Fully Automated DORA Metrics Measurement for Continuous Improvement

Janick Rüegger
janick.rueegger@gmail.com
University of Applied Sciences and Arts Northwestern Switzerland
Windisch, Switzerland

Martin Kropp
martin.kropp@fhnw.ch
University of Applied Sciences and Arts Northwestern Switzerland
Windisch, Switzerland

Sebastian Graf
sebastian.graf@fhnw.ch
University of Applied Sciences and Arts Northwestern Switzerland
Windisch, Switzerland

Craig Anslow
craig.anslow@ecs.vuw.ac.nz
Victoria University of Wellington
Wellington, New Zealand

ABSTRACT

Many organizations implementing DevOps are adopting DORA metrics to measure their capabilities. Surveys are commonly used to gather the DORA metric data. The data are typically captured only in longer intervals, and on the team level, omitting important details on individual software systems. In this paper, we present a solution that enables completely automated measurement and calculation of the DORA metrics from DevOps tooling on individual microservices level, and in real-time.We evaluated the developed solution in an industrial case study consisting of 37 microservices over a four-week period. The evaluation demonstrated the ability for a completely automated DORA metrics calculation which provides much greater transparency for data-driven decision-making and optimizations on a more fine-granular level allowing teams to continually improve their software processes and outputs.

CCS CONCEPTS

• **Software and its engineering → Software evolution**.

KEYWORDS

Software delivery, DORA metrics, DevOps practices, Agile methodologies, Continuous improvement, Microservices

ACM Reference Format:
Janick Rüegger, Martin Kropp, Sebastian Graf, and Craig Anslow. 2024. Fully Automated DORA Metrics Measurement for Continuous Improvement. In *International Conference on Software and Systems Processes (ICSSP '24), September 4–6, 2024, München, Germany*. ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3666015.3666020

1 INTRODUCTION

In today's fast-paced global markets, software is at the heart of change. As a result, excellence in software delivery becomes an essential capability to provide high-quality and innovative services to compete [17]. To accelerate their delivery, many organizations start to adopt agile methodologies and DevOps tooling [10]. However, they often lack a clear and consistent set of indicators and critical capabilities to assess their software delivery progress. Thus, many organizations face major challenges in their DevOps transformation when trying to adjust their priorities based on data-driven insights [4]. In 2014, Forsgren et al. published their industrial research on the DORA (DevOps Research and Assessment) metrics, which identified four empirical software delivery metrics that predict overall business performance [15]. The DORA metrics have since gained widespread industrial interest in the software industry and many organizations are increasingly adopting these metrics [41].

When starting with DORA metrics, most organizations rely on qualitative surveys. While standardized qualitative surveys offer a simple yet established method to assess the DORA metrics, they are based on subjective assessments and can only be collected occasionally. Typically, the performance of the main application is used to draw conclusions about the systems it comprises [31]. Surveys are also limited in their execution frequency, often to avoid survey fatigue of the participants. Hence, while serving as good starting point, surveys may lead to outdated, simplified and over-generalized assessments based on sparse data.

To the best of our knowledge, research on automated DORA metrics is still scarce, apart from a few industrial proofs of concept. Thus, we aim to provide empirical evidence on the practicality and potential of automated DORA metrics computation to support continuous software delivery performance improvements. In contrast to traditional surveys, by improving the precision, granularity, and scalability of the DORA metrics [18], the automated collection should enable us to gain insights into individual systems' performances and behaviors.Thus, we introduce an open-source system for microservice-based software systems to compute the DORA metrics from DevOps tooling data. The transparency introduced by this approach may enable data-driven decision-making tailored to the specific challenges of individual systems.

In case study, we conducted a comparative analysis on the performance of a team responsible for thirty-seven micro-services. For overall team-performance, we use the aggregate over all individual micro-services. The software delivery performance of individual systems is defined by the data collected from various DevOps systems. We use the term 'individual systems' through the paper when referring to the individual micro-services. Finally, we compared the resulting DORA metrics to identify potential similarities and differences between the micro-services. Thereby, the following research questions were addressed:

- **RQ1:** How do the DORA metrics on individual systems differ from the teams performance?
- **RQ2:** How do the DORA metrics differ between individual systems?
- **RQ3:** Which technical events are needed to calculate each DORA metric?
- **RQ4:** Which technical data sources can be used for capturing these events?

To collect the data and provide the DORA metrics, we developed a measurement system and published it as an open-source project [38]. In the design process, we identified the data sources and the software development lifecycle events, that provide the needed data to calculate the DORA metrics. Our case study shows that team performance has limited representational capabilities for individual microservices performance. The study reveals significant differences between the performance of individual microservices, and compared to the overall team performance. Our study shows the

ICSSP '24, September 4–6, 2024, München, Germany

ACM ISBN 979-8-4007-0991-3/24/09.
https://doi.org/10.1145/3666015.3666020

benefits of evaluating microservice performance in addition to the overall team performance, and emphasizes the significance of automated DORA metrics to gain continuous insights into software delivery performance. Our approach represents a promising solution to solve the challenges faced by organizations in assessing their software delivery progress and making data-driven decisions. The remainder of the paper is structured as follows. Section 2 describes related work about automated DORA metrics. Section 3 presents the case study where we applied our solution. Section 4 describes the solution approach. Section 5 presents the results. Section 6 discusses the results. Section 7 summarizes our conclusions.

2 RELATED WORK

Agile adoption and DevOps practices have gained popularity in recent times [5, 28, 29]. Andreessen [1] and Cockburn [6] have discussed the reasons for the acceleration in software development and the implications of agile adoptions. Highsmith emphasizes the importance of having access to continuous real-time metrics that also enables dynamic prioritization and adjustments on planning [26]. By shortening the feedback cycles using real-time insights [13, 32], teams are empowered to experiment with development practices [35, 40]. GitLab has released its own implementation of DORA metrics for its services, released several blog entries covering practical experiences [13, 32], and plans to expand in this area.

Ebert et al. [10] and Zhu et al. [46] introduce DevOps and its key characteristics. They use tools like CI/CD, or telemetry to measure some of the DORA metrics. Perera et al. provide empirical evidence of the positive impact of measuring DORA metrics [34]. The history and characteristics of micro-services, relevant for tracking DORA metrics, are introduced by Dragoni et al. [9]. Numerous papers provide insight into which software metrics can be used to measure these new paradigms. Fenton et al. [12] describe the history of traditional software development metrics, however that work does not address modern DevOps metrics. Kupianinen et al. [30] analyze commonly applied metrics and deduce characteristics of high-influence indicators for modern software systems. Forsgren et al. [16] discuss how DevOps performance, in particular, can be measured, and which quality criteria have to be met.

Lomio et al. [31] and Sallin et al. [39] addressed the usefulness of measuring the DORA metrics. Sallin et al. present a case study on automated DORA metric measurement using a working implementation. Harmel-Law authored work on software architecture metrics, sharing his experiences using DORA metrics in projects with Thoughtworks, sharing practical lessons, insights on the daily use of these metrics, and concrete calculation methods. [25]. He argues that automated DORA metrics have received a high degree of industrial interest and popularity [25]. Since the first edition in 2014, the DORA reports have announced better business outcomes for companies with a high DORA metrics performance [35].

Thoughtworks developed an open-source application on automated DORA metrics - namely the projects *dora-team/fourkeys* [42] and *thoughtworks/metrics* [44]. In 2019, they included the DORA metrics in the "Adopt" phase, highlighting their practicality [43]. In a recent study, Gebrewold and Wirell [19] analysed the challenges in measuring the DORA metrics, focusing on deployment frequency and lead time for change. They report challenges with reporting the data in a manner that reflects reality, issues in understanding the collected data, and doubts regarding the usefulness, value, and significance of DevOps metrics.

In this paper, we close the gap by introducing DORA metrics to gain insights about the delivery performance on individual systems and address some of its challenges.

3 CASE STUDY

The presented case-study in this paper is a collaboration with a Zurich-based software company. Their main business focus is on mass-internet-of-things (IoT). At the time of the case study, the company employed twelve developers, who worked in different technical fields such as backend, frontend, testing, and infrastructure. They are organized as a single cross-functional Scrum team. The team applies modern DevOps practices and state-of-the-art tooling. Their sophisticated DevOps tool-chain supports an event-based distributed micro-service architecture. All involved engineers have at least three years of experience in building and managing distributed systems.

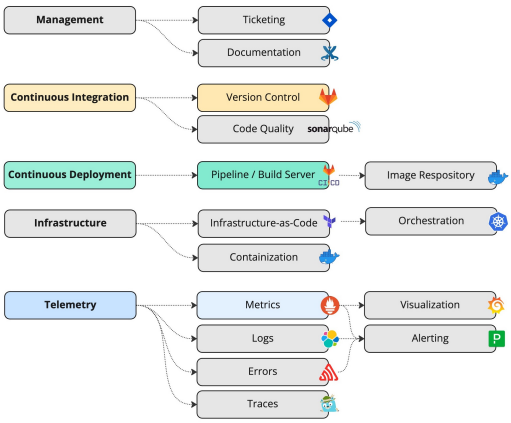

Figure 1: Initial case-study DevOps tooling infrastructure, leveraged to create automated DORA metrics.

Based on our inclusion criteria defined below in Section 4, thirty-seven systems have been selected for the study. Our partners were highly motivated to collaborate in this project as they have a strong need to scale the event-based architecture and are very interested in improving their software delivery performance. Their DevOps tooling setup shown in Figure 1 provides the data needed to calculate the DORA metrics. The relevant tools and aspects for the measurement of the DORA metrics and the development of our solution are listed in Table 1.

Table 1: List of tools used for DORA metrics measurement.

Task	Tool in use
Ticketing	Tickets and documentation with Jira
Version Control	Trunk-based development with Gitlab.
CICD	Continuous Deployment on Gitlab CICD.
Infrastructure	Container Orchestration with Kubernetes.
Monitoring	Metrics with Prometheus.

4 SOLUTION DESIGN

4.1 Background

Before we dive into the development or our new approach, we provide an analysis of the limitations of survey-based DORA metrics assessment as described by Forsgren et al. [14, 16].

4.1.1 The Four Key Metrics. The four key metrics to measure software delivery performance as defined by Forsgren et al. [15] are:

- *Deployment Frequency (DF)*
- *Lead Time for Changes (LTFC)*
- *Change Failure Rate (CFR)*
- *Mean Time to Restore (MTTR)*

While the first two indicators conceptualize velocity or tempo in software delivery, *CFR* and *MTTR* define aspects of stability [15]. A fifth metric, Reliability, was introduced in 2021 after the official definition of the DORA metrics. Reliability nowadays is based on product-defined Service Level Indicators (SLI) and objectives. Since handling these metrics is already

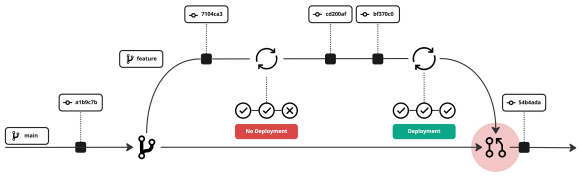

Figure 2: Illustration of DF. In the example, two pipeline runs are illustrated, where only one succeeds. Only a single deployment gets triggered (highlighted in red).

covered by modern monitoring systems, we focus on the original four metrics in this paper.

4.1.2 Limitations of Survey-Based Metrics. Software delivery performance is typically collected on the primary application [35]. In this case, the main system acts as a proxy for the aggregated performance [18]. Thus, survey-based assessments assume that software delivery performance is shared across all sub-systems in an equal manner, ignoring local context and challenges in individual systems.

Subjective responses. The objectivity of responses is questioned due to various factors: Surveys rely on subjective responses and human interpretation. While practitioners have little confidence in their own capabilities to assess their progress, organizations often overestimate their capabilities [4]. Political pressure to accelerate their digital transformation [34] is one of the lead causes [15].

Accuracy of responses. The surveys typically offer a 4-6 point Likert scale for responses with time ranges, or vague options, like "On Demand" for delivery frequency, which introduces vagueness.

Recall. Surveys are typically only conducted occasionally. A repeated inquiry may results in survey fatigue [15]. In addition, the responses are influenced by the recall capability of the participants. Extreme events or the latest experience are typically well remembered, but the response quality degrades over time.

Scalability. Due to the occasional execution of the surveys, the available data is coarse-grained. This may hide important details. Thus, surveys are often weakly conclusive in their scope of detail. Comparisons over longer periods may not be not appropriate [14].

Recurring Costs. Executing surveys requires the time of the users as well as of the survey analysts. This time is constant. To improve data quality, more surveys need to be executed, with the additional drawback of increased costs.

In summary, collecting DORA metrics via surveys does not provide teams with the necessary capabilities to capture local context and challenges, which we would expect in today's highly distributed and heterogeneous software systems. One promising approach to unveil these differences are automated DORA metrics computed from DevOps tooling data.

4.1.3 Tooling-Based Metrics. Automated DORA metrics based on DevOps tooling data promise to solve these issues [14]. This approach leverages data from DevOps tools, such as version control, CICD, or telemetry data[9, 10]. An inquiry based on system data introduces several benefits such as higher precision, high granularity, and scalability [15].

Granularity. Automatically measured system data are less prone to response biases, subjectivity, and inconsistencies, and generally achieve more reliable insights [25]. In addition, automated metrics provide new capabilities, such as the ability to assess software delivery performance for all subsystems individually or in aggregation. This enables detailed visibility and transparency into a team's software development process and systems. Even on individual systems, teams can detect specific issues and propose tailor-made solutions to reflect local requirements and constraints.

Feedback-cycles. By shortening the feedback cycles using real-time insights [13, 32], teams are empowered to experiment with development practices [35], [40]. Having access to continuous real-time metrics enables dynamic prioritization and adjustments on planning [26] allowing teams to take full responsibility based on production data. Joint decisions are made quickly and with confidence [6].

4.2 Metrics Definition & Implementation

The technical implementation of each metric is critical for the collection of the underlying data and the subsequent analysis of the software delivery performance. Thus, we define the quantitative representation of each metric, identify the required resources and define their concrete method of calculation. Thereby, we considered the requirements and constraints given by the case-study specific DevOps infrastructure.

4.2.1 Deployment Frequency (DF). The deployment frequency addresses the question of "[...] how often does your organization deploy code to production or release it to end users?" [35]. Thus, the technical definition for the deployment frequency DF is the total of all deployments(d) divided by the length of the time period(t): $DF = \frac{\sum d}{t}$

Design. According to the introduced definition, the deployment frequency acts as a proxy for a team's capability to deliver software in small batches to accelerate the feedback cycle time [26]. This not only applies to customer-value but also to the software delivery process itself [15]. This definition emphasizes the need for a deployment to successfully pass and ship code to a production environment. If these qualities do not both apply, a pipeline run is not considered a successful deployment [25].

Today, deployments can be done using a wide array of established DevOps tools, such as GitLab CICD [23], GitHub Actions [20], Jenkins [27], Bamboo [3], Argo CD [2], and others. As can be seen in Figure 1, our use-case partner uses GitLab CICD.

As we can get the data for deployments from GitLab's Pipeline API [22], the technical definition of a deployment is a successful pipeline run, automatically triggered by a push to the default branch and ending in a deployment to production. Manual executions, merging to other branches as well as failed runs are not considered. An example of a deployment according to our technical definition is shown in Figure 2.

4.2.2 Lead-Time for Changes (LTFC). The metric lead time for changes is defined by "[...] how long it take[s] to go from code committed to code successfully running in production" [35]. Given this narrow definition, this metric is probably the easiest to interpret [39]. Lead time for changes ($LTFC$) can be calculated as the total of all change lead times (LT) divided by the number of all changes (C) in a time period: $LTFC = \frac{LT}{\sum C}$

Design. This metric introduces the concept of change. Whereas a change is colloquially understood as a modification to source code, we are interested in its duration. In contrast to the other metrics, Forsgren et al.'s definition of lead time for changes introduces a strong technical implementation assumption [39]. According to their definition, changes are calculated based on commit and deployment data [15]. Our design adopts this definition and technical instructions. We calculate the duration of a change as the time between the deployment of a change and the first commit timestamp. To find the first commit of a change, some concepts inherent to Git need to be addressed. A change includes one or multiple commits. The merge-commit created after a successful merge has two parents – the merge-commit of the previous merge, and the last commit of a change. We are not interested in the last, but rather the first commit of a change. Using the knowledge about the parents of a merge-commit, however, the first commit of a change is identical to the first commit after the previous merge-commit. The use of commit meta-data prohibits the use of certain Git practices, as explained in detail in the limitations subsection. As the software development team of our case-study partner does not use any of the problematic practices, these

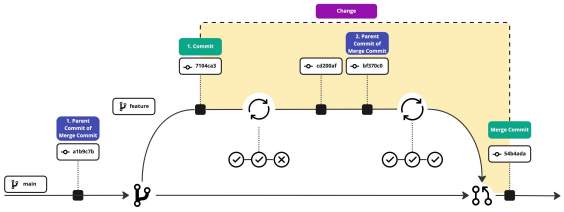

Figure 3: Illustration of *LTFC*. A change with three commits and one merge commit is illustrated as the time between the first commit and deployment (highlighted in yellow color).

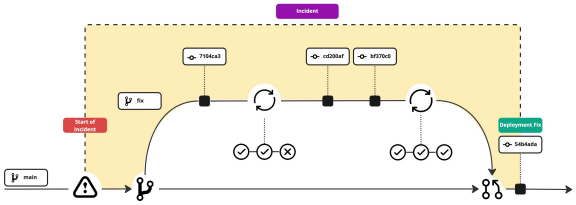

Figure 4: Illustration *MTTR*. In the example given, a single incident is illustrated as the time between the incident start and the deployment of a fix (highlighted in yellow color).

limitations did not restrict us. All data for commits and the deployment is again provided by the GitLab API [21, 22]. The introduced concepts are shown in Figure 3.

4.2.3 Mean Time to Restore (MTTR). Mean time to restore is defined by "[…] how long […] it generally takes to restore service when a service incident or defect occurs that impacts users". Furthermore, an incident is defined for example as "unplanned outage or service impairment" [35]. Given these definitions, mean time to restore (*MTTR*) is calculated as the total amount of incidents restore duration(t_i) divided by the number of all incidents(i) in a time period: $MTTR = \frac{\sum t_i}{\sum i}$

Design. For our purposes, incidents are defined in accordance with Peters et al. definition as outages or impairments, such that a user cannot access a service's functionality [35]. Smaller bugs are minor impairments are not included. To track incidents in an automated way, we rely on telemetry data in the proposed design. Specifically, we integrate the telemetry tool Prometheus [36], since the development team already implemented and uses several custom alerts.

An incident is defined as a defective state of a system with a start and an end time. Since we monitor incidents using Prometheus, incidents are defined by predefined PromQL alert queries. Although the alert definitions in PromQL have been shared for all systems in this case study, the proposed design would allow the definition of specific queries for each individual system. The concept of how to calculate the incident duration is shown in Figure 4.

4.2.4 Change Failure Rate (CFR). The change failure rate is defined by "[…] what percentage of changes to production or released to users result in degraded services […] and subsequently require remediation" [35]. Thus, this definition builds on the two concepts of deployments and incidents, which have been previously introduced. Given these definitions, the change failure rate (*CFR*) is calculated as the total number of incidents (i) divided by the total number of deployments (d): $CFR = \frac{\sum i}{\sum d}$

Design. We know from anecdotal evidence [25] that this metric is notoriously hard to calculate when trying to match incidents with specific deployments since it's almost impossible to find the exact source of failures. To work around this problem, we calculate the change failure rate as the percentage between the incidents and deployments that happened during a specific period. Thus, the change failure rate for a period might result in a value bigger than one, since technically multiple incidents could happen per single deployment. An example is shown in Figure 5.

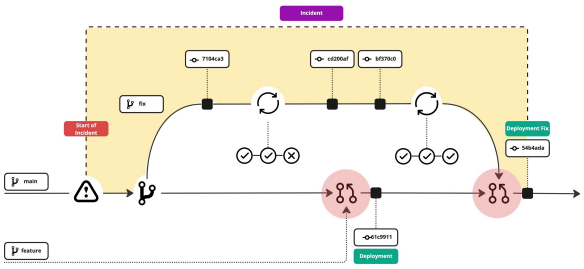

Figure 5: Illustration of *CFR*. In the example given, a single incident, as the time between the incident start and the deployment of a fix (highlighted in yellow color), and two separate deployments are illustrated (highlighted in red color).

4.3 Events & Data Sources

Building on the provided DevOps infrastructure presented in Section 3, and Section 4.2 on the individual metrics definition, research questions **RQ3** and **RQ4** can be answered:

> **RQ3**: *Which technical events are needed to calculate each DORA metric?*
> **RQ4**: *Which technical data sources can be used for capturing these events?*

In consistency with the concepts used by other open-source solutions [42, 44], the events asked for in research question *RQ3* include the types of deployments, commits, and telemetry alerts. Based on the raw data of these events, changes, and incidents can be calculated. Given the infrastructure introduced in Section 3, the respective data sources for the raw data are derived. Thus, we can also answer research question *RQ4* as follows. The Source-Code Management System (SCM) is used for commits. A CICD-system takes care about deployments' data. Platform telemetry is utilized for alerts. These findings are summarized in Table 2 and in Figure 6. Based on these concepts, the technical possibilities, and the limitations identified, the software solution for the backend system is defined.

Table 2: DORA metric's relationship to raw data and calculated events, and their data sources.

Metric	Raw Event	Calculated Event	Data Source
DF	Deployments	-	CICD (Gitlab CICD)
LTFC	Deployments, Commits	Changes	SCM / CICD (Gitlab)
MTTR	Alerts	Incidents	Platform Monitoring (Prometheus)
CFR	Deployments, Commits, Alerts	Changes, Incidents	CICD / CICD / Platform Monitoring (Gitlab, Prometheus)

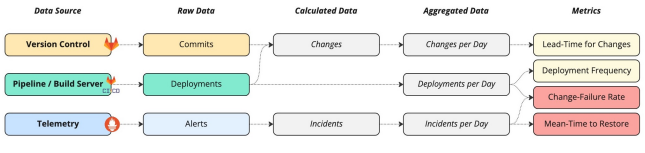

Figure 6: Data source's raw, calculated, and aggregated events, and the resulting DORA metrics (left to right).

4.4 Software Architecture

For the collection of raw data and the calculation of DORA metrics for the case study, a system was designed and implemented. The overall architecture with the basic functionalities of the system and a typical flow are shown in Figure 7.

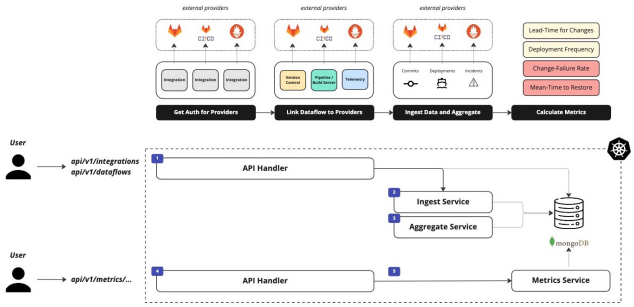

Figure 7: System architecture including its technical components (bottom) and their functionality (top).

The most important elements of the architecture are:

api: *Handler for API calls for raw data and metrics.*
connectors: *Clients for all provided integrations.*
daos: *Data access objects for all event types.*
database/mongodb: *MongoDB integration used by the DAOs.*
services: *Services to ingest, aggregate and calculate.*

The system was implemented in *go 1.19* and runs as an independent micro-service in a *Container* (e.g. with *Docker Compose*), which can be deployed to *Kubernetes* (e.g. as *Helm chart*). With the *Docker Compose* and *Helm* installation, a *MongoDB* instance is automatically started, responsible for the persistence layer. The project itself was built according to current standards and best practices of the *go* ecosystem [7, 8, 24] and is freely available as open-source software under the project name *unnmdnwb3/dora* [37].

4.5 Method

To answer the research questions **RQ1** and **RQ2**, we conducted the case study over a period of four weeks with our case study partner. This approach allowed us to research as close as possible to today's industrial challenges and to achieve maximum relevance [11].

Inclusion Criteria. Our case-study partner currently maintains over 100 repositories, including experimental and Infrastructure-as-Code repositories. To identify the relevant ones for our study, we defined the following inclusion criteria which the software systems must meet:

- *All systems must be deployed directly to production.*
- *All systems must contain custom business logic.*
- *All systems must use version control.*
- *All systems must use continuous deployment.*
- *All systems must use telemetry.*

In total, thirty-seven systems met the required qualities. For each of these systems an additional assessment of importance was made by the case-study partner. A binary value was assigned to each system, forming two subcategories of critical and non-critical systems. We ran the study from January 23, 2023 to February 19, 2023. For the calculation of the metrics of all thirty-seven systems, an observation period of seven days was used as a sliding window for the moving average.

We compared each individual system's DORA metrics to the aggregated DORA metrics calculated over all systems. As one team is responsible for all examined micro-services, we used the aggregate of the individual micro-services as a substitute for the overall team performance. We discuss the limitations of this approach in the discussion section. We used a moving average window of seven days for the analysis of the DORA metrics to smooth out short-term fluctuations, since the events for individual systems do not happen continuously on a daily base. The seven-day moving window will allow us to highlight longer-term trends in the analysis of the systems.

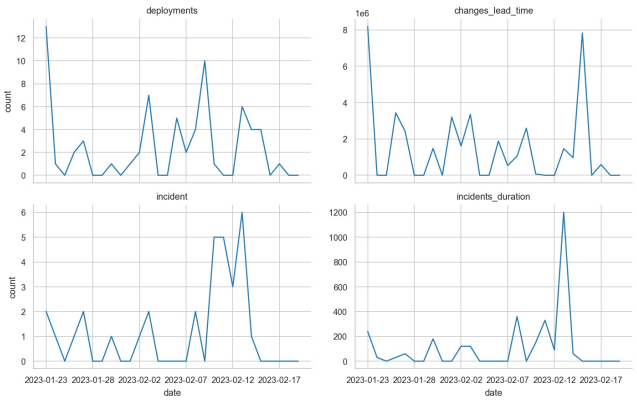

Figure 8: The raw data on the events during the measurement period (duration unit in seconds).

The representational potential of the aggregated DORA metrics (team performance) is determined using the Pearson correlation between the DORA metrics of individual systems and their aggregate. In addition, we quantify differences in both the raw data and the metrics. Finally, an analysis of the two groups by system importance was carried out. To ensure that our results were "testable, and empirically valid" [11], we published the source-code to collect the data. The code for the software and the evaluation are published in the repositories *unnmdnwb3/dora* [37]. The evaluation as Jupyter Notebooks and the complete dataset on events and metrics on *dora-evaluation* [38] can be made available on request after consultation with our case-study partner.

5 RESULTS

We present the results of our study and start with the presentation of the events happening during the four weeks followed by the importance and correlation analysis. The following example demonstrates that the continuously derived DORA metrics provide insights into the DevOps life cycle.

5.1 Events

The aggregated raw data across all thirty-seven systems for the four events, deployments, change lead-time, incidents, and incident duration over the measurement period are shown in Figure 8. In this overall view, it can be seen that all four event types are not evenly distributed. Apart from flattening out during the weekends, however, no additional obvious patterns are observable.

In the time frame discussed, a total of 67 changes were deployed with a total of 423 commits. A single change consists on average of 6.3 commits. The average time for the creation of a change is about 168 hours (7 days). The exact context of this high number was not determined in detail but might have several reasons; the changes might have stayed for a long time in a development branch before being deployed to production, changes might have been unresolved over the weekend, or changes had internal dependencies which needed to be deployed first. During the study 32 incidents occurred which resulted in a total downtime of 49.5 minutes.

Table 3 shows the detailed data for the top 10 systems for the deployment and change lead time events. In total 78.4% (29) of all projects were deployed at least once to production during the study duration. The top 10 projects account for 62.8% of all deployments. Apart from three outliers accounting for 31.3%, the deployments are quite evenly distributed across all projects. However, with 27% (10), the biggest subset of projects by total deployments has only seen a single deployment during the study duration. Only 8.1% (3) of projects got deployed more than a single time on a day.

For the change lead time, the top 10 accounts for 69.2% of the total change lead time. One of the two outliers accounting for 20.0% of the total change lead time is not in the top 10 of deployments, while the second one being 7th place. The mean of all projects with at least a single change is 13.9 hours. Only 16.2% (6) of all projects have a mean change lead time of under 8 hours or a working day.

Table 3: The top 10 total numbers of "deployments" (left) and the spread of all of "deployments" (right, unit in seconds).

Number of Deployments						
dataflow_name	sum	mean	Std	min	max	pct
auth-service	8	0.286	0.810	0	3	0.119
query-service	8	0.286	0.659	0	3	0.119
rule-runner-service	5	0.179	0.390	0	1	0.075
storage-service	3	0.107	0.567	0	3	0.045
asset-service	3	0.107	0.315	0	1	0.045
log-processor-service	3	0.107	0.315	0	1	0.045
output-fanout-service	3	0.107	0.315	0	1	0.045
permissions-service	3	0.107	0.315	0	1	0.045
script-runner	2	0.071	0.262	0	1	0.030
flow-service	2	0.071	0.262	0	1	0.030

Change Lead Time [s]						
system	sum	mean	Std	min	max	pct
rule-scheduler	4388125.0	156718.8	829277.7	0	4388125	0.108
output-fanout-service	3635922.0	129854.4	607568.3	0	3203625	0.089
api-gateway	2658035.0	94929.8	502321.4	0	2658035	0.065
script-runner	2513209.0	89757.5	473300.5	0	2504769	0.062
aggregation-service	2420670.0	86452.5	457463.6	0	2420670	0.059
rule-service	2419880.0	86424.3	455633.4	0	2411288	0.059
query-service	2333927.0	83354.5	241885.5	0	940660	0.057
log-processor-service	2106381.0	75227.9	251277.0	0	1185643	0.052
websocket-gateway	2004957.0	71605.6	378901.3	0	2004957	0.049
rule-runner-service	1920959.0	68605.7	212445.8	0	933617	0.047

The raw data on incidents and duration are in Table 4. Only 24.0% (9) of all projects have had at least a single incident. One major outlier accounted for 72.0% (23) of all incidents, while only having been deployed 3 times during the same period. This project has also seen the most incidents on a single day (5). When comparing grand totals, we observe 2.1 deployments per incident. When removing the "log-processor-service" from the incident statistic, an incident on average only occurred every 7.4 deployments.

The total incident duration across all systems amounts to 49.5 minutes. On average, an incident lasted 93 seconds before a new running version could take over. Also for the incident duration, the project "log-processor-service" was a major outlier, accounting for 64.6 percent of downtime.

Table 4: The top 10 total numbers of "incidents" (left) and the spread of all "duration" (right, unit in seconds).

Number of Incidents						
system	sum	mean	Std	min	max	pct
log-processor-service	23	0.821	1.634	0	5	0.719
rule-runner-service	2	0.071	0.262	0	1	0.062
output-fanout-service	1	0.036	0.189	0	1	0.031
user-service	1	0.036	0.189	0	1	0.031
trend-indicator-service	1	0.036	0.189	0	1	0.031
script-runner	1	0.036	0.189	0	1	0.031
output-coordinator	1	0.036	0.189	0	1	0.031
flow-coordinator	1	0.036	0.189	0	1	0.031
downlink-coordinator	1	0.036	0.189	0	1	0.031
mqtt-connector	0	0	0	0	0	0

Duration [s]						
system	sum	mean	Std	min	max	pct
log-processor-service	1920	68.571	221.187	0	1140	0.646
script-runner	210	7.500	39.686	0	210	0.071
output-coordinator	180	6.429	34.017	0	180	0.061
trend-indicator-service	150	5.357	28.347	0	150	0.051
flow-coordinator	150	5.357	28.347	0	150	0.051
downlink-coordinator	120	4.286	22.678	0	120	0.040
user-service	90	3.214	17.008	0	90	0.030
rule-runner-service	90	3.214	12.488	0	60	0.030
output-fanout-service	60	2.143	11.339	0	60	0.020
mqtt-connector	0	0.000	0.000	0	0	0.000

5.2 DORA Metrics

We show the aggregated DORA metrics for the 7-day moving average of all systems. Both the speed and stability metrics tend to increase over time of the study. The cause of this is difficult to discern based on the raw data. Effects such as the New Year period or a stronger focus on stability are conceivable. The metrics also fluctuate during their recording. The velocity metrics experience a temporary 4.7 times increase in deployment frequency and a 2.5 times increase in lead time. At the same time, stability metrics also increase 9.5 times in change-failure rate and 9.1 times increase in mean-time to restore. Subsequently, we discuss the results for each metric separately. For this purpose, for both values of moving-average windows, we list the top 10 systems by metric mean and visualize the spread of all systems.

The per-system measurement enables us a much more detailed analysis of the DORA metrics, than the overall or team aggregate. Tables 5 & 6 show the DORA metrics over the whole period with a 7-day moving average for the top 10 systems ($micro-service$). The different systems show significant differences.

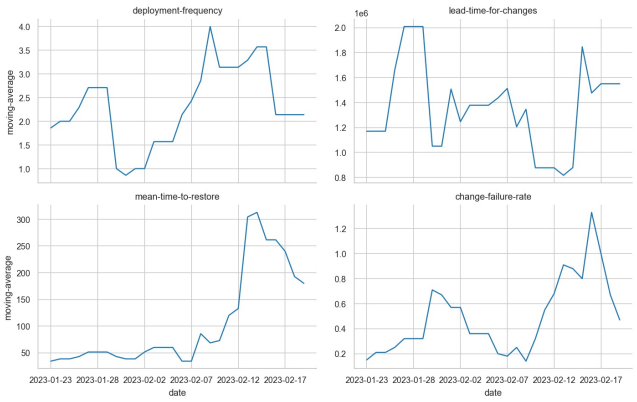

Figure 9: Aggregated DORA metrics over all systems with a 7-day moving average (duration in seconds).

Since the deployment frequency is based on the deployment events, all systems that experienced any deployments have a positive mean. However, looking at the details, we see in Table 5 that the deployment frequency for the system with the biggest mean ("auth-service") is about 4 times the lowest of the top 10 systems. The analysis of all thirty-seven systems shows even bigger differences between the systems: the biggest positive mean is 11.4 times bigger than the smallest positive mean, and 3.6 times bigger than the average of all positive means. The much higher deployment frequency might be an indicator of an issue with this system.

Since the lead time for changes is based on commits and deployment events, all systems that experienced any deployments also have a positive mean. However, here again, we see significant differences between the systems. The system with the highest lead time for changes is about 5 times higher than the lowest system in the top ten, and even 2445 times bigger than the system with the smallest positive mean of all systems.

Table 5: The top 10 means for the "deployment frequency" (left) and "lead time for change" (right, unit in seconds).

Mean DF				
system	mean	Std	min	max
auth-service	0.285	0.230	0.0	0.57
query-service	0.270	0.268	0.0	0.86
rule-runner-service	0.168	0.105	0.0	0.43
storage-service	0.108	0.190	0.0	0.43
asset-service	0.106	0.093	0.0	0.29
permissions-service	0.105	0.073	0.0	0.29
output-fanout-service	0.095	0.078	0.0	0.29
log-processor-service	0.086	0.098	0.0	0.29
user-service	0.070	0.082	0.0	0.29
mqtt-connector	0.070	0.071	0.0	0.14

Mean LTFC				
system	mean	Std	min	max
output-fanout-service	125579	196523	0.00	457661
rule-scheduler	111942	244494	0.00	626875
api-gateway	94930	167440	0.00	379719
aggregation-service	86453	152488	0.00	345810
rule-service	86424	151717	0.00	344470
query-service	83257	76294	0.00	218570
websocket-gateway	71606	126300	0.00	286422
script-runner	64199	139531	0.00	359030
auth-service	63501	53948	0.00	138296
rule-runner-service	63321	62654	0.00	200022

Analysing the change failure rate and the mean time to restore metrics on per-system base, again shows significant differences between the systems for both metrics. The system with the largest change failure rate has a CFR value 18.0 times bigger than the smallest positive mean of all thirty-seven systems. For the mean time to restore metric the system with the highest $MTTR$ value is 32.0 times bigger than the smallest positive mean.

Further analysis shows us that the same system "log-processor-service" accounts for the high metric values, and this system is also in the top ten for the deployment frequency. The incident analysis in Table 4 highlights that the same system accounts for most incidents. This might be an indicator that there is an issue with this system, to be further looked at. This would not have been possible with the team-based aggregated DORA metric.

Table 6: The top 10 means for the "change-failure rate" (left) and "mean-time to restore" (right, unit in seconds).

Mean CFR				
system	mean	Std	min	max
log-processor-service	3.857	6.197	0	18
rule-runner-service	0.387	0.458	0	1
trend-indicator-service	0.250	0.441	0	1
output-coordinator	0.250	0.441	0	1
flow-coordinator	0.250	0.441	0	1
downlink-coordinator	0.250	0.441	0	1
output-fanout-service	0.232	0.419	0	1
user-service	0.232	0.419	0	1
script-runner	0.214	0.418	0	1
mqtt-connector	0.000	0.000	0	0

Mean MTTR				
system	mean	Std	min	max
log-processor-service	68.572	92.399	0.0	244.290
script-runner	7.500	13.229	0.0	30.000
output-coordinator	6.428	11.337	0.0	25.710
trend-indicator-service	5.358	9.450	0.0	21.430
flow-coordinator	5.358	9.450	0.0	21.430
downlink-coordinator	4.285	7.558	0.0	17.140
user-service	3.215	5.671	0.0	12.860
rule-runner-service	2.909	3.510	0.0	8.570
output-fanout-service	2.143	3.779	0.0	8.570
mqtt-connector	0.000	0.000	0.0	0.000

5.3 System Importance

At the beginning of the study, our case-study partner provided a binary classification of the systems into critical and noncritical groups based on perceived importance. We were interested if those systems differ in the DORA metrics, and evaluated the DORA metrics against these two categories. The resulting means (we averaged the means for all critical, and all non-critical systems) and their standard deviations are shown in Table 7.

For the velocity metrics, we see that the deployment frequency for critical systems is 2.6 times higher than for uncritical systems and 1.6 times higher than the mean over all systems. The lead time for changes in critical systems is 1.7 times higher than in uncritical systems and 1.4 times higher than the average.

For the stability metrics, the contrary applies — all metrics covering incidents are lower for critical systems in comparison to uncritical ones. The change failure rate in critical systems is 31.1% smaller than the observed values for uncritical systems, and also 30.0% smaller than the overall average. We see similar results for the mean time to restore. Critical systems took 42.1% less time to restore in comparison to uncritical ones, and 31.2% less time when comparing against the mean over all systems.

These results thus show clear differences between systems according to their perceived importance. The reason is discussed later in the following section. Finally, the correlation of the DORA metrics of individual systems with their overall aggregate is examined.

Table 7: Comparison of the resulting DORA metrics based on a binary criticality classification (duration in seconds).

Importance			
metric	importance	mean-of-means	mean-of-stds
deployment-frequency	1	0.100	0.096
deployment-frequency	0	0.038	0.052
lead-time-for-changes	1	49841.833	72201.462
lead-time-for-changes	0	28644.910	50721.679
change-failure-rate	1	0.112	0.185
change-failure-rate	0	0.189	0.308
mean-time-to-restore	1	1.968	3.355
mean-time-to-restore	0	3.401	4.757

5.4 Correlations

We analyzed the correlation of each DORA metric of individual systems with their aggregate to find out, how well the aggregate, which we use as a proxy for the team-based DORA metrics, represents the average system. We use the Pearson correlation for this analysis, as we can assume that (1) the relationship between the variables is linear, (2) both variables are quantitative.

Tables 8 & 9 show the top 10 correlations of the four DORA metrics for the systems. They give a diverse picture of the correlation of the different systems with their aggregate. While some systems correlate highly and significantly with the aggregate, others do not, or even have a significant negative correlation. "NaN" values in the table indicate that the correlation could not be calculated due to missing data. Looking at the already discussed "log-processor-service", this correlates moderately to strongly with the aggregate for all four metrics. On the other side, the "trend-indicator-service" has a moderate to strong negative correlation for three of the four metrics. Other systems indicate low or no correlation, however, some also, with a p-value above the typical threshold of 0.05, so do not give clear picture. Overall, the diversity of the correlations of the individual systems with the aggregate is large.

Table 8: The top 10 correlations for the "deployment frequency" (left) and "lead-time for change" (right, duration in seconds).

Correlations DF		
system	coev	pval
rule-runner-service	0.897	0.00000
websocket-gateway	0.754	0.00000
tracking-service	0.754	0.00000
flow-coordinator	0.698	0.00004
billing-service	0.698	0.00004
trend-indicator-service	0.640	0.00024
rule-service	0.640	0.00024
script-runner	0.629	0.00034
query-service	0.629	0.00034
mqtt-connector	0.591	0.00093

Correlations LTFC		
system	coev	pval
rule-scheduler	0.422	0.025
coap-connector	0.403	0.033
log-processor-service	0.365	0.056
api-gateway	0.363	0.057
integration-service-new	0.347	0.071
dashboard-service	0.316	0.101
datapoint-metrics-processor	0.312	0.106
aggregation-service	0.312	0.106
company-ui	0.274	0.159
output-fanout-service	0.144	0.464

Table 9: The top 10 correlations for the "change-failure rate" (left) and "mean-time to restore" (right, duration in seconds).

Correlations CFR		
system	coev	pval
output-fanout-service	0.661	0.0001
log-processor-service	0.377	0.0478
output-coordinator	0.215	0.2720
script-runner	0.157	0.4261
rule-runner-service	0.111	0.5747
flow-coordinator	0.077	0.6978
downlink-coordinator	-0.271	0.1626
user-service	-0.43	0.0225
trend-indicator-service	-0.507	0.0059
aggregation-service	NaN	NaN

Correlations MTTR		
system	coev	pval
output-fanout-service	0.754	0.00000
log-processor-service	0.666	0.00011
rule-runner-service	0.537	0.00321
flow-coordinator	0.497	0.00710
script-runner	0.497	0.00710
user-service	-0.169	0.38950
downlink-coordinator	-0.226	0.24850
output-coordinator	-0.261	0.17900
trend-indicator-service	-0.467	0.01234
aggregation-service	NaN	NaN

Comparison between Metrics. For the comparison of the correlation, we classified the correlation into the categories of positive strong, positive moderate, negative moderate, and negative strong. A positive strong correlation is defined as higher than 0.4, a positive moderate from 0.0 to less than 0.4, and the negative categories accordingly. For this comparison, we counted all systems with a significant p-value $p \leq 0.05$ from all thirty-seven systems.

The overall statistics on the correlation for all four DORA metrics and the moving-average windows of 7-day are shown in Table 10. For the correlation of the "deployment frequency", 29.0% of the systems (9 systems) have a significant strong positive correlation with the overall aggregate. Another 16.1% (5) of the systems have a moderate positive correlation. The correlation of the "lead time for changes" metric, only 6.4% (1) of systems have at least a significant moderate positive correlation with the

Table 10: Correlations of the DORA metrics for individual systems with the aggregate over all systems. Categorized by percentage in positive strong, moderate, and negative strong, moderate correlations.

	Percentage			
Metric	Pos. Strong	Pos. Mod.	Neg. Mod.	Neg. Strong
DF	29.0	16.1	3.2	0.0
LTFC	0.0	6.4	6.4	6.4
CFR	11.1	0.0	0.0	22.2
MTTR	22.2	33.3	11.1	0.0

overall aggregate. In addition, 13.0% (2) have at least a negative moderate correlation. For the "change failure rate" only 11.1% (1) of systems have at least a moderate correlation with the aggregate metric. Another 22.2% (2) have a strong negative correlation. For the "mean time to restore" metric, 22.2% (2) of systems have a strong positive correlation. In addition, 33.3% (3) have a positive and 11.1 (1) have a negative medium correlation. Calculating the mean correlation over all four metrics with a 7-day moving average, only 26.8% of the systems have a strong correlation with the aggregate. These results show the operational setup of the product can neither be gained by static analysis of traceability measures nor with the help of manual examination.

6 DISCUSSION

We structure the discussion according to the research questions into an analysis of the correlation between the systems' DORA metrics and their aggregate (team performance), as well as a quantification of any differences between the DORA metrics of individual systems. Based on both analyses, we finally present general insights into the collection and interpretation of automated DORA metrics. We also discuss the limitations of our work and their results.

6.1 Insights on Aggregates

For RQ1, we calculated the DORA metrics for each system. We analyzed the correlations between the individual DORA metrics and their aggregate (team performance).

RQ1: How do automated DORA metrics on individual systems differ from the teams performance?

Team performance does not correlate to individual system performance. The results of our analysis clearly show that individual systems differ significantly from their aggregate. We see clear differences between individual systems. The range is wide. Some projects have a very strong correlation of 0.88, while others with a value of 0.0 are not correlated at all, and yet others have even a negative correlation to the team DORA metric. The big differences in correlations between the DORA metrics of individual systems and the aggregated indicators result in a large standard deviation. This is also reflected in the very high standard deviation of between 0.98 to 3.96 times the mean.

Conclusion. While aggregated DORA metrics still adequately represent overall team performance, the sum of all DORA metrics in our data is neither indicative nor representative of actual performance in individual systems. Thus, any differences between individual systems could be masked by the exclusive use of aggregate DORA metrics. The high differences in the DORA metrics helped us to identify outliers of the systems, which would need further investigations, making **RQ2** all the more relevant. To our knowledge, the correlation of DORA metrics of individual systems with their aggregate has not been explored before. The majority of research projects used surveys as the main method for data collection. Thus, they implicitly share the assumption that a team's main application reflects the performance of all other systems. In this respect, our results contradict the general assumption in the academic literature that the performance in the main application is representative for all maintained systems [14, 15, 18].

Insights on Systems. To answer RQ2 we evaluated both the differences in the raw data, as well as in the resulting DORA metrics between systems.

RQ2: How do automated DORA metrics differ between individual systems?

Systems are not equal. The results of our analysis show that the events are not distributed equally between systems. 27.0% of all systems account for 62.8% of all deployments and 69.2% of the total lead time for changes. At the same time, a single project is responsible for 71.9% of all outages. The large differences can often be explained with outliers. For example, some outliers in the lead time have a total duration 50.8 days. This also explains the high standard deviation of 1.04 times the average. However, there are also large differences in the DORA metrics of individual systems. The highest measured value of a single system is between 2.7 and 5.9 times the average value of all systems. When comparing the highest with the lowest values, the difference even increases from 11.4 to 2445 times. Again, the standard deviations were very high with 0.99 to 3.95 times the mean. With an extension of the observation period, these tend to increase only minimally.

Perceived importance matters. We categorized all systems based on our case-study partner's assessment of perceived importance, resulting in two groups based on a binary definition of criticality. Comparing the two groups revealed clear differences. Compared to unimportant systems, the important category experienced roughly three times as many changes on average, had a change-failure rate over 40% lower, and a mean-time to restore only half as long. A change for critical systems also took twice as long which could indicate a higher level of caution when dealing with critical systems. The analysis indicates that large differences exist in the handling of systems with different perceived importance.

A holistic view is essential. When comparing metrics and raw data, it is important to always do so with the same scope [25]. Differences in team size or observed period make comparison difficult. It is important not to focus exclusively on one metric, but to always consider the overall context of its' metadata, especially in relation to other DORA metrics. If this step is disregarded, the entire framework no longer works, since, for example, velocity may be improved at the expense of stability [25].

Conclusion. Individual systems of a team project differ significantly in the raw data, the resulting metrics, and even based on perceived importance. Although in the time-based metrics outliers have an influence on the strong spread, similar tendencies can also be identified in the deployment frequency and the change failure rate. To our knowledge, no previous scientific research has addressed the differences of DORA metrics between individual systems, which makes our insights novel in this area.

6.2 Insights on System Design

We introduced the solution design of our measurement system and the calculation methods (§4). However, the decisions made are strongly based on the existing use case. Apart from the chosen design, there are still some aspects that are relevant for increased usability and generalizability, which need to be discussed.

RQ3: Which events are needed to calculate each DORA metric?
RQ4: Which data sources can be used for capturing these events?

The automation's increase in complexity and overhead is only justified if the resulting metrics provide additional insights on individual systems. Otherwise, simpler approaches such as surveys across teams might provide a better cost-benefit ratio. We emphasize that this case study has shown that a) automated measurement of DORA metrics is feasible, b) provides new insights into individual system performances, and c) offers capabilities relevant to continuous improvements not only in theory but also in practice.

Integration into the existing DevOps tooling. Wiedemann et al. explain that "DevOps requires a custom solution for each organization" and that "each context is unique, and a prescriptive approach to DevOps implementation and adoption is unlikely to be successful" [45]. Teams should continue to use their preferred choice of tools in a best-of-breed manner [4], as this capability is one of the leading causes for the practical adoption of metrics [30] and job satisfaction [18]. In addition, it significantly lowers the entry barrier by removing the need to deploy dedicated and complex instrumentalization [33]. Thus, the automated measurement systems must be adapted to the current tool-set of an organisations or even individual teams. Currently, we only support a subset of integrations, which could be of good use for teams [39]. Possible additions, such as Kubernetes-native deployment technologies could provide a even wider array of meaningful data.

Don't miss out on manual inputs. A criticism for the automated collection of DORA metrics is their use of system data and limited applicability. System data only capture events within their own system boundaries and humans can detect and report phenomena, which lay outside those boundaries [15]. Our results show defining and identifying incidents is hard. While manual inputs may introduce issues with the correctness of the data based on biases or workplace politics [25], they could be integrated as an additional source to extend the existing automated capabilities. Automated and manual approaches should be combined into a hybrid solution [14]. For example, to track incidents, manual issue tracking systems like Atlassian Jira could be integrated. This would for a more diverse definition of incidents. In addition, data on corporate or team culture could be collected and discussed in teams [14].

Help teams improve. As Sallin et al. [39] have noted, even if all relevant data is captured, and teams are allowed to approach challenges in whatever they want [6], the adaption of DORA metrics could still fail because teams do not know which steps to take next in their efforts to improve. Aspects like the alignment of enterprise strategy with action, a lightweight management framework, creativity and the ability to provide rapid feedback are sustainable competitive advantages [15]. Therefore, it should be obvious that the DORA metrics alone do not suffice to create a true learning organization or a world-class delivery performance. They are only a means to an end and should be treated as such. A key distinction of high-performing organizations is their focus on modern software development practices, collaboration, and learning. These organizations are also more likely to have clear goals and metrics for continuous improvements [18, 35]. Thus, we argue to support teams in adopting these practices as well.

Scalability and Continuous data. The automated measurement system was used with thirty-seven software systems. It scales up to any number of systems with very little effort. Once set up, the systems can be run automatically eliminating execution costs to a large degree, especially compared to manual surveys. This enables a continuous data set which allows for timeline analysis on the individual systems and their aggregates.

Conclusion. The target audience for the DORA metrics are software development teams. These metrics and their underlying data need to be available and easily accessible for the engineers who actually make the changes. They must be empowered to assess their performance and decide on ways to improve [25]. Instead of using a top-down approach, discussing the DORA research results for such insights for a team can help to achieve more modular, testable, and observable systems over time [25]. A set of clear and common metrics is further critical to introduce baseline transparency as the basis for further inspection and adaption [40]. In contrast to survey-based DORA metrics, automated DORA metrics enable real-time insights on software delivery to individual systems, as well as aggregated team performance. Based on production data, they allow us to track the real-time progress of each indicator, capture trends over time [14], and provide critical insights on system context [35]. These baseline capabilities can support teams to continuously improve and provide achieve customer value with high quality and stability [30]. Having such detailed insights as demonstrated, could further support a wide array of shared activities such as planning or defining and tracking team objectives [30].

6.3 Limitations

Size of case study. Our research methods introduce limitations concerning the generalizability of the results and findings [16]. While case studies are very suitable to achieve insights into specific phenomena in industrial settings, the raw data and the resulting metrics are very much related to the context in which the data was generated. All data originate exclusively from a single team. Temporal, technological, and cultural aspects can have a major impact on the generalizability of the insights from this specific case. For example, it must be acknowledged that the case-study partner is a scaling start-up, which determines its processes, goals, and constraints. Further case studies are needed to compare DORA metrics between systems and different types of teams and organizations.

Aggregate as Team Performance. We used the aggregate of the individual micro-services as a proxy for the team performance. This has several limitations. While this situation might be acceptable for the given case, in other companies multiple teams might work on a shared set of micro-services. Also a survey-based determination of the DORA metrics raises different questions and, thus, captures different data which limits the comparability of the aggregate to the survey-based approach.

Limited Software Development Lifecycle. DORA metrics focus on software delivery from commit to telemetry on the running system [15]. This excludes all steps prior, such as the design phase or the infrastructure provisioning [39]. It is important to keep in mind that not all aspects and periods of a typical software development life cycle are covered by these indicators.

Data Quality. In addition, the data quality must also be considered to assess the significance of our results [16]. The completeness of the raw data is an important aspect. As for the raw data and metrics on stability, we must assume the potential for optimization. The design of our system currently only allows one single definition of an incident per system. A single definition of what constitutes an incident probably oversimplifies the complexity of errors in production systems. It also does not consider that systems are built and maintained, having different goals, limitations, and planning horizons. This raises the question of whether comparisons need to be done in sub-categories. At the same time, however, cross-system assessments could also be used to identify commonalities and categories across systems to better scale solution approaches. However, not only the completeness but also the validity of the data must be critically questioned. It has already been stated, for example, that the metadata of commits does not necessarily have to be meaningful for calculating the change lead time. Some alternative workarounds have already been discussed, but they often bring their own shortcomings. In this respect, it is important to be aware of these limitations or even to introduce better approaches.

6.4 Future Work

This case study has shown that measuring the DORA metrics on the system level brings new insights about software development teams which may help for a more focused continuous improvement. However, it also raises some new questions and requires more research. Further data analysis must be done to completely understand the different system results and the correlations. Longer longitudinal and cross-sectional studies over multiple teams and organizations to gain larger and more robust data sets might help to address some of the questions raised in the discussion. The developed OSS DORA measurement system might be extended for other infrastructures and more incident types for more generalizability.

7 CONCLUSION

In this work, we explored automating DORA metrics measurement using DevOps system data, examining how individual systems deviate from the aggregated norm. We determined the correlation and quantified differences between individual systems' DORA metrics and their aggregate. An open-source solution was developed for data collection and metrics calculation. In an industrial case study, data from thirty-seven systems over four weeks were evaluated, revealing significant performance differences between systems and groups categorized by perceived importance. These differences stem from diverse challenges, goals, and constraints. Our analysis suggests traditional survey-based DORA metrics may oversimplify and miss critical local insights. The automated collection method scaled to thirty-seven systems effortlessly, introducing real-time insights and additional indicators. Our results highlight the benefits of automated DORA metrics, encouraging further research to enhance software delivery performance.

REFERENCES

[1] Marc Andreessen. 2011. Why Software Is Eating The World. *Wall Street Journal* (2011), 1–5. http://online.wsj.com/article/SB10001424053111903480904576512250915629460.html

[2] ArgoCD. 203. Argo CD. https://argo-cd.readthedocs.io/en/stable/ https://argo-cd.readthedocs.io/en/stable/ (retrieved: 28.12.2023).

[3] Atlassian. 2023. Bamboo. https://www.atlassian.com/software/bamboo (retrieved: 28.12.2022).

[4] Atlassian & CITE. 2020. 2020 DevOps Trends Survey. https://www.atlassian.com/whitepapers/devops-survey-2020

[5] Robert Biddle, Martin Kropp, Andreas Meier, and Craig Anslow. 2021. Agile Software Development: Practices, Self-Organization, and Satisfaction. *The Agile Imperative: Teams, Organizations and Society under Reconstruction?* (2021), 39–54.

[6] Alistair Cockburn and Jim Highsmith. 2001. Agile Software Development: The People Factor. *Computer* 34, 11 (2001), 131–133.

[7] Go Community. 2023. Effective Go. https://go.dev/doc/effective_go (retrieved: 27.2.2023).

[8] Go Community. 2023. golang-standards/project-layout. https://github.com/golang-standards/project-layout (retrieved: 27.2.2023).

[9] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: Yesterday, Today, and Tomorrow. *Present and Ulterior Software Engineering* (2017), 195–216. https://doi.org/10.1007/978-3-319-67425-4_12 arXiv:1606.04036

[10] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. 2016. DevOps. *IEEE Software* 33, 3 (2016), 94 – 100.

[11] Kathleen M. Eisenhardt. 1989. Building Theories from Case Study Research. *Academy of Management Review* 14, 4 (1989), 532–550. https://doi.org/10.5465/amr.1989.4308385

[12] Norman E. Fenton and Martin Neil. 1999. Software metrics: successes, failures and new directions. *Journal of Systems and Software* 47, 2 (1999), 149–157. https://doi.org/10.1016/S0164-1212(99)00035-7

[13] Gustaw Fit. 2022. How Zoopla used DORA Metrics to Boost Deployments, Increase Automation and More. https://about.gitlab.com/blog/2022/01/24/how-zoopla-uses-dora-metrics-and-your-team-can-too/ https://about.gitlab.com/blog/2022/01/24/how-zoopla-uses-dora-metrics-and-your-team-can-too/ (retrieved 12.12.2023).

[14] Nicole Forsgren, Monica Chiarini Tremblay, Debra Vander Meer, and Jez Humble. 2017. DORA Platform: DevOps Assessment and Benchmarking. *International Conference on Design Science Research in Information Systems* May (2017). https://doi.org/10.1007/978-3-319-59144-5

[15] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. *Accelerate: The Science of Lean Software and DevOps - Building and Scaling High Performing Technology Organizations.* IT Revolution.

[16] Nicole Forsgren and Mik Kersten. 2018. DevOps Metrics. *Commun. ACM* 61, 4 (2018), 44 – 48. https://doi.org/10.1145/3178368.3182626

[17] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Thomas Zimmermann, Brian Houck, and Jenna Butler. 2021. The SPACE of Developer Productivity: There's more to it than you think. *Queue* 19, 1 (2021), 20–48.

[18] Nicole Forsgren Velasquez, Gene Kim, Nigel Kersten, and Jez Humble. 2014. 2014 State of DevOps Report. *Puppetlabs* (2014). http://puppetlabs.com/2014-devops-report

[19] Yamo Gebrewold and Johanna Wirell. 2023. Challenges with Measuring Software Delivery Performance Metrics: A case study at a software organisation. https://urn.kb.se/resolve?urn=urn:nbn:se:lnu:diva-121933

[20] Github. 2023. Github Actions. https://docs.github.com/en/actions https://docs.github.com/en/actions (retrieved: 28.12.2023).

[21] Gitlab. 2023. Gitlab API - Commits. https://docs.gitlab.com/ee/api/commits.html https://docs.gitlab.com/ee/api/commits.html (retrieved: 5.1.2023).

[22] Gitlab. 2023. Gitlab API - Pipelines. https://docs.gitlab.com/ee/api/pipelines.html https://docs.gitlab.com/ee/api/pipelines.html (retrieved: 26.2.2023).

[23] Gitlab. 2023. Gitlab CICD. https://docs.gitlab.com/ee/ci/ (retrieved: 28.12.2023).

[24] Google. 2023. Google Go Style. https://google.github.io/styleguide/go/ (retrieved: 27.2.2023).

[25] Andrew Harmel-Law. 2022. Four Key Metrics Unleashed. In *Software Architecture Metrics : Case Studies to Improve the Quality of Your Architecture*, Christian Ciceri, Dave Farley, and Neal Ford (Eds.). O'Reilly, 15 – 42.

[26] Jim Highsmith and Alistair Cockburn. 2021. Development: The Business of Innovation. *Computer* 34, 9 (2021), 120–127.

[27] Jenkins. 2023. Jenkins. https://www.jenkins.io/ https://www.jenkins.io/ (retrieved: 28.12.2023).

[28] Brendan Julian, James Noble, and Craig Anslow. 2019. Agile Practices in Practice: Towards a Theory of Agile Adoption and Process Evolution. In *Agile Processes in Software Engineering and Extreme Programming*, Philippe Kruchten, Steven Fraser, and François Coallier (Eds.). Springer International Publishing, Cham, 3–18.

[29] Marco Kuhrmann, Paolo Tell, Regina Hebig, Jil Klünder, Jürgen Münch, Oliver Linssen, Dietmar Pfahl, Michael Felderer, Christian R. Prause, Stephen G. MacDonell, Joyce Nakatumba-Nabende, David Raffo, Sarah Beecham, Eray Tüzün, Gustavo López, Nicolas Paez, Diego Fontdevila, Sherlock A. Licorish, Steffen Küpper, Günther Ruhe, Eric Knauss, Özden Özcan-Top, Paul Clarke, Fergal McCaffery, Marcela Genero, Aurora Vizcaino, Mario Piattini, Marcos Kalinowski, Tayana Conte, Rafael Prikladnicki, Stephan Krusche, Ahmet Coşkunçay, Ezequiel Scott, Fabio Calefato, Svetlana Pimonova, Rolf-Helge Pfeiffer, Ulrik Pagh Schultz, Rogardt Heldal, Masud Fazal-Baqaie, Craig Anslow, Maleknaz Nayebi, Kurt Schneider, Stefan Sauer, Dietmar Winkler, Stefan Biffl, Maria Cecilia Bastarrica, and Ita Richardson. 2022. What Makes Agile Software Development Agile? *IEEE Transactions on Software Engineering* 48, 9 (2022), 3523–3539. https://doi.org/10.1109/TSE.2021.3099532

[30] E. Kupiainen, M. V. Maentylae, and J. Itkonen. 2015. Using Metrics in Agile and Lean Software Development - A Systematic Literature Review of Industrial Studies. *Information and Software Technology* 62 (2015), 143–163.

[31] Francesco Lomio, Zadia Codabux, Dale Birtch, Dale Hopkins, and Davide Taibi. 2022. On the Benefits of the Accelerate Metrics: An Industrial Survey at Vendasta. *Proceedings - 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2022), 46–50. https://doi.org/10.1109/SANER53432.2022.00017

[32] Aathira Nair. 2022. How the DORA Metrics can help DevOps Team Performance. https://about.gitlab.com/blog/2022/04/20/how-the-dora-metrics-can-help-devops-team-performance/ https://about.gitlab.com/blog/2022/04/20/how-the-dora-metrics-can-help-devops-team-performance/ (retrieved: 12.12.2023).

[33] Pelorus. 2023. konveyor/pelorus. https://github.com/konveyor/pelorus (retrieved: 26.2.2023).

[34] Pulasthi Perera, Roshali Silva, and Indika Perera. 2017. Improve Software Quality through Practicing DevOps. *Proceedings of the 17th International Conference on Advances in ICT for Emerging Regions (ICTer)* (2017). https://doi.org/10.1109/ICTER.2017.8257807

[35] Claire Peters, Dave Farley, Daniella Villalba, Dave Stanke, Derek DeBellis, Eric Maxwell, John Speed Meyers, Kaiyuan Xu, Nathen Harvey, and Todd Kulesza. 2022. 2022 Accelerate: State of DevOps Report. https://cloud.google.com/devops/state-of-devops

[36] Prometheus. 2023. Prometheus. https://prometheus.io/ https://prometheus.io/ (retrieved: 16.1.2023).

[37] Janick Rüegger. 2023. unnmdnwb3/dora. https://github.com/unnmdnwb3/dora (retrieved: 27.2.2023).

[38] Janick Rüegger. 2023. unnmdnwb3/dora-evaluation. https://github.com/unnmdnwb3/dora-evaluation (retrieved: 27.2.2023).

[39] Marc Sallin, Martin Kropp, Craig Anslow, James W. Quilty, and Andreas Meier. 2021. Measuring Software Delivery Performance Using the Four Key Metrics of DevOps. *XP 2021: Agile Processes in Software Engineering and Extreme Programming* 419 (2021), 103 – 122.

[40] Ken Schwaber and Jeff Sutherland. 2020. The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game. https://doi.org/10.1002/9781119203278.app2

[41] Dustin Smith, Daniella Villalba, Michelle Irvine, Dave Stanke, and Nathen Harvey. 2021. Accelerate: State of DevOps 2021. https://services.google.com/fh/files/misc/state-of-devops-2021.pdf?hl=pt-br

[42] DORA Team. 2023. dora-team/fourkeys. https://github.com/dora-team/fourkeys (retrieved: 26.2.2023).

[43] Thoughtworks. 2022. Thoughtworks Techradar: March 2022. https://www.thoughtworks.com/radar/techniques/four-key-metrics (retrieved: 24.2.2024).

[44] Thoughtworks. 2023. thoughtworks/metrik. https://github.com/thoughtworks/metrik (retrieved: 26.2.2023).

[45] Anna Wiedemann, Nicole Forsgren, Manuel Wiesche, Heiko Gewald, and Helmut Krcmar. 2019. Research for practice: The Devops phenomenon. *Commun. ACM* 62, 8 (2019), 44–49. https://doi.org/10.1145/3331138

[46] Liming Zhu, Len Bass, and George Champlin-Scharff. 2016. DevOps and Its Practices. *IEEE Software* 33, 3 (2016), 32–34. https://doi.org/10.1109/MS.2016.81