with the existing alert rules since the current ones are not successfully managed. Recently, the team reported that they missed over 20 000 failed Http requests with error code 400. They did not notice this anomaly because they were overwhelmed with other alert notifications but also due to the fact that they do not usually analyze logs or fix issues before they cause severe problems. Hence, designing new or redesigning existing alert rules to optimize the signal to noise ratio, is another challenge that they are facing while at the same time it is important that the number of non-relevant alerts is not increased and that the most critical alerts are prioritized.

5.3 Interoperability flaws between developed system and external systems

Many large-scale software systems depend on external services developed by third parties. In this way, the original system can offer more features to their end customers. This seems to be a huge benefit but may also increase the vulnerability of the entire system since even the smallest glitches in an external service might cause serious deviations in the original system. Similar issues are experienced in the case company as their backend system also depends on external payment providers, Azure databases, and other software projects developed in their company. There is a special Slack channel where RSS (Really Simple Syndication) feeds and emails from external services are forwarded. However, many problems are still discovered through customer service and user complaints. So, they get notified when something has already failed and is visible to end-users instead of in advance. Moreover, the uncertainty of potential disruptions makes developers even more confused. It is their responsibility to decide if a raised issue is something temporary or it really represents an issue they should look into and report. They usually make a decision based on the alert frequency and side effect appearance. There are no statistics that can prove developers' claims, but a huge number of alerts are caused due to interoperability flaws with external services. The existence of failed Http responses with unknown and unexpected error codes complicates root cause analysis even more. It is important to address this problem, otherwise the system stability will be degraded.

6 Solution Design

As stated in Section 3, we provide a conceptual design for the second problem instance, *alert flooding as an optimization problem*. This problem causes the highest information overflow in the feedback loop. By addressing this specific instance, the scope of the first and the third problem instances will be reduced, and individual solutions simplified. The first and the third problem instances will not be individually treated in this paper but will be considered in our future work.

Hence, we propose one solution design and focus on the following challenges related to the second problem instance: 1) reduce the number of noisy alerts without missing the critical ones; 2) increase the number of alert rules without causing an overload of alert notifications; 3) improve developer's responses to the fired alerts while minimizing interference with their development related tasks. Accordingly, we present the overview of the proposed solution for the second problem instance in Figure 3.

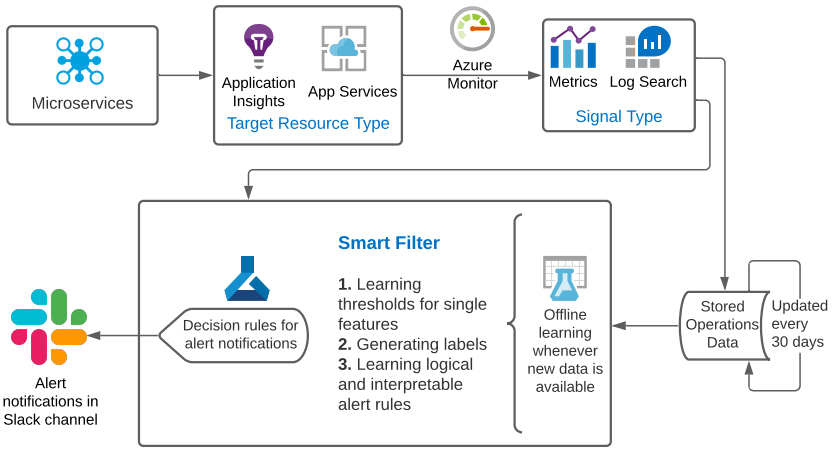


Figure 3: Overview of the proposed solution for the second problem instance

The upper part of Figure 3, illustrates the previously explained architecture of the software system, consisting of 20 micro services and Azure Monitor, that monitors real-time application performance (Application Insights) and performance of Http-based services for hosting applications (App Services). The lower part of Figure 3, visualizes the enhanced alert system with a new addition, representing the bridge between MS Azure Monitor and Slack, the platform where alert notifications are sent. The new box, the smart filter, serves as a middle-ware and provides additional features to the existing alert management.

The main task of the introduced box is to generate alert rules for sending alert notifications to the messaging platform. Hence, we temporally disregard current alert notifications and instead focus directly on the most important data, specifically metrics shown in Table 3, holding information about the system's performance. The reason for such an approach is that the current alert rules only catch a limited number of system glitches and failures while at the same time not being able to differentiate noisy alerts from important ones. The smart filter will analyze more data and learn over time to identify new dependencies that may generate new and better decision rules. In this way, we will reduce the risk of omitting

Table 3: Overview of the selected data, service vulnerabilities and desired decision rules

Selected application and system metrics	- CPU Time - Number of failed requests - Number of exceptions - Number of dependency failures - Http 4xx errors - Internal server errors - Total number of requests - Response time
Services with known vulnerabilities	- Service B –>buying tickets on vending machines - Service G –>service for validating selected locations - Service M –>main service for ticketing - Service P –>bridge to an external payment service
Example of a decision rule	IF num_of_failed_requests_SG >threshold_1 AND response_time_SB >threshold_2 AND num_of_Http500_SB >threshold_3 THEN send_notification

important alert notifications while keeping the the Slack channel clean from noisy information. Therefore, in our proposed solution design, new decision rules are learnt based on the features representing the systems' and applications' performance metrics of the mostly affected services. The output of the smart filter is binary, meaning that new decision rules are able to determine when to send and when not to send alert notifications. As shown in Figure 3, the smart filter involves preprocessing and labeling of the data required for the learning process. The exact procedure is presented in Section 7.

All things considered, the proposed approach of generating new decision rules aims at filtering the incoming performance data and sending only relevant alert notifications to the Slack channel. Newly learnt alert rules should not increase the number of alert notifications in the Slack channel since the learning process also involves learning about the noisy data.

Therefore, the proposed solution design addresses the aforementioned challenge regarding the insufficient alert rules. The purpose of the enhanced alert management is to provide more insights into correlations between alerts and operations data and at the same time enable forwarding more details about potential failures within the alert notifications. In this way, the development team could have all information needed to discover the root causes of potential failures. Moreover, it is expected that developer's awareness of raised alerts will increase and that they will need less time for resolving critical systems behaviors. Therefore, the proposed solution design intends to resolve the previously listed challenges related to

the second problem instance.

7 Prototype Implementation and Empirical Validation

In this section, we present technical details of the prototype implementation[1] as well as the effects of the implemented solution prototype in the identified problem context. Prototype implementation includes *data selection*, *tools and methods selection*, *threshold detection for each of the features*, *labeling process*, *training process* and *testing*. While working on the implementation of a solution prototype, we have decided to stick with basic machine learning techniques since we primarily wanted to examine the limitations of the suggested design. Hence, using deep learning or reinforcement learning for identified problem instances is beyond the scope of this paper.

Data selection. For the prototype implementation, we have chosen to only work with numerical values representing the various systems' and applications' performance metrics, to keep the simplicity. Logs are not included in the preliminary data selection due to their complex structure and due to the fact that the observed logs including traces, types of exceptions, or failed requests could only help with the explainability of potential failures. The metrics and services selected to be part of the training data (see Table 3) are chosen based on the observations made in the messaging and monitoring platform focusing on metrics frequency and service vulnerability. Therefore, we selected 8 metrics for each of the 11 services, which makes in total 88 features. Every feature vector has 8623 samples collected during a period of one month with a time granularity of 5 minutes, which was selected based on the current practice within the project.

Tools and method selection. The presented solution design involves learning new decision rules in the form of logical expressions "IF conditions THEN response" and for such an approach the first choice of ML methods are tree based methods, such as bagging and random forest. Therefore, for implementation, we use Skope-rules [21], a Python machine learning module for extracting rules from the tree ensemble as suggested by Friedman and Popescu [19]. The classification is binary, thus, if an instance representing the combination of multiple features satisfies conditions of the rule, then it is assigned to one of two output classes, "send_notification" or "dont_send_notification". Using this Python module requires labeled data for the learning process, thus making this approach even more challenging since the monitoring data platform collects only raw data and the knowledge about the expected outcomes is unknown.

Identifying thresholds. Therefore, we decided to generate labels based on the known service vulnerabilities and desired level of contamination. The first step

[1] `https://github.com/adha7/smart-alert-filter`, available upon request

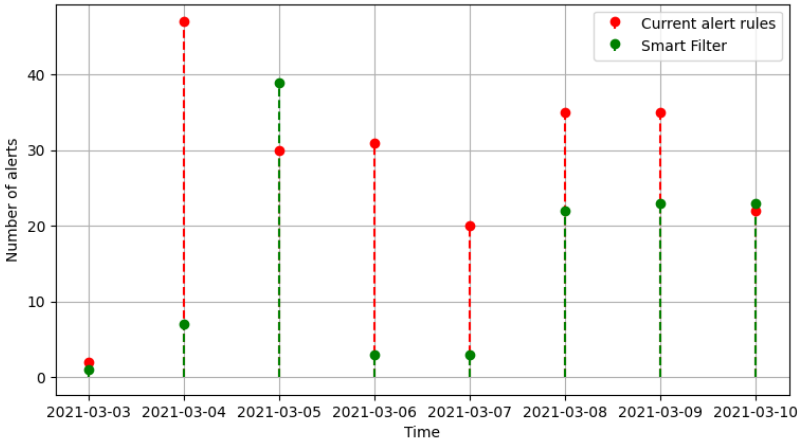
of the labeling process is to identify thresholds for single features using machine learning for anomaly detection (see Figure 3, step 1). For that purpose, we used a Python toolkit PyOD [74] consisting of 30 different detection algorithms. Hence, the thresholds are predicted for each of the 88 features where the outliers are expected to be extremely high values. By applying one of the algorithms from the PyOD module on a feature vector, we get anomaly scores for each of the values within a feature vector. Larger anomaly scores are assigned to outliers and the threshold is simply determined by picking a value from a sorted feature vector with a large enough score. The score value on the borderline between inliners and outliers is chosen so that the level of contamination of the entire training data equals 0.05. The contamination is determined by the number of outlying objects in the data set, in our case alert notifications that need to be sent to the messaging platform. Selected level of contamination corresponds to the 13 alert notifications per day and represents three times less of the current number of alert notifications. Since there is no optimal number of alert notifications per day we consider this decrease significant and at the same time large enough to not miss the important system failures.

Labeling process. After determining the thresholds for each of the features, the warnings are raised in the cases where the features reach values above these border values. Based on these warnings, we generate labels (see Figure 3, step 2) considering a fixed number of raised warnings in a time slot of 5 minutes as well as capturing for which services warnings are raised, targeting services shown in Table 3. Accordingly, the output class is labeled as 1, if there are more than 8 raised warnings in the same time slot, which means that there are at least two services affected considering that 8 warnings can be related to one service. Further, the output is also denoted as anomalous or 1, if there are warnings raised for the most vulnerable services, as shown in Table 3, no matter the number of raised warnings. When the labeling process is completed, learning logical and interpretable alert rules can be activated (see Figure 3, step 3).

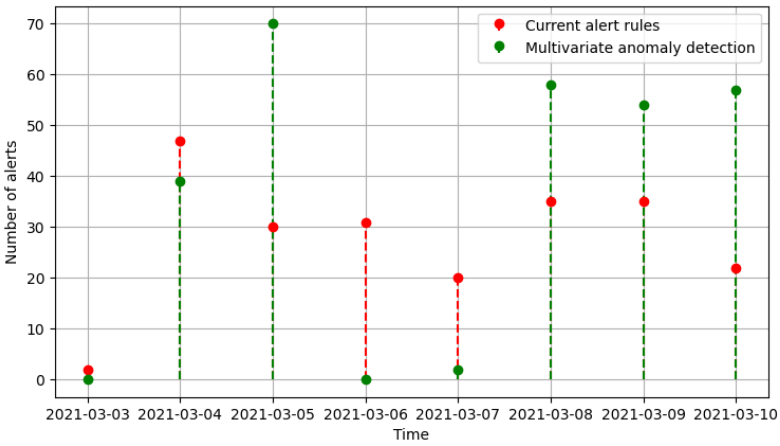
Training process. Through the training process, Skope-rules generated 120 rules for the class "dont_send_notification" and 43 rules for the class "send_notification". The rules are generated by fitting single estimators, decision trees, with predefined precision and recall as input parameters. The precision and recall reached during the training phase are between 0.92 and 0.99 for the output class "dont_send_notification". The precision score for the output class "send_notification" is evenly high as for the opposite class but the recall was significantly lower due to very low contamination, the number of outliers, in the training data set. A low recall score makes the algorithm "picky" when selecting outlying samples which might be good for filtering the noise but on the other hand, it might miss single and isolated outliers.

Testing. On this account, we analyze how the implemented prototype scales the number of predicted alert notifications per day to the actual number of raised alerts. We use test data collected within the 7 days (March 3, 20:35 – March 10,

19:40) for predicting outlying objects, alerts, and present the results in Figure 4.



(a) Smart filter



(b) Multivariate anomaly detection

Figure 4: Number of alerts per day in the test data. RED color: alerts raised with current alert rules; GREEN color: alerts raised with a) the smart filter and b) multivariate anomaly detection

We conclude that the smart filter produces half the number of alerts in a period of 7 days, 108 compared to 211. Regarding the distribution of alert notifications per day, the number of predicted alerts during the weekend (March 6 and 7) is very low which is expected due to lower stress on the ticketing and payments system. During the workdays, the number of predicted alerts is less than actual except when there are issues in the system that the current alert system is not able to capture.

This was the case on March 5, when there was a problem with buying tickets on the vending machines. The smart filter raised an alert 30 minutes earlier than it was reported by customers, which means that this specific failure could have been caught before it was noticed by users.

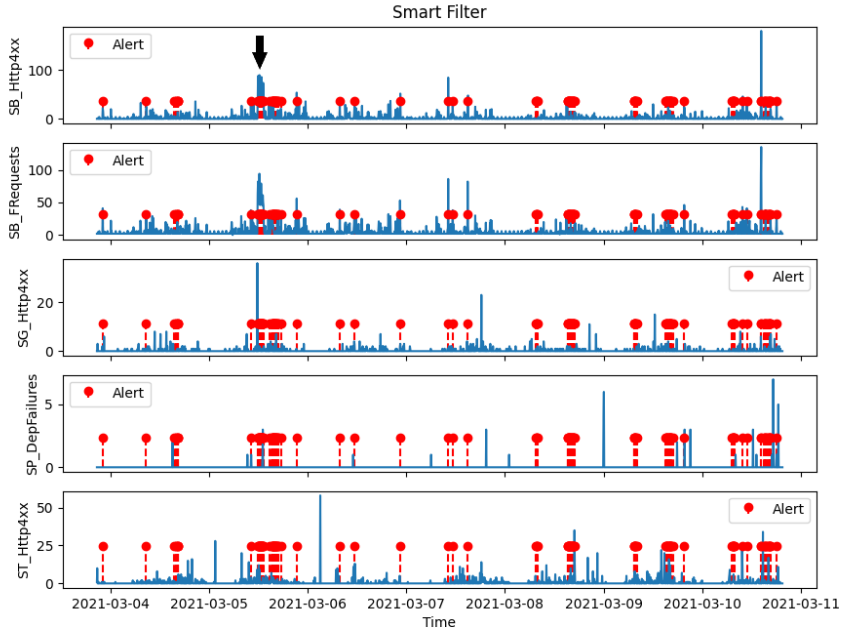
The implemented prototype reduces the overall overload on the development team but also gives space for further improvement by introducing prioritization of alerts and sending the alerts on different Slack channels based on their priority for even better and clearer differentiation.

Empirical validation. In addition to the smart filter implementation, we also implemented multivariate anomaly detection (MAD) to validate our prototype by comparing it with the pure unsupervised ML technique for detecting outliers, representing alerts, in multivariate unlabeled data set. We used the same Python toolkit PyOD [74] for the MAD implementation and selected the COPOD model, copula-based outlier detection introduced by Li at al. [38]. The COPOD model was trained using the same training data but without labels. The predictions, shown in Figure 4 (b), using the same test data set, revealed that the MAD trained model does not scale very well the number of predicted alerts. It predicts almost the same number of alerts as the actual alert system, making the same level of noise. Both models, trained using the smart filter and MAD respectively, reach the F1-score, a harmonic mean of precision and recall denoting a model's accuracy, above 0.9. However, the pure unsupervised ML might not be able to capture the imbalance between the target classes and the importance of specific services and their metrics. To clarify this, we look at the alert distribution over the metrics of highly affected services shown in Figure 5 (a) and (b). We noticed that the smart filter produces less noise around the actual failures, such as the one marked with the black arrow from March 5. This means that the actual failure can be more easily identified among the alerts that appear close to the selected alert on the graph. The predicted alerts using multivariate anomaly detection are grouped and based on the graph, they produce several alert floods which is the opposite to what we want to achieve. On the other hand, the smart filter predicts isolated alerts in case of short system's glitches and smaller groups of alerts when there is a larger issue rolling out.

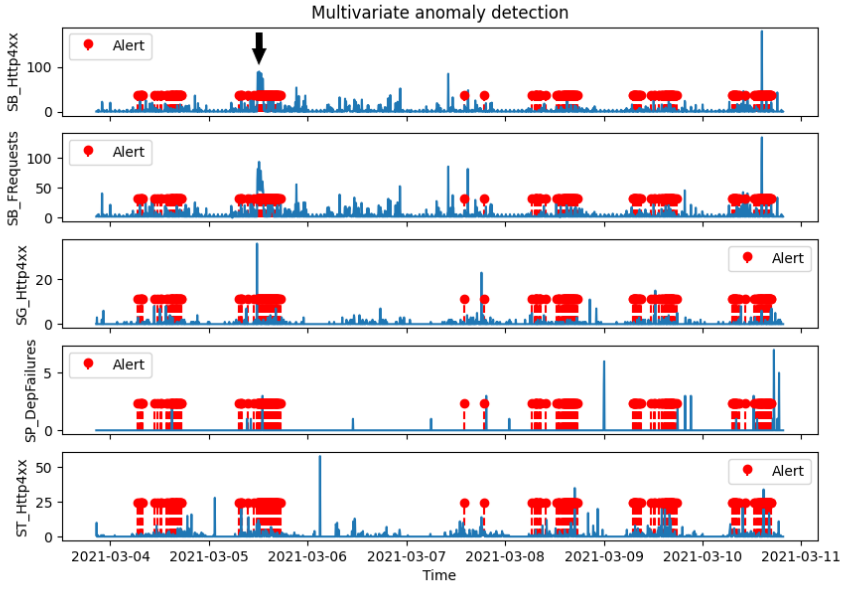
There are still some individual events that passed unnoticed but since this is only a prototype version, imperfections and shortcomings are expected. Furthermore, we used a limited data set collected within one month, which could have also affected the training process and learning when to send alert notifications due to a low number of outlying objects. We aim to address this in our future work by considering the larger data set.

8 Discussion and Conclusion

The synergy between development and operations in DevOps is important for developing and releasing high-quality software systems, but even more for gaining



(a)



(b)

Figure 5: Distribution of raised alerts in the test data using a) the smart filter and b) multivariate anomaly detection. BLUE color: selected performance metric; RED color: raised alerts

insights into the system's behavior in the production environment. In order to ensure the latter, raw operations data, collected through runtime monitoring tools, is analyzed to discover valuable feedback information. Our results have shown that monitoring and utilizing data available in the production may help developer teams to more easily identify, understand and communicate issues in the operations. Further, it helps present the valuable information in an actionable manner and reduces the pressure and overload.

The results obtained, following design science principles, directly relate to three main contributions mentioned in the introduction section, problem conceptualization (C1), solution design (C2), and prototype implementation (C3). We started with the problem conceptualization since the first step in solving a particular problem is understanding its causes and effects. Before our attempt to identify the main challenges on the borderline between development and operations, the everyday routine work at the case company obscured shortcomings in the information flow between operations and development. During the initial stage of interviews and observations, we managed to identify *targeting, optimization* and *interoperability* problem instances related to *alert flooding*. The problem conceptualization (C1) helped both the development team in acknowledging existing issues and the research team, in creating a solution design, which is our second contribution. After presenting our findings, the development team seemed relieved since they finally understood what was hindering them from making full use of operational data and how data overload in operations could be prevented.

The solution design (C2), as previously mentioned, addresses the problem of alert flooding with the emphasis on reducing the number of noisy alerts. The presented conceptual model includes a new element in the feedback loop, responsible for learning new advanced alert rules capable of reducing the total number of alerts and increasing their relevance. The smart filter addresses challenges in the alert management such as insufficient number of alert rules, noisy alert notifications, and slow developer's response on fired alerts. Therefore, this addition in the feedback loop improves the information flow from operations to development by introducing alert rules which combine various systems' and applications' metrics and services with the aim of capturing unexpected and faulty system's behaviors and providing more detailed insights to the development team.

The third contribution (C3) includes implementation of the solution prototype and validation in a specific context, i.e. our case, the ticketing and payment system operated in the DevOps environment. We successfully implemented a prototype version of the smart filter using a hybrid method consisting of unsupervised anomaly detection and supervised decision tree-based Python toolkit while also considering the importance of highly vulnerable services in the labeling process. The prototype was validated using a limited test data set collected through the monitoring system in the production environment. Accordingly, we demonstrated that a severe failure could have been caught if the smart filter was integrated in the feedback loop instead of the current alert system. Furthermore, we compared

the implementation of our prototype with the pure unsupervised ML technique for multivariate anomaly detection. We showed that the customized hybrid method better captures the systems' unbalanced operations data and system-specific characteristics needed for catching both systems' glitches and severe failures. Hence, the feedback information obtained as a final result has tightened the connection between operations and development. There have been several attempts at addressing similar challenges using state of the art solutions based on deep learning [15, 28, 71], while our solution proposal reach promising results while keeping simplicity of the ML approach.

The smart filter in the feedback loop improves the connection between operations and development but at the same time raises more challenges that need to be addressed in the future. Even though it reduces the total number of alerts, it could still be improved by increasing the level of differentiation between the raised alerts by introducing several levels of priorities and target recipients. We plan for further work to address the raised challenges by considering deep learning and other machine learning techniques as well as implementing the smart filter in the production environment. Consequently, the smart filter will be fully integrated and automated in the feedback loop and will require minimum human assistance. In this way, we would be able to get immediate feedback and insights from developers involved in the alert management, which is needed for obtaining a complete evaluation of the smart filter. Moreover, since our study provides prescriptions for problems in a very specific industrial context, in the future we aim to validate our solution in other similar contexts.

PAPER II

OPTIMIZATION OF ANOMALY DETECTION IN A MICROSERVICE SYSTEM THROUGH CONTINUOUS FEEDBACK FROM DEVELOPMENT

Adha Hrusto, Emelie Engström, Per Runeson

Abstract

Monitoring a microservice system may bring a lot of benefits to development teams such as early detection of run-time errors and various performance anomalies. In this study, we explore deep learning (DL) solutions for detection of anomalous system's behavior based on collected monitoring data that consists of applications' and systems' performance metrics. The study is conducted in a collaboration with a Swedish company responsible for ticket and payment management in public transportation. Moreover, we specifically address a shortage of approaches for evaluating DL models without any ground truth data. Hence, we propose a solution design for anomaly detection and reporting alerts inspired by state-of-the-art DL solutions. Furthermore, we propose a plan for its in-context implementation and evaluation empowered by feedback from the development team. Through continuous feedback from development, the labeled data is generated and used for optimization of the DL model. In this way, a microservice system may leverage DL solutions to address rising challenges within its architecture.

1 Introduction

The complexity and dimensionality of software systems are continuously increasing to meet higher market demands and customer expectations. In order to stay competitive, software companies have started adopting new and modern system architectures that enable, among other things, development of scalable and reliable applications. The recent and popular architectural style entails microservices as an approach towards developing applications as a suite of small services. The main advantage of those services is that they are independently developed, tested, deployed and maintained [63]. Some microservice systems can be considered as a kind of a System-of-Systems (SoS) since they share a common origin from monolithic system architectures while several defining features of microservices such as componentization, decentralized governance, and infrastructure automation, are foundations of the SoS main properties, including operational independence, geographical distribution, and evolutionary development [12].

In this paper, we specifically focus on infrastructure automation for building, deploying, and operating microservices, known as Continuous Integration/Continuous Deployment (CI/CD) pipeline [63]. The overall popularity of microservices has increased with the availability of various cloud platforms and solutions that enable flexible and frequent releases. Developing microservices is even more empowered in DevOps environments where collaboration, automation, measurement, and monitoring [44] are the main principles for eliminating the barriers between development and operations teams. The DevOps concept of continuous monitoring in operations assists in revealing unexpected and unwanted system's behavior.

We perform the study under the design science paradigm [56], as a continuation of our previous work [26] which conceptualized the problem and proposed a solution, expressed as technological rules, i.e. grounded recommendations for practice [56]. More precisely, we dig deeper into vulnerabilities of a microservice system and how they may be discovered early by raising alerts before they turn into a problem with severe consequences, which we refer to as *alert management*. Monitoring in operations provides direct insights into the status of the running microservice system while all monitoring data is continuously saved for potential further analysis. Many software companies rely only on visualization tools for tracking the current health of their system. However, the visual representation may not be enough for timely identification of possible anomalies and transferring the alert notifications to development. In our previous study [26], we addressed this issue by exploring how utilization of the operations data may tighten the feedback from operations to development and help the development teams to identify and communicate alerts with less pressure and overload. In this study, we further advance the design of the proposed solution and its evaluation, to meet the complexity of the studied system and unavailability of annotated anomalous data.

The services in our microservice system are independently managed, thus their monitoring component and data storage are also self-contained. However, the en-

tire software system is based on the interaction between the services, which may not only be dependent on each other but even on another external service, hosted on a different cloud platform. Accordingly, the health status of the entire system not only depends on the health of individual services but their relation and coexistence in a software system as a whole. To address this complexity, we treat the data from different services as multivariate time series (MTS) data represented by an ordered set of multidimensional vectors, recorded at a specific timestamp [8].

There have been several attempts at dealing with multivariate time series data with the aim of identifying anomalous system's behavior [28, 36, 46]. However, only a minority discuss the evaluation of the proposed methods in contexts without known ground truth data. Evaluation of the learning model is a required step to achieve a reliable AI solution [34]. Thus, we aim to fill this gap and explore approaches for evaluation of deep learning solutions for anomaly detection in unlabeled multivariate time series. The novelty of the conducted research is in *the adaptation to the real industrial context*. More specifically, we defined guidelines for selecting and evaluating DL solutions in the aforementioned context. Expressed as a technological rule:

> To improve alert management in DevOps environments, integrate a smart filter based on DL for anomaly detection in operations and optimize it utilizing generated labeled data through feedback from development.

The technological rule above is a refinement of the one identified in our initial exploratory study [26]. The advanced solution design builds on the problem conceptualization in that study and is instantiated in the same DevOps environment (see Section 4). In this paper, we investigate the DL methods for anomaly detection in multivariate time series and how to utilize feedback from development to simultaneously *evaluate* and *optimize* the DL model by iteratively learning when to raise alerts. The contributions of our paper are threefold:

- **C0**: **A conceptualization of the problem** of applying and evaluating deep learning methods in a challenging industrial context that includes multidimensional time series data with unknown ground truth anomalies.

- **C1**: **A brief overview of unsupervised deep learning methods** for anomaly detection in multivariate time series representing performance metrics of a microservice system as well as guidelines for selecting minimum feasible DL methods for anomaly detection in the same or similar contexts.

- **C2**: **A cloud solution plan** for deployment of the DL method for anomaly detection in multivariate time series and its in-context evaluation and optimization through continuous feedback from development.

The rest of the paper is structured as follows. In Section 2 we present background and related work in the field of monitoring large-scale cloud applications and anomaly detection. Next, in Section 3 we give an overview of the research approach while in Section 4 we describe the case company and the system under study. We present an overview of unsupervised deep learning methods for multivariate time series in Section 5 and give guidance on selection minimum feasible DL methods in Section 6. A detailed plan for integration of the improved solution proposal into the feedback loop is given in Section 7. Finally, we conclude and briefly discuss future work in Section 8.

2 Background and Related Work

The general background on monitoring microservice systems may be found in publications that explicitly treat this type of systems [9, 30] as well as in papers that rather focus on cloud monitoring while implicitly introducing challenges on monitoring microservice architectures [23, 53], due to their very strong synergy. Ying et al. [30] claim that monitoring has an important role in a microservice system and that it can be conducted on several levels: hardware, network, system, application, and service access level. They propose a monitoring scheme that consists of the following platforms: 1) Logstash for data collection and filtering; 2) Elasticsearch for storage analysis; 3) Kibana for visualization. In a broader overview on monitoring of microservices, Waseem et al. [65] introduce monitoring practices such as exception tracking, health check API, and log management instead of monitoring levels. Moreover, they bring reasons for having a monitoring infrastructure for microservices, including diagnosing and reporting errors, failures, and performance issues, which is also the aim of our study.

From a slightly different perspective, focusing on monitoring cloud systems, Pourmazhangjidi et al. [53] present a similar monitoring framework as Ying et al. [30] with an additional suggestion that includes a stack of InfluxDB and Grafana, open-source time series database, and analytics and interactive visualization web application, respectively. Moreover, one of the challenges they are concerned with is that the monitoring frameworks provide the health status of individual components instead of the overall system [53], which is a concern we are partially addressing in this study.

Further, there is a need for a unified monitoring framework to replace a stack of different tools as in the aforementioned examples. Larger companies providing the cloud services such as Microsoft and IBM [26, 28] have already started offering a unified platform that enables version control, building, deploying, and monitoring of applications.

Data collected through the monitoring system may take different forms, but we specifically focus on multivariate time series, mainly denoting various applications' and systems' performance metrics. Mining the time series data aims at

extracting meaningful knowledge using the tasks such as classification, clustering, forecasting, and anomaly detection [8]. Anomaly detection is the process of identifying unusual and unexpected events across time [3]. Hagemann and Katsarou [23] summarize anomaly detection methods for cloud computing environments into three categories: machine learning, deep learning, and statistical approaches. Recent studies [28, 46] show that the researchers were mainly interested in deep learning methods due to their ability to learn highly complex and non-linear correlations with no prior assumptions about the data. For instance, Islam et al. [28] argued that Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) autoencoders have the strongest predictive power. Among other researchers, there is the same tendency when selecting anomaly detection methods for multivariate time series [1, 28, 46]. Another method attracting growing attention is Generative Adversarial Networks (GANs) [2, 36] that belongs to reconstruction-based methods like the aforementioned autoencoders. Similar to autoencoders, GANs aim at reconstructing the normal behavior of time series and detecting anomalies based on how much the reconstructed data points deviate from the normal behavior. In Section 3, we provide an in-depth review of deep learning methods for the studied context.

3 Research Approach

As stated in Section 1, we aim to further investigate one of the technological rules from the previous study [26] that maps the problem of alert flooding with a solution that optimizes alert to noise ratio by adding a smart filter to the feedback loop from operations to development. Hence, as the starting point of this study, we revisited the current version of the smart filter by discussing its limitations with practitioners and surveying relevant literature. In a design science cycle [56], this step corresponds to the problem conceptualization and it is directly related to contribution **C0** in Section 1. In other words, we reduced our challenging industrial context to a problem of learning from multidimensional time series data with unknown ground truth anomalies.

Next, we investigate solutions for an upgrade since the current implementation of the smart filter relied on a supervised anomaly detection method and manually labeled data, which was insufficient to fully characterize all notions of anomalousness [55]. Hence, we wanted to explore unsupervised deep learning approaches that are capable of learning the complex dynamics with no prior assumptions about the anomalous and non-anomalous data distribution. Thus, in this step we design an upgraded solution of the smart filter inspired by available solutions in the literature, which is stated in contribution **C1** in Section 1. Hence, we approach the literature with the following research questions:

- **RQ1**: What unsupervised deep learning methods are mostly used for anomaly detection in multivariate time series?

- **RQ2**: What unsupervised deep learning methods for anomaly detection in MTS may be used in the studied context?

Eighteen techniques were compared through a 3D lens proposed by Choi et al. [8]. The details of this analysis are reported in Section 5 and summarized as generic guidelines in Section 6.

Further, we plan how to integrate the upgraded deep learning version of the smart filter in the feedback loop from operations to development and evaluate it, without having any known anomalies to compare it with. This step was mainly empowered by the first author's practical experience and through the support of the practitioners in the case company. This step is a preparation for actual solution implementation and evaluation and intend to answer the following research question related to contribution **C2** in Section 1:

- **RQ3**: What cloud infrastructure is required to implement the proposed solution design for optimization of anomaly detection in a microservice system?

In Section 7 we present the proposed solution design in its evaluation context.

While completing the mentioned design steps provides us with a solution design and a plan for its in-context implementation and optimization, the full-scale execution of the empirical evaluation will be carried out in future work.

4 Problem Context

This study is conducted in a collaboration with a Swedish company, developing and operating the backend system of an application for ticket and payment management in public transportation. It is an example of a system that utilized benefits from both a cloud platform and a microservice architecture. The entire system is developed using Microsoft tools and cloud solutions, following the aforementioned DevOps principles. The health status of each of the 20 microservices is monitored using Microsoft Azure Monitor that collects data, including performance metrics and logs from various sources, such as applications and Azure resources. Collected data is mainly used for analysis, visualization, and alerting. For further analysis of anomaly detection, we consider only metrics – numerical representations of specific systems' observations taken within a predefined timestamp. Logs are disregarded in this study due to their complexity and due to a fact that logs' attributes such as target resources, types of resources, various operations' IDs could only enable more detailed explainability of potential failures, rather than detection of anomalous behavior.

As stated in Section 1, it is important to take into account performance metrics of all monitored microservices and analyze dependencies over a time axis but even more importantly dependencies on other variables' observations from other microservices. Hence, we consider *application metrics* (dependency failures, exceptions, failed requests) and *system metrics* (errors Http 2xx, errors Http 4xx, errors

Http 5xx, number of requests, response time) across *14* and *13* different services, respectively. We selected those services based on their vulnerability and significance identified during the initial study [26]. In total, we get a 120-dimensional vector consisting of 120 real-valued observations denoting specific performance metrics, which is an instance of a multivariate times series. The training data set is composed of samples recorded at a specific time during a period of 6 months. Moreover, the training data set is contaminated with approximately 1% of undetected anomalies which makes the two classes of anomalous and non-anomalous data highly imbalanced.

We have already attempted at analyzing smaller data set consisting of the aforementioned performance metrics in order to address reported challenges in managing alerts, that notify the developer team about strange and unusual system's behavior. However, creating new advanced decision rules [26] that combine metrics across different services is a prototype solution used for exploring the limits of this very complex industrial context. Therefore, we aim to explore deep learning methods, since they can learn highly complex dynamics with no assumptions about the data distributions and underlying patterns. A detailed analysis of deep learning methods applicable for the multivariate time series is presented in the following section.

5 Review of DL Methods for Anomaly Detection in MTS

We reviewed eighteen recent studies proposing deep learning methods for anomaly detection in time series. Inspired by a review by Choi et al. [8], we analyze and present a condensed overview of unsupervised DL methods across three dimensions: 1) inter-correlation between variables, 2) temporal context modeling, and 3) anomaly score criteria. The first dimension covers the various methods that may be employed for calculating the correlation between multiple variables, such as dimensional reduction, 2D matrix, or graphs [32, 37, 69]. Thus, high-dimensional monitoring data may be represented with fewer feature representations in order to reduce the problem dimensionality and the number of computing resources needed for the analysis of the raw data. The second dimension considers the temporal context of the time series and it is defined by the selection of the neural network architectures, such as Recurrent Neural Networks (RNN) [33], Long Short Term Memory (LSTM) [69], Gated Recurrent Unit (GRU) [70], or Convolutional Neural Networks (CNN) [29]. The third dimension relates to the calculation of the anomaly score that indicates the levels of anomalousness. The greater the score is, the more likely it is that the observed time series sequence is abnormal. The anomaly score can be calculated based on the reconstruction error [28], prediction error [69] or dissimilarity [29]. The mapping of DL methods for anomaly detec-

tion in multivariate time series across the aforementioned dimensions is shown in Table 1.

The first column in Table 1 classifies DL methods based on the reconstruction error along the inter-correlation between variables and different neural networks for modeling temporal context. The reconstruction-based anomaly detection methods leverage the encoder-decoder architecture, namely autoencoders (AE), to reconstruct input time series and measure how much the reconstructed time series deviate from the original samples. An encoder is a fully connected neural network that encodes a time series sequence into a lower dimension to obtain the most representative features. A decoder uses the output from the encoder to reconstruct the encoded time series into the original dimensions. An autoencoder aims to reproduce the input time series with noise and anomalies removed, since the hidden layers learn to generalize the distribution of the normal data [33]. This architecture has been used as a single solution for addressing anomaly detection problems [27] as well as in combination with Generative Adversarial Networks (GANs) [22] or anomaly likelihood functions, as introduced by Ahmad et al. [1].

Autoencoders have been widely used in various setups. Kieu et al. [33] proposed autoencoder ensembles by building a set of RNN autoencoders that may be trained in an independent framework or in a shared framework, where all autoencoders in an ensemble interact through a shared layer. In this example, the anomaly score is calculated as the median of all reconstruction errors obtained by applying the Euclidean norm to the original and reconstructed time series. This approach successfully deals with the problem of overfitting to the original time series [33]. Zhang et al. [68] and Zhao et al. [73] presented similar solutions that include constructing 2D feature matrices needed for representing the inter-correlations between the pairs of time series before applying convolutional encoders and decoders for reconstructing the input time series. Benefits of combining CNN and LSTM layers in autoecoders for learning spatial and temporal features have inspired many researchers [31, 67]. Unlike the aforementioned solutions, Chevrot et al. [7] explored using several decoders, one per mini batch. The mini batches were created by separating the original batch of data based on the discrimination feature, which simplifies the solution but provides better accuracy and timely anomaly detection.

As previously mentioned, the encoder-decoder structure has been used in GANs solutions, mainly to model a *Generator*. Hence, the generator also aims at reconstructing time series data and generating samples that possibly could have been drawn from the original data set. In addition to the Generator, GANs consist of another sub-model, a *Discriminator*. The Discriminator learns to distinguish the real samples from the original data set and fake samples obtained from the Generator. The two GAN sub-models are trained together with the objective of minimizing the adversarial loss to match the distribution of the generated time series to the data distribution of original samples [22]. Geiger et al. [22] applied the GAN approach on normalized raw data while calculating the anomaly score as a linear combination of reconstruction errors and outputs from the Discriminator.

Table 1: Review of unsupervised deep learning methods for multivariate time series across three dimensions: 1) Anomaly criteria (reconstruction error, prediction error, dissimilarity); 2) Inter-correlation (dimensional reduction, 2D matrix, graph); 3) Modeling temporal context (LSTM, GRU, CNN, RNN)

	ANOMALY CRITERIA									
	Reconstruction error					Prediction error			Dissimilarity	
INTER-CORRELATION	LSTM	GRU	CNN	RNN	CNN+LSTM	LSTM	GRU	CNN	RNN	CNN
Dimensional reduction	[37], [7]		[69]			[69]			[59]	[42]
2D matrix	[32]		[29]		[68], [73]					
Graph		[70]					[70]			
Other	[22], [27]	[28]		[33]	[31], [67]	[14]		[48]		

Differently from the aforementioned approach, Khoshnevisan et al. [32] and Wenqian et al. [29] used a 2D correlation matrix to capture inter-correlation between multiple time series and explore the most representative features before applying the GAN method. Both approaches use the same Generator structure, encoder-decoder-encoder, to optimize the input reconstruction in original and latent space. The approaches differ in modeling the temporal context and calculating the anomaly score. Khoshnevisan et al. [32] built the corresponding neural networks out of the convolutional-LSTM layer in order to learn spatial and temporal dependencies while Wenqian et al. [29] relied only on the convolutional layers. In the testing phase, Wenqian et al. [29] used both the apparent and latent loss, a L_1 distance between the real and generated time series, and Euclidean distance between latent representations of the original samples and encoded generated samples, respectively. Differently, Khoshnevisan et al. [32] determined the anomalous samples based on the reconstruction errors as a measure of differentiation between the original and reconstructed time series. Although the results of the GAN approaches seem remarkable as in previous and similar examples [37], simultaneous training of two competing sub-models may lead to a failure mode. Thus, a very unstable training process may cause a collapsing mode in which the Generator will always output the same value for any input time series.

The second column in Table 1 lists the DL methods for anomaly detection based on prediction error. These methods predict the values of the time series for the next time steps and calculate residuals between the predicted values and the actual observations. Examples of such methods were reported by Munir et al. [48] and Ding et al. [14] with a slightly different choice of neural networks, CNN and LSTM respectively, and different selection of the metrics for calculating the anomaly score. In addition to the prediction models, Zhao et al [70] and Zhang et al. [69] consider prediction and reconstruction errors mutually in their model architectures. Zhang et al. [69] proposed a Convolutional Autoencoding Memory (CAE-M) built of a convolutional encoder whose feature representations were fed into the predictive network. Zhao et al [70] used the feature-oriented and time-oriented graph attention layer to model the relationship between the features and time steps, respectively, before simultaneous optimization of the reconstruction and predictive model. By combining reconstruction and predictive models, the aforementioned solutions bypass the shortcomings of the individual models [70].

Dissimilarity-based methods have been less appealing to the researchers, thus, the third column in Table 1 contains fewer DL methods in comparison to the other columns. However, the examples of dissimilarity-based methods have shown promising results in tackling anomaly detection in multivariate time series [42,59]. The core idea behind these methods is to measure the distance between the value obtained by the DL model and the distribution or cluster of the original data set [8]. In particular, Liu et al. [42] proposed an architecture that consists of a feature extractor and anomaly detector. The feature extractor is a stack of CNN layers used for extracting a low-dimensional feature vector while the anomaly detector uses

Mahalanobis distance to calculate how far the current observation exists from the distribution of normal and abnormal data. Similarly, Shen et al. [59] used feature extractor in their temporal hierarchical one-class (THOC) model but differently, cosine similarity to measure the distance between the features and the cluster obtained by deep support vector data description.

As previously mentioned, this is a brief overview of the DL methods for anomaly detection in multivariate time series. It encompasses state-of-the-art DL methods that one may consider when tackling anomaly detection problems. In Section 6 we discuss the selection of the minimum feasible DL method in terms of simplicity and applicability for our and similar problem contexts, while mainly reflecting on the methods and classification presented in this section.

6 Guidance for a Minimum Feasible DL Method

Most of the currently available deep learning approaches for anomaly detection in multivariate time series are context-specific and their implementation may seem overwhelming for researchers or practitioners that are novices in the field. Moreover, there is no single solution that fits all use cases. Thus, in this section, we aim to give basic guidelines for the selection of a minimum feasible deep learning method for anomaly detection, specifically applicable for multivariate time series. This means that the suggested method may be used as the starting point when exploring the specific context and limitations of the available data set. Further, the method may be adjusted or optimized to be used for larger and more complex data sets.

Type of anomaly. The anomaly in time series data may be represented by one or more data points that significantly deviate from previous time steps. We consider a point and subsequent anomaly [3], as two types of anomalies relevant for our problem context. The point anomaly is a data point that significantly deviates from either its neighboring data points (local anomaly) or other points in the time series (global anomaly). The subsequent anomaly refers to a set of data points whose mutual behavior diverges from the rest of the time series [3]. Both types of anomalies may affect one or more time-dependent variables in time series. To detect either of these two anomaly types, the DL methods listed in Table 1 may be applied with an additional modification for detecting subsequent anomalies since the history of detected anomalies must be maintained. Thus, the methods including sliding windows may be explored when detecting subsequent anomalies [28, 68].

Dimension of the data set. Dimensionality in time series refers to a number of attributes measured in each time step. A multivariate time series is an ordered set of n-dimensional vectors recorded at a specific time, where n denotes the number of attributes, which can be equal or greater than two [8]. As presented in Section 5, various methods for dimensional reduction, including autoencoders (AE) [69] and convolutional feature extractor [42], may be employed for extracting

features based on the relationship among attributes and reducing the dimensionality of the problem context.

Temporality. A time series is a collection of data points measured and indexed in time order. The observations are recorded at equal time intervals and each data point is dependent on its prior values. Based on the review in Section 5, LSTM layers are mostly used for modeling the temporal context since they have ability of keeping the information about previous states for long periods of time. This enables learning of the long-term dependencies. Further, CNN layers were a favoured selection for extracting spatio-temporal dependencies in multivariate time series with a spatial dimension as in [68, 73], which is usually not the case with the performance monitoring data.

Selection of the DL method. For the selection of the minimum feasible methods, we exclude solutions based on GANs since they may require more development time and computing resources for their implementation but still do not guarantee satisfactory results due to their unstable training process. Thus, we rather focus on reconstruction- and prediction-based methods to be the first methods to explore when tackling the anomaly detection problems in time series. Further, we present three variations of the DL methods that could be explored as the starting point when addressing anomaly detection tasks.

- **Method 1:** A pure reconstruction-based method utilizing the encoder-decoder architecture for obtaining a reconstruction of the input data. For modeling the temporal context, LSTM layers are preferred while anomaly scores may be calculated as the Euclidean distance between original and reconstructed data points. Moreover, one could use fixed window size to capture subsequent anomalies while keeping track of the previously detected anomalous data points as suggested by Hsieh et al. [27].

- **Method 2:** A combination of reconstruction- and prediction-based methods as proposed by Zhao et al. [70]. Inter-correlation may be applied in the case of high dimensionality [32] with previous data normalization. The suggestion is still to stick with the LSTM layers and use the Euclidean norm to calculate the difference between reconstructed and predicted values, and actual observations at the corresponding time steps.

- **Method 3:** A dissimilarity-based method with prior feature extraction using one of the approaches for discovering inter-correlations between attributes, such as LSTM autoencoders [8] or CNN feature extractor as in a solution proposed by Liu et al. [42]. The next step is to determine how far actual observed values are from the clusters or distribution of anomalous and non-anomalous samples using any clustering method, such as K-Means [55] or Gaussian Mixture Models [42].

We suggest exploring the methods in the order they are presented. The three presented methods will be considered for advancing the smart filter. Since only one

of the methods can be deployed to production, we will start with the first method. When the labeled data is obtained based on the feedback from the development team, we aim to explore the feasibility and accuracy of the second and the third methods with no or minor adjustments. The details of the in-context implementation and optimization of the proposed solution design for anomaly detection in multivariate time series are presented in Section 7.

7 Towards Reliable DL Solutions: Evaluation and Maintenance

As previously mentioned in Section 1, we aim to address a shortage of approaches for evaluating DL methods in the DevOps context where ground truth data on anomalous system's behavior is unknown. Hence, in this section, we present an example of cloud infrastructure for implementation and optimization of the DL model, shown in Figure 1. When investigating examples of cloud infrastructure, we were mainly inspired by Microsoft tools and services since the system under study is implemented and deployed using the same platform. A detailed explanation of each of the numbered parts in Figure 1 is given below.

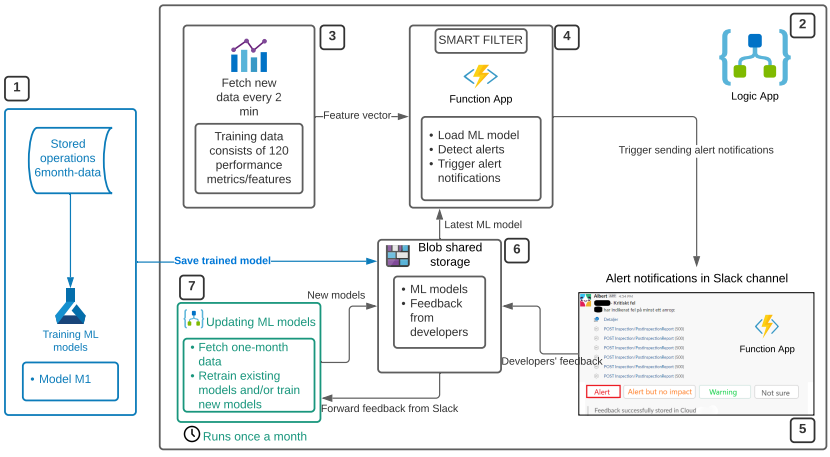


Figure 1: Overview of the proposed solution design for optimization of anomaly detection

1. This part relates to the training process of the selected model within Method 1, introduced in the previous section. Depending on the size of the training data set and the model complexity, the training process can be performed

either on the local machines or the cloud computing resources, such as Microsoft Azure ML Studio. The trained model will be saved into shared storage to be ready for deployment.

2. The overall solution is implemented using the Azure Logic Apps, a cloud-based platform for developing and running automated workflows. With this service, we aim to automate fetching of monitoring data in near-real-time, predicting anomalies, sending an alert notification to the Slack channel, and saving the feedback from the development team to a shared storage.

3. The automated workflow is triggered every two minutes since the fetching data may take up around one minute. The data is collected through sequential loops iterating over performance metrics and services, thus the data is received in the same order every time. The output of this action is a feature vector consisting of 120 performance metrics forwarded to the Function App.

4. The logic of the smart filter is implemented within the Function App, a cloud solution for running even the smallest code snippets in the cloud on-demand. The implementation includes loading the saved DL models and declaring the multidimensional feature vector as non-anomalous or anomalous based on the calculated anomaly score. In case that there is an observation that significantly deviates from a normal learned pattern, an alert notification with all needed details will be sent to the development team.

5. Sending notifications to the development team is implemented within another Function App and will be triggered if an anomaly has been discovered within the previous Function App. For each reported alert, the developer is required to provide feedback by selecting one of the available options: 1) alert; 2) alert but no impact 3) warning; 4) not sure. The feedback on the reported alerts will be continuously collected and saved in a shared storage.

6. This is a shared storage mainly used for storing the trained DL models and feedback from developers. We utilize benefits of the Azure Blob storage, a Microsoft's object storage solution that is optimized for storing unstructured data in the cloud.

7. This is a separate Logic App specifically used for the maintenance of the DL models. It uses the labeled data, generated through the feedback process, and performance one-month data to update the existing DL model since the Azure Monitoring platform keeps the metrics in the memory for up to one month.

The core idea of this cloud solution is to utilize feedback from the development team to evaluate and update the DL model based on generated labeled data denoting true positive alerts, true negative alerts, warnings, and unspecified alerts.

The first selected method, Method 1, will be evaluated after one month of usage in the production environment against the feedback from the development. Further, collected labeled data will be used for exploring and evaluating other DL methods, e.g. a Method 2 or Method 3 as suggested in the previous section. The method that gives the best accuracy will be selected as a new best choice and will be deployed and periodically updated based on the feedback that is continuously collected and saved. In every iteration, more and more labeled anomalous data will be collected and used for optimizing the unsupervised DL model and possibly learning new supervised models that will be optimized separately or in an ensemble with unsupervised DL models.

8 Conclusion and Future Work

In this paper, we address the importance of a DevOps concept, continuous monitoring, in revealing unexpected and unwanted failures in a microservices system. Thus, we present an overview of the unsupervised deep learning approaches for anomaly detection in order to identify potential discrepancies in multivariate monitoring data. Furthermore, we provide generic guidelines for the selection of the three minimum feasible methods for anomaly detection in time series based on the review in Section 5. Therefore, in Section 5 and Section 6 we answered on research questions **RQ1** and **RQ2**.

The last **RQ3** is addressed in Section 7 where we explore the cloud infrastructure that could be used for deploying and evaluating the DL model. Moreover, we investigate how to evaluate and keep the DL model updated by utilizing the aforementioned cloud infrastructure for the identification of false-negative alerts and obtaining labeled data. In this way, multivariate data keeping the information about the health of microservices can be processed in real-time and declared as anomalous or non-anomalous while the team of practitioners will be continuously evaluating the reported anomalous events and providing the feedback. The feedback will be afterward used for continuous maintenance and optimization of unsupervised or newly created supervised DL models.

The actual implementation and empirical evaluation of the proposed solution design for optimization of anomaly detection will be carried out in future work.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Subutai Ahmad, Alexander Lavin, Scott Purdy, and Zuha Agha. Unsupervised real-time anomaly detection for streaming data. *Neurocomputing*, 262:134–147, November 2017.

[2] Md Abul Bashar and Richi Nayak. TAnoGAN: Time Series Anomaly Detection with Generative Adversarial Networks. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1778–1785, Canberra, ACT, Australia, December 2020. IEEE.

[3] Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A. Lozano. A Review on Outlier/Anomaly Detection in Time Series Data. *ACM Computing Surveys*, 54(3):1–33, June 2021.

[4] Jeanderson Cândido, Maurício Finavaro Aniche, and Arie van Deursen. Contemporary software monitoring: A systematic literature review. *CoRR*, abs/1912.05878, 2019.

[5] Antonio Capizzi, Salvatore Distefano, Luiz J. P. Araújo, Manuel Mazzara, Muhammad Ahmad, and Evgeny Bobrov. Anomaly detection in DevOps Toolchain. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 37–51. Springer International Publishing, 2020.

[6] Antonio Capizzi, Salvatore Distefano, and Manuel Mazzara. From DevOps to DevDataOps: Data management in devops processes. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 52–62. Springer International Publishing, 2020.

[7] Antoine Chevrot, Alexandre Vernotte, and Bruno Legeard. DAE : Discriminatory Auto-Encoder for multivariate time-series anomaly detection in air transportation. *arXiv:2109.04247 [cs]*, September 2021.

[8] Kukjin Choi, Jihun Yi, Changhwa Park, and Sungroh Yoon. Deep Learning for Anomaly Detection in Time-Series Data: Review, Analysis, and Guidelines. *IEEE Access*, 9:120043–120065, 2021.

[9] Marcello Cinque, Raffaele Della Corte, and Antonio Pecchia. Advancing monitoring in microservices systems. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 122–123, Berlin, Germany, 2019. IEEE.

[10] Jürgen Cito, Philipp Leitner, Harald C. Gall, Aryan Dadashi, Anne Keller, and Andreas Roth. Runtime metric meets developer: Building better cloud applications using feedback. In *2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!) - Onward! 2015*, pages 14–27, Pittsburgh, PA, USA, 2015. ACM Press.

[11] Jürgen Cito, Johannes Wettinger, Lucy Ellen Lwakatare, Markus Borg, and Fei Li. Feedback from Operations to Software Development—A DevOps Perspective on Runtime Metrics and Logs. In Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, volume 11350, pages 184–195. Springer International Publishing, Cham, 2019.

[12] Carlos E. Cuesta, Elena Navarro, and Uwe Zdun. Synergies of system-of-systems and microservices architectures. In *Proceedings of the International Colloquium on Software-Intensive Systems-of-Systems at 10th European Conference on Software Architecture - SiSoS@ECSA '16*, pages 1–7, Copenhagen, Denmark, 2016. ACM Press.

[13] Yingnong Dang, Qingwei Lin, and Peng Huang. AIOps: Real-World Challenges and Research Innovations. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 4–5, Montreal, QC, Canada, May 2019. IEEE.

[14] Nan Ding, HaoXuan Ma, Huanbo Gao, YanHua Ma, and GuoZhen Tan. Real-time anomaly detection based on long short-term memory and gaussian mixture model. *Computers & Electrical Engineering*, 79:106458, 2019.

[15] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, Dallas Texas USA, October 2017. ACM.

[16] Tolga Ergen and Suleyman Serdar Kozat. Unsupervised Anomaly Detection With LSTM Neural Networks. *IEEE Transactions on Neural Networks and Learning Systems*, 31(8):3127–3141, August 2020.

[17] Michael Felderer, Barbara Russo, and Florian Auer. On testing data-intensive software systems. In Stefan Biffl, Matthias Eckhart, Arndt Lüder, and Edgar R. Weippl, editors, *Security and Quality in Cyber-Physical Systems Engineering, With Forewords by Robert M. Lee and Tom Gilb*, pages 129–148. Springer, 2019.

[18] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, January 2017.

[19] Jerome H. Friedman and Bogdan E. Popescu. Predictive learning via rule ensembles. *The Annals of Applied Statistics*, 2(3):916 – 954, 2008.

[20] Xiaoyu Fu, Rui Ren, Sally A. McKee, Jianfeng Zhan, and Ninghui Sun. Digging deeper into cluster system logs for failure prediction and root cause diagnosis. In *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 103–112, Madrid, Spain, September 2014. IEEE.

[21] Florian Gardin, Ronan Gautier, Nicolas Goix, Bibi Ndiaye, and Jean-Matthieu Schertzer. Machine learning with logical rules in Python. `https://github.com/scikit-learn-contrib/skope-rules`, 2020.

[22] Alexander Geiger, Dongyu Liu, Sarah Alnegheimish, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. TadGAN: Time Series Anomaly Detection Using Generative Adversarial Networks. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 33–43, Atlanta, GA, USA, December 2020. IEEE.

[23] Tanja Hagemann and Katerina Katsarou. A systematic review on anomaly detection for cloud computing environments. In *2020 3rd Artificial Intelligence and Cloud Computing Conference*, AICCC 2020, page 83–96, New York, NY, USA, 2020. Association for Computing Machinery.

[24] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. Identifying impactful service system problems via log analysis. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*, pages 60–70, Lake Buena Vista, FL, USA, 2018. ACM Press.

[25] Shilin He, Jieming Zhu, Pinjia He, and Michael R. Lyu. Experience Report: System Log Analysis for Anomaly Detection. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, pages 207–218, Ottawa, ON, Canada, October 2016. IEEE.

[26] Adha Hrusto, Per Runeson, and Emelie Engström. Closing the Feedback Loop in DevOps Through Autonomous Monitors in Operations. *SN Computer Science*, 2(6):447, August 2021.

[27] Ruei-Jie Hsieh, Jerry Chou, and Chih-Hsiang Ho. Unsupervised Online Anomaly Detection on Multivariate Sensing Time Series Data for Smart Manufacturing. In *2019 IEEE 12th SOCA Conference*, pages 90–97, Kaohsiung, Taiwan, November 2019. IEEE.

[28] Mohammad S. Islam, William Pourmajidi, Lei Zhang, John Steinbacher, Tony Erwin, and Andriy Miranskyy. Anomaly detection in a large-scale cloud platform. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 150–159, Madrid, ES, 2021. IEEE.

[29] Wenqian Jiang, Yang Hong, Beitong Zhou, Xin He, and Cheng Cheng. A gan-based anomaly detection approach for imbalanced industrial time series. *IEEE Access*, 7:143608–143619, 2019.

[30] Ying Jiang, Na Zhang, and Zheng Ren. Research on intelligent monitoring scheme for microservice application systems. In *2020 International Conference on Intelligent Transportation, Big Data Smart City (ICITBS)*, pages 791–794, Vientiane, Laos, 2020. IEEE.

[31] Yıldız Karadayı, Mehmet N. Aydin, and A. Selçuk Öğrenci. A Hybrid Deep Learning Framework for Unsupervised Anomaly Detection in Multivariate Spatio-Temporal Data. *Applied Sciences*, 10(15):5191, July 2020.

[32] Farzaneh Khoshnevisan, Zhewen Fan, and Vitor R. Carvalho. Improving robustness on seasonality-heavy multivariate time series anomaly detection. *CoRR*, abs/2007.14254, 2020.

[33] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S. Jensen. Outlier Detection for Time Series with Recurrent Autoencoder Ensembles. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 2725–2732, Macao, China, August 2019. International Joint Conferences on Artificial Intelligence Organization.

[34] Abhishek Kumar, Tristan Braud, Sasu Tarkoma, and Pan Hui. Trustworthy ai in the age of pervasive computing and big data. In *2020 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 1–6, Austin, TX, USA, 2020. IEEE.

[35] Eero Laukkanen, Juha Itkonen, and Casper Lassenius. Problems, causes and solutions when adopting continuous delivery—A systematic literature review. *Information and Software Technology*, 82:55–79, February 2017.

[36] Chang-Ki Lee, Yu-Jeong Cheon, and Wook-Yeon Hwang. Studies on the GAN-Based Anomaly Detection Methods for the Time Series Data. *IEEE Access*, 9:73201–73215, 2021.

[37] Dan Li, Dacheng Chen, Lei Shi, Baihong Jin, Jonathan Goh, and See-Kiong Ng. MAD-GAN: multivariate anomaly detection for time series data with generative adversarial networks. *CoRR*, abs/1901.04997, 2019.

[38] Zheng Li, Yue Zhao, Nicola Botta, Cezar Ionescu, and Xiyang Hu. Copod: Copula-based outlier detection. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 1118–1123. IEEE, 09 2020.

[39] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, Regunathan Radhakrishnan, and Joseph Fernandez. Unveiling clusters of events for alert and incident management in large-scale enterprise it. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '14*, pages 1630–1639, New York, New York, USA, 2014. ACM Press.

[40] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. Log clustering based problem identification for online service systems. In *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16*, pages 102–111, Austin, Texas, 2016. ACM Press.

[41] Ying Lin, Zhengzhang Chen, Cheng Cao, Lu-An Tang, Kai Zhang, Wei Cheng, and Zhichun Li. Collaborative Alert Ranking for Anomaly Detection. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 1987–1995, Torino Italy, October 2018. ACM.

[42] Jianwei Liu, Hongwei Zhu, Yongxia Liu, Haobo Wu, Yunsheng Lan, and Xinyu Zhang. Anomaly detection for time series using temporal convolutional networks and Gaussian mixture model. *Journal of Physics: Conference Series*, 1187(4):042111, April 2019.

[43] Matthieu Lucke, Moncef Chioua, Chriss Grimholt, Martin Hollender, and Nina F. Thornhill. Integration of alarm design in fault detection and diagnosis through alarm-range normalization. *Control Engineering Practice*, 98:104388, May 2020.

[44] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. Dimensions of devops. In Casper Lassenius, Torgeir Dingsøyr, and Maria Paasivaara, editors, *Agile Processes in Software Engineering and Extreme Programming*, pages 212–217, Cham, 2015. Springer International Publishing.

[45] Lucy Ellen Lwakatare, Pasi Kuvaja, and Markku Oivo. An exploratory study of devops: Extending the dimensions of devops with practices. In *The Eleventh International Conference on Software Engineering Advances (ICSEA)*, Rome, Italy, 08 2016.

[46] Sepehr Maleki, Sasan Maleki, and Nicholas R. Jennings. Unsupervised anomaly detection with LSTM autoencoders using statistical data-filtering. *Applied Soft Computing*, 108:107443, September 2021.

[47] Alok Mishra and Ziadoon Otaiwi. Devops and software quality: A systematic mapping. *Comput. Sci. Rev.*, 38:100308, 2020.

[48] Mohsin Munir, Shoaib Ahmed Siddiqui, Andreas Dengel, and Sheraz Ahmed. DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. *IEEE Access*, 7:1991–2005, 2019.

[49] Alina Oprea, Zhou Li, Ting-Fang Yen, Sang H. Chin, and Sumayah Alrwais. Detection of early-stage enterprise infection by mining large-scale log data. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 45–56, 2015.

[50] Alessandro Orso, Donglin Liang, Mary Jean Harrold, and Richard Lipton. Gamma system: Continuous evolution of software after deployment. *SIGSOFT Softw. Eng. Notes*, 27(4):65–69, July 2002.

[51] Roberto Pietrantuono, Antonia Bertolino, Guglielmo De Angelis, Breno Miranda, and Stefano Russo. Towards Continuous Software Reliability Testing in DevOps. In *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, pages 21–27, Montreal, QC, Canada, May 2019. IEEE.

[52] Roberto Pietrantuono, Antonia Bertolino, Guglielmo De Angelis, Breno Miranda, and Stefano Russo. Towards continuous software reliability testing in devops. In *AST*, pages 21–27. IEEE/ACM, May 2019.

[53] William Pourmajidi, John Steinbacher, Tony Erwin, and Andriy Miranskyy. On challenges of cloud monitoring. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, CASCON '17, page 259–265, USA, 2017. IBM Corp.

[54] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M. Verner, and Markku Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, 123:263–291, January 2017.

[55] Lukas Ruff, Jacob R. Kauffmann, Robert A. Vandermeulen, Grégoire Montavon, Wojciech Samek, Marius Kloft, Thomas G. Dietterich, and Klaus-Robert Müller. A Unifying Review of Deep and Shallow Anomaly Detection. *Proceedings of the IEEE*, 109(5):756–795, May 2021.

[56] Per Runeson, Emelie Engström, and Margaret-Anne Storey. The design science paradigm as a frame for empirical software engineering. In Michael Felderer and Guilherme Horta Travassos, editors, *Contemporary Empirical Methods in Software Engineering*, pages 127–147. Springer, 2020.

[57] Carla Sauvanaud, Mohamed Kaâniche, Karama Kanoun, Kahina Lazri, and Guthemberg Da Silva Silvestre. Anomaly detection and diagnosis for cloud services: Practical experiments and lessons learned. *Journal of Systems and Software*, 139:84–106, May 2018.

[58] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices. *IEEE Access*, 5:3909–3943, 2017.

[59] Lifeng Shen, Zhuocong Li, and James Kwok. Timeseries anomaly detection using temporal hierarchical one-class network. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13016–13026. Curran Associates, Inc., 2020.

[60] Daniel Ståhl, Torvald Mårtensson, and Jan Bosch. Continuous practices and devops: Beyond the buzz, what does it all mean? In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 440–448, Vienna, September 2017. IEEE.

[61] Sampo Suonsyrja, Laura Hokkanen, Henri Terho, Kari Systa, and Tommi Mikkonen. Post-Deployment Data: A Recipe for Satisfying Knowledge Needs in Software Development? In *2016 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement (IWSM-MENSURA)*, pages 139–147, Berlin, Germany, October 2016. IEEE.

[62] Sampo Suonsyrjä, Laura Hokkanen, Henri Terho, Kari Systä, and Tommi Mikkonen. Post-deployment data: A recipe for satisfying knowledge needs in software development? In *IWSM-MENSURA*, pages 139–147. IEEE, 2016.

[63] Stefan Throner, Heiko Hutter, Niklas Sanger, Michael Schneider, Simon Hanselmann, Patrick Petrovic, and Sebastian Abeck. An Advanced DevOps Environment for Microservice-based Applications. In *2021 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 134–143, Oxford, United Kingdom, August 2021. IEEE.

[64] Deepika Tiwari, Long Zhang, Martin Monperrus, and Benoit Baudry. Production monitoring to improve test suites. *IEEE Transactions on Reliability*, pages 1–17, 2021.

[65] Muhammad Waseem, Peng Liang, Mojtaba Shahin, Amleto Di Salle, and Gastón Márquez. Design, monitoring, and testing of microservices systems: The practitioners' perspective. *Journal of Systems and Software*, 182:111061, 2021.

[66] Xiwei Xu, Liming Zhu, Min Fu, Daniel Sun, An Binh Tran, Paul Rimba, Srini Dwarakanathan, and Len Bass. Crying wolf and meaning it: Reducing false alarms in monitoring of sporadic operations through pod-monitor. *2015 IEEE/ACM 1st International Workshop on Complex Faults & Failures in Large Software Systems (COUFLESS)*, pages 69 – 75, 2015.

[67] Ang Zhang, Xiaoyong Zhao, and Lei Wang. CNN and LSTM based Encoder-Decoder for Anomaly Detection in Multivariate Time Series. In *2021 IEEE 5th Information Technology,Networking,Electronic and Automation Control Conference (ITNEC)*, pages 571–575, Xi'an, China, October 2021. IEEE.

[68] Chuxu Zhang, Dongjin Song, Yuncong Chen, Xinyang Feng, Cristian Lumezanu, Wei Cheng, Jingchao Ni, Bo Zong, Haifeng Chen, and Nitesh V. Chawla. A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:1409–1416, July 2019.

[69] Yuxin Zhang, Yiqiang Chen, Jindong Wang, and Zhiwen Pan. Unsupervised Deep Anomaly Detection for Multi-Sensor Time-Series Signals. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.

[70] Hang Zhao, Yujing Wang, Juanyong Duan, Congrui Huang, Defu Cao, Yunhai Tong, Bixiong Xu, Jing Bai, Jie Tong, and Qi Zhang. Multivariate time-series anomaly detection via graph attention network. *CoRR*, abs/2009.02040, 2020.

[71] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. Understanding and handling alert storm for online service systems. In *2020 IEEE/ACM 42nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 162–171, 2020.

[72] Nengwen Zhao, Panshi Jin, Lixin Wang, Xiaoqin Yang, Rong Liu, Wenchi Zhang, Kaixin Sui, and Dan Pei. Automatically and Adaptively Identifying Severe Alerts for Online Service Systems. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2420–2429, Toronto, ON, Canada, July 2020. IEEE.

[73] Peihai Zhao, Xiaoyan Chang, and Mimi Wang. A Novel Multivariate Time-Series Anomaly Detection Approach Using an Unsupervised Deep Neural Network. *IEEE Access*, 9:109025–109041, 2021.

[74] Yue Zhao, Zain Nasrullah, and Zheng Li. Pyod: A Python toolbox for scalable outlier detection. *Journal of Machine Learning Research*, 20(96):1–7, 2019.

[75] Chong Zhou and Randy C. Paffenroth. Anomaly Detection with Robust Deep Autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674, Halifax NS Canada, August 2017. ACM.