Bachelor Degree Project

Challenges with Measuring Software Delivery Performance Metrics

- A case study at a software organisation

Author: Yamo Gebrewold
Author: Johanna Wirell
Supervisor: Tobias Olsson
Semester: VT 2023
Subject: Computer Science

Linnæus University
Sweden

Abstract

Online software businesses constantly face new challenges. Businesses are competing to deliver high-quality software solutions to their end users as quickly as possible. The performance of a software team in terms of software delivery needs to be measured to identify bottlenecks and understand what can be improved [1].
This project is a case study of a software organisation that delivers online web solutions for a business. The organisation strives to follow a DevOps mindset and work data-driven by collecting data, learning from it and acting on the learnings. Our method to perform the case study was to study existing data collected by the organisation and conduct interviews with software professionals to get their insights about the software delivery performance of three software teams within the organisation.
We focus on two metrics - deployment frequency and lead time for change - which measure the throughput of a software team and how often and fast it can deploy changes to production.
Software organisations that adopt a DevOps approach are facing challenges with collecting data about their software delivery performance and learning from that data to improve their throughput. We aim to identify these challenges and discuss various problems that software organisations need to be aware of when measuring software delivery performance.
We present the results from the interviews and collected metrics, analyse them and discuss them, as well as give suggestions on future research.
We observe that there are multiple factors impacting software delivery performance and various challenges with measuring it accurately. This is related to the way of reporting data reliably and developing models to study and understand the collected data.

Keywords: software delivery performance, DORA metrics, deployment frequency, lead time for change, DevOps speed metrics

Preface

We want to thank our supervisor Tobias Olsson from Linnaeus University who assisted us throughout the project and gave insightful ideas and suggestions, and Daniel Toll from Linnaeus University for aiding us during the process of working with the thesis. We also want to thank the software organisation that allowed us to perform the case study and use their data and resources to gain insights into their processes and ways of working. We enjoyed the process of the case study and learned a lot about software engineering, DevOps and DORA metrics.

Contents

1. Introduction **4**
1.1. Background 6
1.2. Related work 6
1.3. Problem formulation 7
1.3.1. Research questions 7
1.4. Motivation 8
1.5. Results 8
1.6. Scope/Limitation 8
1.7. Target group 9
1.8. Outline 9
2. Method **10**
2.1. Research project 10
2.2. Research methods 10
2.2.1. Pre-study 11
2.2.2. Interviews 11
2.2.3. Data collection 11
2.2.4. Data collection of DORA metrics 12
2.2.5. Analysis of the DORA metrics 13
2.2.6. Analysis of the interview data 13
2.3. Reliability and Validity 13
2.3.1. Reliability 13
2.3.2. Validity 14
2.3.3. Selection of methodology 14
2.3.4. Sampling strategy and size justification 14
2.3.5 Triangulation 15
2.4. Ethical considerations 15
3. Theoretical background **16**
3.3. Definition of terms 16
3.3.1. Continuous integration, continuous delivery and continuous deployment 16
3.3.2. DevOps 16
3.3.3. DORA metrics 16
3.3.4. Inter-team dependencies 19
3.3.5. Deployment 20
3.4. Organisation structures and processes 20
3.4.1. Structures 20
3.4.2. Processes 21
4. Research project – Implementation **23**
4.1. Hypotheses 23
4.2. Data collection of DORA metrics 23

4.3. Interviews 24
4.3.1. Conducting interviews with software engineers 24
4.3.2. Collecting interview data conducted by the DevOps team 25
4.4. Data processing 25
5. Results **26**
5.1. Deployment frequency metrics 26
5.2. Lead time for change metrics 27
5.3. Interviews conducted by us 28
5.3.1. Interview with Team 1 28
5.3.2. Interview with Team 2 29
5.3.3. Interview with Team 3 30
5.4. Interviews conducted by the dedicated DevOps team 31
5.4.1. DORA usage statistics 31
5.4.2. Software release process statistics 32
5.4.3. How DORA metrics are used in software teams 32
5.4.4. Goals and suggestions regarding software release processes 33
6. Analysis **35**
6.1. Deployment frequency average 35
6.2. Lead time for change average (days) 36
6.3. RQ1 Is there a significant difference in the DevOps speed metrics between teams? 37
6.4 RQ2 What are some of the causes for fluctuations in DevOps speed metrics? 38
6.5 RQ3 What are some of the challenges with measuring DevOps metrics and learning from the data? 40
7. Discussion **41**
8. Conclusions and future work **44**
9. References **46**
A Appendix **47**
A.1 Interview questions 47

1. Introduction

This is a 15 HEC Bachelor's thesis in computer science. The thesis investigates the topic of the software engineering cycle and more importantly the process of delivering software. It is based on a case study at a software organisation which develops solutions for an online business. The main focus is on measuring software delivery performance and more specifically the speed aspect of delivering software.

The case study aims to investigate the challenges that software teams experience with measuring software delivery performance. It explores concepts such as deployment frequency and lead time for change metrics. It also explores the release processes of various software teams.

Throughout the case study, data is collected from interviews with software practitioners as well as other relevant metric data from the product teams and compared within teams to gain new insights about successful strategies and potentially recurring problems.

Our study will focus on the two first metrics which are deployment frequency and lead time for change. Both of these metrics measure the throughput and speed of a software team in terms of software delivery [2]. This is a main priority of this study since quick software deliveries are a key business goal in all professional software organisations, and it is therefore interesting to learn how deployment frequency and lead time for change can differ between software teams.

1.1. Background

The need for delivering software quickly is ever-growing in today's industry. Businesses all over the world compete to have an online presence and sell their products through software solutions. This implies that the software must be delivered fast, and the software must meet the expectations of the customer.

A modern practice in software delivery is continuous integration and continuous deployment (CI/CD) which can be described as a methodology that emphasises the use of automation to enhance software delivery at every stage of the software development process. CI/CD improves the speed of delivering software by automating the deployment process. [3]

The goal is to deploy software as often as possible, even daily, and ship the latest changes to the end users which would increase customer satisfaction. This will inevitably come with the risk of a drop in the quality of the software. It, therefore, becomes crucial that the processes of testing changes in regression are well-defined and proven to fulfil the needs of the business.

The DORA (DevOps Research and Assessment) organisation has researched the topic of software delivery performance and identified four key measurable metrics known as the four key DevOps metrics or the DORA metrics. These metrics are deployment frequency, lead time for change, time to restore service and change failure rate. The first two metrics deal with the speed of software delivery, and the two other metrics deal with the quality of software delivery.

The thesis will examine the challenges of measuring software delivery performance. The case study will explore the factors that affect the deployment frequency as well as the lead time from committing changes to the deployment to production. To gather data for our research, we will conduct interviews and collect metrics from product teams at a software organisation which is part of an online business. Through this research, we hope to provide insights into the best practices and strategies for improving the software delivery process. Ultimately, this thesis aims to contribute to the advancement of software engineering practices and support the development of high-quality software solutions in a fast-paced and competitive industry.

1.2. Related work

Since our thesis project revolves around the topic of software delivery performance, the most relevant papers in this field are the "State of DevOps" reports from recent years and more importantly the most recent one [2]. This report is relevant as it discusses the concept of DORA metrics[1] and relates that to recent industry data from software companies which can be an interesting reference while performing the case study.

One of the most important and relevant papers written about the DevOps metrics is the paper called "Measuring software delivery performance using the four key metrics of DevOps" by M. Sallin et. al [1] which discusses the practice of automating the metric

[1] We use the terms *DORA metrics* and *DevOps metrics* interchangeably. We don't distinguish between them.

measurement as well as the value of measuring DevOps metrics. The authors of this paper also mention that there is no scientific research made on the topic of DevOps metrics and measuring software delivery performance, which made them resort to grey literature. Additionally, the work of Forsgren called "Accelerate: The Science of Lean Software and DevOps" [4] is of relevance and introduces the theory of the four key DevOps metrics.

As Sallin et al. mentioned there is much grey literature written on the topic of DevOps metrics and measuring software delivery performance [1]. A relevant grey literature is the "DevOps Trends" survey conducted by Atlassian, where participants shared their insights and experiences regarding DevOps practices. Participants voiced their opinions on the benefits of improved collaboration, faster software delivery, and increased efficiency that DevOps can bring to an organisation. However, they also highlighted challenges such as skill gaps, managing legacy infrastructures, and the need for cultural adjustments within the organisation to fully embrace DevOps principles. [5]

Furthermore, the "State of Agile Report" by State of Agile is relevant to our work as it provides valuable insights into the adoption of agile methodologies, which can greatly influence and contribute to the understanding of factors affecting lead time and deployment frequency in software development processes. [6]

Another relevant grey literature is the "Technology Radar" by ThoughtWorks as it focuses specifically on the key metrics related to software delivery and DevOps practices. This report offers significant perspectives and recommendations for measuring lead time and deployment frequency. [7]

1.3. Problem formulation

This thesis investigates a research gap regarding the factors that impact software delivery performance metrics and more importantly deployment frequency and lead time for change. While previous studies have focused on the usefulness and value of measuring DevOps metrics as well as discussing the topic of automating the metric measurement [1], this study shifts the focus to the factors that affect the DevOps metrics and also expands the discussion on the value of DevOps metrics. This study also discusses the challenge of measuring the DevOps metrics accurately.

At the time of writing, we did not find any concrete case studies that investigated how specific software teams measure software delivery performance. The studies that we found focused on conducting surveys with a broader population of software engineers about the overall usefulness of DevOps metrics, whereas our study consists of interviews with practitioners that have adopted the DevOps metric measurement.

1.3.1. Research questions

Our study aims at investigating the challenges of measuring and improving software delivery performance. Consequently, we formulate the following research questions:

- RQ1 Is there a significant difference in the DevOps speed metrics[2] between teams?
- RQ2 What are some of the causes for fluctuations in DevOps speed metrics?
- RQ3 What are some of the challenges with measuring DevOps metrics and learning from the data?

1.4. Motivation

From a societal point of view, improved software delivery performance can ultimately save software engineers time and effort. In e-commerce, for instance, retailers will be able to test and deploy new features more quickly, providing customers with a more dynamic and satisfying shopping experience. This can lead to increased customer satisfaction, which in turn can boost the economy and improve the overall quality of life in the community.

Another key motivation is that the time during which the software is not yet deployed is the time during which the software does not generate money. If we hypothetically presume that a new feature will generate a 1% higher cart value per successful purchase, and this new feature is postponed for several months due to factors causing a low software delivery performance, then the business will lose out on potential income during that period.

Furthermore, research on issues related to software delivery can bring additional economic benefits by identifying and addressing bottlenecks in the deployment and testing process, streamlining operations and reducing costs associated with delays and rework.

1.5. Results

We expect that the results will provide insights into the best practices for maintaining high software delivery performance and the role of CI/CD in improving the development process. Another area of investigation is the release processes that have proven to be successful and the most effective strategies for deploying software in terms of time to market. By addressing these research questions, our study will provide practical recommendations and pitfalls for software teams to improve the software development process.

1.6. Scope/Limitation

This project will be limited to investigating the state of a single software organisation and its software teams. The project is also mainly focused on the context of web frontend engineering which may not make it as applicable to other types of software engineering such as mobile application development since the release processes may differ and the way that end users get software updates is also fundamentally different.

The project's main concern is the release processes of the engineering teams which are to a big extent manifested in the two metrics of deployment frequency and lead time

[2] DevOps speed metrics here is a reference to deployment frequency and lead time for change. DevOps quality metrics deal with the time to restore service and change failure rate metrics.

for change. The project will not look into other metrics such as change failure rate or time to restore service.

The project is focused on the topic of software delivery. It will not discuss other parts of the software development lifecycle such as designing, implementing or testing software.

1.7. Target group

The target group for this information is primarily software engineers who are currently adopting continuous deployment and delivery practices and seeking ways to optimise their processes.

Moreover, other members of production teams such as project managers, product owners, and DevOps specialists may also find this information useful. These individuals play a crucial role in ensuring the successful development and deployment of software products and are often responsible for making decisions relating to the adoption of new technologies and development practices. Therefore, they may be interested in understanding the strategies and best practices used by product teams to improve the lead time from code commit to release in production.

1.8. Outline

We have organised the report in the following way. In Chapter 2, we discuss the method that we have selected for conducting the case study as well as the topic of reliability, validity and ethical considerations. Chapter 3 contains the theoretical background of the problem area and discusses terminology related to the context of the case study, as well as describes the structure and processes of the studied software organisation. In Chapter 4, we describe the steps that we followed to execute the method of our case study. Chapter 5 presents the results that we collected from executing our method, such as the interview answers and the reports. Chapter 6 and 7 analyse and discusses the collected results from the case study. In Chapter 8, we summarise the findings of our case study and give suggestions for future work.

2. Method

2.1. Research project

This project was initiated at a software organisation which develops web solutions for an online business. This organisation which consists of multiple product teams has gathered data about software delivery performance and other data about the software engineering processes. Some teams have measured for a smaller period of less than a year, and some have not measured this at all.

We planned to investigate the strategies applied by the software product teams in terms of releasing to production to learn about best practices for deploying continuously and delivering software at a fast pace. We wanted to identify the recipes for success at these teams and address the potential technical challenges that they may have faced.

We aimed to perform a collection and analysis of the existing data to gain a better understanding of what teams are doing well and what can be improved in terms of release processes.

We also intended to interview software practitioners to gain their insights on the interpretation of the existing data and get a better view of what the data can indicate regarding the state of development workflows and increase the validity of the results.

2.2. Research methods

We plan to collect and analyse data about two of the four common software delivery performance metrics (DORA metrics) that the product teams have gathered over time, namely deployment frequency and lead time for change. We believe that this data is useful for understanding which teams perform well and which do not in terms of software delivery and that it will help us learn more about proven strategies and technical challenges. We decided to not gather data about other metrics such as failure rate and time to restore service due to lack of data as well as time constraints.

Finally, we aim to conduct interviews with software practitioners and look at the data together with them to gain a better understanding of how the data should be interpreted. We chose to not conduct surveys at the cost of less generalisability due to lack of time. We also decided to shed more insights on the topic by using data from previous interviews with product teams conducted by the DevOps team about the overall DORA metrics usage and software release processes of the teams.

To ensure the credibility and validity of our findings, we chose to adhere to the case study guidelines proposed by Runeson and Höst [8], which are widely used in software engineering research. In addition, we decided to leverage established methods utilised in previous case studies such as Debbiche et al. [9] study on similar topics and Ionzon's and Jägstrand's [10] general software development case study.

In summary, our research methodology involves conducting interviews with software professionals and collecting relevant metrics to provide new insights. By following established research guidelines and utilising appropriate data analysis methods, we aimed to provide credible and informative findings that can be used to inform future software development projects.

2.2.1. Pre-study

The research team has had a background in software development and therefore brought some level of experience related to the study. This background allowed us to approach the research with a nuanced understanding of the challenges and opportunities associated with DevOps in software development projects. Additionally, our experience in software development enabled us to ask informed questions during the interviews and interpret the data collected in the context of software development practices.

2.2.2. Interviews

One essential data collection method was semi-structured interviews with software professionals who have hands-on experience with DevOps. These interviews will hopefully provide valuable insights into the challenges, opportunities, and effective strategies for measuring DevOps metrics in software development projects.

To ensure consistency in data collection, we developed a set of predefined questions that cover various topics related to software delivery performance. These questions were built on initial questions asked in interviews conducted by the dedicated DevOps team of the organisation with other teams. These questions are listed in the Appendices section. However, the interviews were decided to be semi-structured to allow for flexibility and spontaneity. This means that follow-up questions and prompts were to be used to explore interesting points in more detail or to clarify any ambiguities.

All interviews conducted by us were initiated by explaining the purpose of the interview and also clarifying that the answers are to be anonymous and the data is going to be protected by destroying the original recordings after the transcription.

The interviews were then transcribed and archived in a public GitHub repository [11].

2.2.3. Data collection

We invited software practitioners of the software organisation to participate in interviews. The interviews were conducted physically or virtually and recorded for later analysis.

We focused our interviews on a single software development organisation. This decision was made to ensure a deeper understanding of the DevOps metric measurement implementation within this organisation, as well as to facilitate the interview process by having easier access to potential interviewees.

We recruited interviewees from various teams within the organisation who have hands-on experience with DevOps metric measurement implementation, focusing on software engineers. These interviewees were chosen based on their roles, expertise, and availability. We also ensured that we have a diverse set of interviewees to capture different perspectives and experiences with DevOps metric measurement implementation.

While we understood that limiting our interviews to a single company would reduce the generalisability of our findings, we believed that the in-depth insights we were to gain from this approach would still provide valuable insights into DevOps metric measurement implementation. In addition, we chose to supplement our findings with insights and best practices from the existing literature and case studies.

To ensure ethical standards are met, all interviewees were informed of the study's purpose and their participation's voluntary nature. We also obtained their informed consent and ensured the confidentiality and anonymity of their responses.

Even though the interviewees have not seen the results in a concrete form, the aim is to present the final results from the research in a planned event as requested by one of the engineering managers of the organisation.

2.2.4. Data collection of DORA metrics

Since we were concerned with software delivery performance, we focused on collecting DORA metrics from the product teams that measure them. More specifically, we collected the two metrics called *Deployment Frequency* and *Lead Time for Change*. These would in turn give insights into the process of software development within teams.

The metrics collected would then be compared between teams to get an understanding of which teams perform better in terms of metrics, and that combined with the semi-structured interviews would provide new insights about the state of the teams, what can potentially improve the metrics and what can lower them.

This study aims to focus on analysing metrics about software delivery. We, therefore, need data about how often product teams deploy to production. We also need data about how long it takes from code committed to code running in production. These two types of data will bring new insights about the throughput of the software teams which make up a significant part of overall software delivery performance.

There has been wide interest from software organisations to measure this type of software delivery performance metrics, but the companies have stated that measuring DevOps progress is difficult, and there has not been a defined model or strategy for how to perform this measurement. Forsgren et al. attempted to search for a performance measurement of software teams which focuses on the global outcome in DevOps and not pure output, i.e. measuring the results that add business value instead of the mere amounts of work. The four key metrics of DevOps are today widely applied in today’s industry and used by companies such as Zalando, RedGate, HelloFresh, PBS and Contentful [1]. Sallin et al. noted that no scientific research had been done which suggests how to automatically measure the four key metrics of DevOps [1]. There is a big interest from software organisations to measure their software delivery performance. However, measurement of the four metrics is often done manually and through surveys [1].

To make sense of this data, we need to compare it between teams to learn why some teams may deploy more often than others and what enables them to deploy more frequently than others and why some teams have shorter or longer lead times for change than others.

As Höst and Runeson suggest [8], a recommended approach to gather metrics is to use the Goal Question Metric method (GQM) in which goals are first formulated, and the questions are refined based on these goals, and after that metrics are derived based on the questions. In our case, we can not follow the GQM approach because we study already available data that has been collected over time by the software teams. Therefore, as researchers, we are aware that we can not control the quality of the collected data, and there is a risk of missing important data. We still believe that the collected data is of value and that many insights can be gained from analysing it.

2.2.5. Analysis of the DORA metrics

To analyse the data we collected from the software teams about deployment frequency and lead time for change, we conducted a common statistical test called the analysis of variance (ANOVA) test which has been defined as a statistical formula used to compare variances across the means or average of different groups. It can be used to determine if there is any difference between the means of different groups [12]. We used the results from this test to accept our null hypotheses or reject them. This analysis was then used to answer RQ1.

2.2.6. Analysis of the interview data

After completing the interviews, we transcribed the result and analysed the data. The analysis involved identifying themes and patterns across the data, as well as comparing and contrasting the strategies and experiences of different interviewees.

In summary, the data collection of the raw data from teams was done first. Then, we analysed the raw data. After the data analysis and the new insights we learned from it, we then formulated interview questions. This data was then used to answer RQ2 and RQ3.

2.3. Reliability and Validity

2.3.1. Reliability

To collect DevOps data, a centralised data source was used where product teams had submitted their information, resulting in a more reliable process.

To improve the accuracy and reliability of the interviewing process and its responses, we recorded sessions instead of relying on note-taking or memorisation. However, we are aware that data analysis could be problematic as the data may be interpreted in various ways and its limited scope may restrict the generalisability and any drawn conclusions.

Another factor that could affect reliability in terms of interviewing is if the interviewer is the same for all of the interviews. To mitigate this, we split up the process of interviewing between both researchers of this paper and let one researcher interview two of the teams, and the other researcher interviews the third team.

Although the semi-structured interview method offers flexibility, there is a risk that it leads to uneven questioning. To mitigate this risk, we ensured a standard set of questions for every interview.

Reliability could also be affected by the risk that the questions are interpreted differently by each interviewee. To mitigate this, we attempted to explain our intent behind each interview question and also let the interviewees ask their questions about how they understood our interview questions. We also asked them to verify our understanding of their answers.

2.3.2. Validity

Construct validity deals with interpreting theoretical constructs. Throughout this project, a common term used is the term *deployment*. Since deployment is a very broad concept and people can mean different things with it, there is a risk that the reader will interpret deployment in a different way than we intended. We chose to mitigate this risk by thoroughly describing what deployment means in the context of this project as well as how the DevOps metrics are related to the deployments.

Internal validity deals with validating that the results and conclusions follow the collected data. Since we are interviewing members of the different product teams, one threat to validity could be that their personal opinions on their ways of working may dictate the answers that they provide, thus affecting our interpretation of the data and our conclusions. Another threat could be systematic errors in the data collection process.

External validity deals with validating the justification of generalising the results. As this thesis was a case study, it was naturally restricted to one single case and could therefore not be generalised in a broader sense. We also only investigated three teams out of all software teams of the software organisation, which is a relatively small number and the generalisability is therefore not possible in this case study.

2.3.3. Selection of methodology

We considered other methodologies such as performing a controlled experiment and collecting the data by ourselves instead of relying on existing data collected by the organisation. Even though we were aware that we would gain more control by measuring and collecting the data on our own, the time constraints did not allow us to perform such a study.

We also considered doing a more quantitative type of analysis, but since we were more interested in the reasons for fluctuations in software delivery performance metrics, it made more sense to do a more qualitative analysis.

2.3.4. Sampling strategy and size justification

Our sampling strategy was to select three teams that we labelled to be average, below average and above average as per recommendation from engineering managers and DevOps specialists who gave their views on the teams that they recommend analysing further based on their previous records of software delivery performance. We also looked at their existing data on DORA metrics and decided to select them as subjects for the case study. We limited the number of teams to three out of all 21 software teams primarily because we would not be able to manage to analyse all the existing teams' data in a short period. We acknowledge that there could be a potential bias in following the recommendations of the engineering managers.

2.3.5 Triangulation

We designed our process of collecting data by starting with the raw data collection before anything else and then reading the already conducted interviews by the dedicated DevOps team with the other teams, and finally designing interview questions whích discuss things that have not already been discussed during previous interviews. In this way, we ensure that the interviews that we conduct are built on earlier findings and insights from both quantitative and qualitative data, and we avoid duplicated interview questions or questions that are not based on existing data.

2.4. Ethical considerations

To maintain the confidentiality of the business and colleagues, it was crucial that we refrain from divulging any personal information or sensitive information that could jeopardise the organisation's operations. We took steps to anonymise all data collected and present it in a manner that omits specific details of the organisation.

During the interviews, we recorded the sessions but ensured that the recordings were deleted once we had transcribed the responses. We also ensured that no personal information about the interviewees is included. We also requested one of the engineering managers to review the thesis and give their approval before it was sent for peer review.

In terms of data collection, we only interviewed teams that had access to the DORA data. Additionally, we obtained consent from all interview participants to share their information in a thesis project while emphasising our commitment to ethical practices, such as anonymisation.

We followed a formal process set up by the software organisation and signed a digital agreement with the rules and conditions for conducting a thesis project. We also went through the rules in a separate session together with an engineering manager and agreed to follow their guidelines throughout the process of writing the paper.

For integrity purposes, we also sent the paper to an engineering manager to review it and give their approval before sharing it with the supervisor and examiner.

3. Theoretical background

3.3. Definition of terms

3.3.1. Continuous integration, continuous delivery and continuous deployment

Continuous integration (CI) has been defined as a part of the development process that automatically builds an artefact and runs a series of automated tests for every code commit to assessing whether the code is ready to be deployed [2]. Continuous delivery is the practice that enables the team to deploy software to production or end users at any time and ensures the software is in a deployable state throughout its lifecycle [2]. It is important to note the difference between continuous delivery and continuous deployment - continuous delivery implies that a software build can be deployed at any time, whereas continuous deployment (CD) means that every software build is automatically deployed to production [2].

M. Shahin et. al. [13] describe continuous integration and deployment as a set of practices that enable organisations to frequently and reliably release new features and products. The frequency and reliability that the authors refer to are what this study aims to focus on. These practices enable software organisations to ship new features and patch bug fixes on a continuous basis which in turn increases customer satisfaction. Along with it comes a variety of approaches, tools and strategies to make continuous integration and deployment work as optimally as possible, as well as several challenges that CI/CD teams inevitably have to face.

The practice of CI/CD emerged as an effect of the increasing demands of the competitive software industry to produce and deliver high-quality software at a high pace. Some benefits of CI/CD are customer satisfaction, quicker iterations of the software development life cycle, faster releases and automation of tedious deployment processes. [13]

3.3.2. DevOps

DevOps has been defined as a set of practices, tools, and a cultural philosophy that automates and integrates the processes between software development and IT teams [18].
The term first appeared in 2009 in social media coined by Patrick Debois [1]. DevOps has also been defined as a set of practices intended to minimise the time it takes from making a change to a system to placing it in production while ensuring high quality [19].

3.3.3. DORA metrics

DORA is an organisation founded by Nicole Forsgren and Gene Kim which is famous for their annual “State of DevOps” report [14]. The DORA organisation has defined a set of four key performance indicators in DevOps that can be used to measure the performance of a DevOps team [1].

The **deployment frequency** metric measures how often a team deploys to production. It is calculated by dividing the number of deployments by the total number of hours worked. A high deployment frequency indicates that a team can quickly and efficiently deploy to production, which can result in faster feedback and better-quality software.

The **lead time for change** metric measures the time it takes for a team to make a change to a system and have it deployed to production. It is calculated by measuring the time from when a change is committed to the time it is deployed. A team's ability to quickly deliver changes to production is reflected by a low lead time for change. A low lead time for change, similar to a high deployment frequency, results in faster feedback and higher-quality software.

The **time to restore service** metric measures how quickly a team can restore services after a system outage or failure. It is calculated by measuring the time from when a failure occurs to when the service is restored. A low time to restore service indicates that a team can quickly identify and resolve issues, which can result in better availability and reliability of the system.

The **change failure rate** metric measures the percentage of changes that result in a failure or require a rollback. It is calculated by dividing the number of failed changes by the total number of changes. A low change failure rate indicates that a team can deliver changes to production with minimal risk and disruption.

By measuring these metrics, teams can identify areas for improvement and implement changes that can result in faster, more efficient software delivery.

A high deployment frequency and low lead time for change indicate that the development team can deploy changes quickly to specific parts of the application without affecting the entire system, reducing the risk of errors and improving the overall efficiency of the development process [15].

One of the key software delivery performance metrics is deployment frequency that provides insights into the frequency at which a software team releases code to end users or deploys it to production environments. In simpler terms, it answers the question of how often software changes are made available to users or put into operation. [2] According to the State of DevOps report from 2022, high performers are considered to deploy on-demand, which can mean multiple deploys per day, medium performers have a deployment frequency between once per week and once per month, and low performers deploy between once per month and once every 6 months [2].

Deployment frequency is classified as daily when the median weekly day count with at least one deployment which is successful equals or is bigger than three days [16]. In simpler terms, to classify deployment frequency as daily, the software must be deployed on most working days. If the software is deployed most weeks, it will be weekly, and then monthly [16].

Deployment frequency is well defined by Forsgren et al. as the deployment of software to production, and as a result, other definitions are close to Forsgren's definition. The definitions all share the inclusion of "number of deployments in a certain period". Some mention that they only count successful deployments, and some explicitly mention deployments to production. Some include every deployment attempt even if it was not successful as this is a speed metric [1].

It is worth noting that different terms are used in various contexts like *DORA metrics*, *DevOps metrics* and *the Four Key Metrics of DevOps* (FKM) which are all terms for the same concept.

It is important to note in this context that the metric is deployment frequency and not deployment volume. Therefore, it is not sufficient to show the volume of daily deployments or to show the average weekly deployment count [16]. In the context of the case study, the deployment frequency implies a counter of the number of deployments to different environments.

Another topic is defining a successful deployment to production. It is up to the organisation to define this, like including deployments that are only to 5% traffic for example, but in general, any successful deployment to any level of traffic is included [16]. The organisation of this case study has decided that deployment to the production environment is when the deployment can be reached by users. If a deployment is made to a production environment for the sake of protected internal testing, then this is not considered to be a deployment to production per definition. By protected, we mean that the exact version number is hidden from end users.

The process for collecting data for deployment frequency is integrated into the GitHub workflows. The typical way to collect deployment frequency metrics is to send a request to an API as a step of the workflow that is triggered when publishing a new version to production.

After the request is sent to the API, data is generated in Amazon Web Services (AWS) and transformed into Azure Databricks. Reports can then be generated with Power BI.

Another key software delivery performance metric is the lead time for changes, which answers the question "How long does it take to go from code committed to code successfully running in production?" [2].

As for the definition of lead time for change, all suggestions on the original definition by Forsgren measure the time a commit takes until it reaches production, whereas the only difference is the way that they aggregate (mean, median, etc.) [1]. Since it is a common practice to use version control for altering source code, the commit is defined as the change. The lead time is given by the time between the timestamp of the commit and the timestamp of the deployment [1].
According to the State of DevOps report from 2022, high performers have a lead time of between one day and one week, medium performers have a lead time of between one week and one month, and low performers have a lead time of between one month and six months [2].

To measure lead time for changes, two types of data are needed: the timestamp of the first commit, and the timestamp of the deployment [16].

In the context of the case study, lead time implies the time between the very first Git commit date and the date when the requirement was deployed.

Solutions may only send data when a pull request has been merged. This is because many pull requests are closed and abandoned, and they are of no interest for measurement.

A script is used for calculating the lead time. When calculating stop time for several merged pull requests, the script depends on a changelog generator. The file produced is then parsed and the stop time for the merged pull requests is sent to the API. Figure 4.1 illustrates an example of a merged pull request to a branch. When a developer merges a feature branch into the master branch, a GitHub action, also known as a pipeline, runs in which the artefacts are deployed to the cloud using a service like Azure Blob Storage. In this pipeline, a script is executed which sends the timestamp for the first Git commit to an API which stores this data.

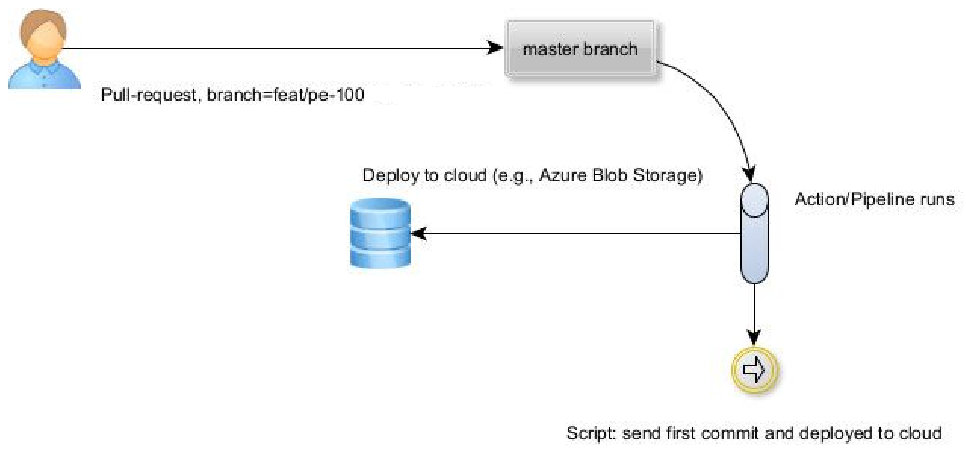

master branch
Pull-request, branch=feat/pe-100
Deploy to cloud (e.g., Azure Blob Storage)
Action/Pipeline runs
Script: send first commit and deployed to cloud

Figure 3.1: Example of sending first Git commit date

It is worth noting that the speed metrics are more precisely defined than the stability (quality) metrics, and as a result automating the measurement of speed metrics is easier than automating the measurement of quality metrics [1].

3.3.4. Inter-team dependencies

According to Bick et al. [17] inter-team dependencies refer to coordination of activities between teams. In other words, situations where one team's work or output depends on the work or output of another team.

These dependencies can arise due to shared code, data, or resources between the software. To ensure that the different teams work seamlessly together, the different teams need to coordinate their development efforts and manage their dependencies effectively.

This can involve establishing communication channels between teams defining clear interfaces and construct, and aligning development and deployment processes. Effective management of inter-team dependencies can help to reduce conflicts and improve the overall quality and maintainability of the system.

3.3.5. Deployment

Deployment in the context of this software organisation means releasing a new version of a solution to the production environment. When a version is deployed, it automatically reaches end users of the application.

3.4. Organisation structures and processes

In this section, we will briefly describe the structures of the organisation and its processes.

3.4.1. Structures

The organisation develops web solutions for an online business. The organisation develops two types of solutions:

1. Customer-facing solutions. These are the solutions that are shipped to and used by end users, and the ones that we will investigate further in terms of DORA usage and software release processes.
2. Supporting solutions. These provide tools and services that simplify and support the development of customer-facing solutions. They are not directly used by end users, but rather consumed by the customer-facing solutions.

There is a one-to-many relationship between the teams and products. One team can own more than one product, and one product is owned by one team.

Software teams consist of members with different types of expertise. A typical software team consists of software engineers, a technology lead, a solution architect, UX designers, a business analyst, a scrum master and a product owner.

There is also much collaboration on a cross-functional level. Software engineers work together across teams to find general solutions to recurring problems. UX designers, business analysts and product owners also have joint meetings to synchronise their work, backlogs and product roadmaps.

The architectures of the different solutions can vary a bit depending on the history of the solution and its natural evolution. However, there are common patterns for all customer-facing solutions. They consume services for data storage, internationalisation, UI libraries and core services that handle critical business logic.

Figure 3.2 illustrates a typical architecture of a customer-facing solution.

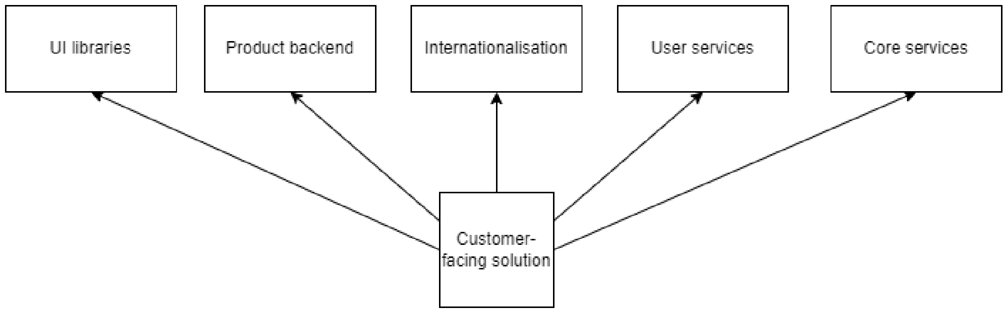


Figure 3.2: Architecture of a customer-facing solution, reworked

3.4.2. Processes

All the software teams of the organisation adopt and follow an agile mindset in the way of working, using the Agile Manifesto as a general guideline [36]. The aim is customer satisfaction, which is achieved by frequent delivery of high-quality software. The teams follow the *fail early, fail fast* approach and strive to continuously learn and adapt.

The organisation is also actively adopting a DevOps way of working focusing on frequent deliveries with speed, quality and stability and getting feedback as early as possible.

Software teams usually follow a type of Scrum or Kanban methodology of working. Software teams work in a sprint-based manner, where the work items of a sprint are planned and included, implemented, tested, deployed and finally presented to a bigger audience.

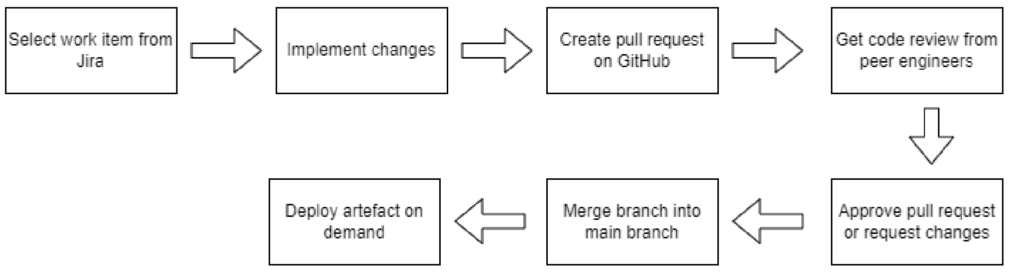
Product owners prioritise work items in the backlog and synchronise roadmaps with each other to stay aligned with the upcoming work. Business analysts assess the current risks and drive actions to mitigate and minimise risks. The scrum master is responsible for coordinating the daily work of the team, supporting the team dynamics and ensuring that all team members can operate productively. Deployments are expected to happen frequently. The management members synchronise the overall objectives and goals with the software teams frequently to ensure that everyone shares a common view of the goals.

From a technical point of view, all teams work in code repositories on GitHub. Software engineers can contribute to different code repositories when needed, and engineers may move between teams and support them when needed.

Both the outer loop of the software development methodology, like merging code reviews, automating deployments and testing the system outcomes, and the inner loop of the software development methodology, like designing solutions, implementing them in code and testing them on different levels are similar for the software teams.

Teams work autonomously, meaning that each software team is responsible for the whole software engineering lifecycle, from design to implementation and testing. There are overall guidelines that all teams are expected to follow, but the members of the team ultimately decide their internal tools and the way that they choose to operate.

The main version control tool used by all teams is Git. Software engineers pick up work items from Jira, work on them in separate Git branches, make the required changes and create pull requests on GitHub to get feedback from their peer engineers. A pull request is then either accepted or declined by the code owners of the repository and merged into the main branch. Figure 3.3 illustrates the workflow of a software change from design to deployment.


Select work item from Jira
Implement changes
Create pull request on GitHub
Get code review from peer engineers
Approve pull request or request changes
Merge branch into main branch
Deploy artefact on demand
Figure 3.3: The workflow of a software change

The process for deploying the software into the production environment may differ for some teams, but usually each software build is in a deployable state.

Software teams are expected to measure the throughput and stability of their software deliveries through DORA metrics.

The process for collecting the DORA metrics is integrated into the GitHub workflows that are developed and maintained by the engineers of each team. All teams that measure DORA metrics measure the same type of metrics using common GitHub workflows.

4. Research project – Implementation

In this chapter, we will provide an overview of the case study activities. Section 4.1 defines the main hypotheses of the case study. Section 4.2 discusses the collection of DORA metrics and reports to gain a deeper understanding of the performance and throughput of software teams within the organisation. In addition, Section 4.3 details the semi-structured interviews conducted with software engineers to gain insights into the challenges and factors that impact software delivery performance. Finally, Section 4.4 explains how the collected data was analysed and processed in the case study.

4.1. Hypotheses

The null hypothesis for deployment frequency is that there is no significant difference in deployment frequency between the teams. The alternate hypothesis for deployment frequency is that there is a significant difference in deployment frequency between the teams.

The null hypothesis for lead time for change is that there is no significant difference in lead time for change between the teams. The alternate hypothesis is that there is a significant difference in lead time for change between the teams.

4.2. Data collection of DORA metrics

In this section, we discuss the data collection process for our case study. We obtained reports for deployment frequency and lead time for change for three software teams and analysed their GitHub workflows integrated into the CI/CD pipelines.

To ensure the case study's primary focus on software delivery performance, it is essential to collect the DORA metrics of each software team within the organisation. This quantitative data collection is crucial to the overall outcome of the study since it can provide additional insights into how different teams perform in terms of software delivery.

The software teams under investigation have collected data on deployment frequency and lead time for change over a specific period, which is stored on the Microsoft platform Power BI. We exported the data from Power BI to Excel sheets.

To obtain technical information regarding the collection of DORA metrics, we analysed the GitHub workflows integrated into the software teams' CI/CD pipelines.

We began the data collection process by obtaining reports for DORA speed metrics including deployment frequency and lead time for change. As we had decided to select only three software teams due to time constraints (average, above average and below average), we asked engineering managers for their recommendations of teams that fulfil this criterion. We then selected three three software teams that we believed had varying scores in the area and analysed their data. We used the overall criteria from the State of DevOps report [2] to determine what is considered average performance. For reference purposes, we assigned the names *Team 1*, *Team 2*, and *Team 3* to these teams, with Team 1 being the above-average performer, Team 2 being average and Team 3 being below average.

4.3. Interviews

In this section, we discuss the interviews conducted to gain a better understanding of how DORA metrics can be interpreted. We interviewed three software engineers, each from a different software team, and asked them questions related to their team structure, software development lifecycle, release process, deployment frequency and lead time for change, and factors that impact them. The interviews were semi-structured and followed a set of introductory questions about the background of the subject.

4.3.1. Conducting interviews with software engineers

A primary purpose of the case study is to gain a new understanding of how DORA metrics can be interpreted. Therefore, interviewing software practitioners was a significant activity of the case study.

We used the case study paper of Höst and Runeson [8] as guidelines for conducting our interviews. As the goal of the interviews was to learn more about the challenges of the software delivery process, it was convenient to allow for flexibility and thus adopt semi-structured interviews as a way of interviewing.

We created a list of questions to ask every interviewee. As the interviews were semi-structured, we didn’t necessarily follow the order of questions, and the interviews were more flexible.

As suggested by Höst and Runeson [8], we divided the interview into several phases:

1. Present the overall objectives of the interview and the case study.
2. Explain how the data from the interview will be used.
3. Ask a set of introductory questions about the background of the subject.
4. Ask the main interview questions and ensure the confidentiality of the interview and that sensitive information will be protected.
5. Present the reports and discuss them with the interviewee.
6. Summarise the major findings by the researcher at the end of the interview.

The list of the main interview questions is as follows:

1. Please explain your team structure.
2. Please briefly explain your work process from sprint planning to release.
3. How often do you aim to deploy to production?
4. How quickly do you wish the time to be from the code commit to the code running in production (lead time for change)?
5. Do you regularly look at DORA metrics? How do you act accordingly? If not, what may be possible reasons for not looking at it?
6. How do you explain your deployment frequency report? (show report)
7. How do you explain your lead time for change report? (show report)
8. What are factors that may impact your deployment frequency?
9. What are factors that may impact your lead time for change?

We chose to look at the reports of deployment frequency and lead time for change together with the engineers to find explanations for potential causes that impact the metrics.

4.3.2. Collecting interview data conducted by the DevOps team

The organisation’s dedicated DevOps team has conducted interviews with several software teams, covering topics such as collecting and using DORA metrics, as well as the software release process and CI/CD practices. Although this data does not directly answer any of our research questions, it provides valuable insights into the overall adoption of collecting and analysing DORA metrics by the software teams.

We acknowledged the importance of incorporating the perspectives of the DevOps team in our research. Therefore, we carefully integrated their interview findings into our study from the beginning, to avoid duplication of any questions already asked during their interviews. This approach ensures a cohesive and comprehensive analysis of the organisation’s software delivery performance.

4.4. Data processing

To process the DORA metrics collected from the product teams, we used statistical tools and data visualisation techniques. We utilised data visualisation tools like Power BI, which enabled us to create interactive dashboards and graphs to identify trends and patterns in the data. We compared the metrics between teams to gain insights into their software development process and identify any trends and patterns in the data. We compared the metrics between teams to gain insights into their software development processes and identify any patterns or trends across the organisation. Our data processing methodology was documented in detail to ensure the transparency and replicability of our study.

For the deployment frequency and lead time for change metrics, we analysed them by performing a common statistical test known as analysis of variance (ANOVA). We used Excel for performing the ANOVA data analysis and specifically the Analysis Toolpak add-in for Excel which has built-in functionality for executing ANOVA tests. We inserted the data that we had collected from the teams and ran the tests. We collected the results and observed the P-values to determine if the results hold as sufficient evidence to reject the null hypothesis. We defined the significance level to be 0.05 as it is common to be a standard value.

5. Results

In this chapter we present the results from the case study which contain software delivery performance metrics and the data from the interviews.

5.1. Deployment frequency metrics

Figure 5.1 illustrates the deployment frequency metrics for the three teams from July 2022 to March 2023.

The graph shows how often the three teams deployed to production during this period. The Y-axis represents the number of releases made per month. Each team is assigned a unique colour for clearer visualisation. We can note for example that Team 1 deployed 5 times per month during August, October and January, whereas Team 2 deployed 5 times in March. On the other hand, Team 2 only deployed once in July and January, and Team 3 only deployed once in August, December and January. It is also notable that Team 1 has always deployed more than once during this period.

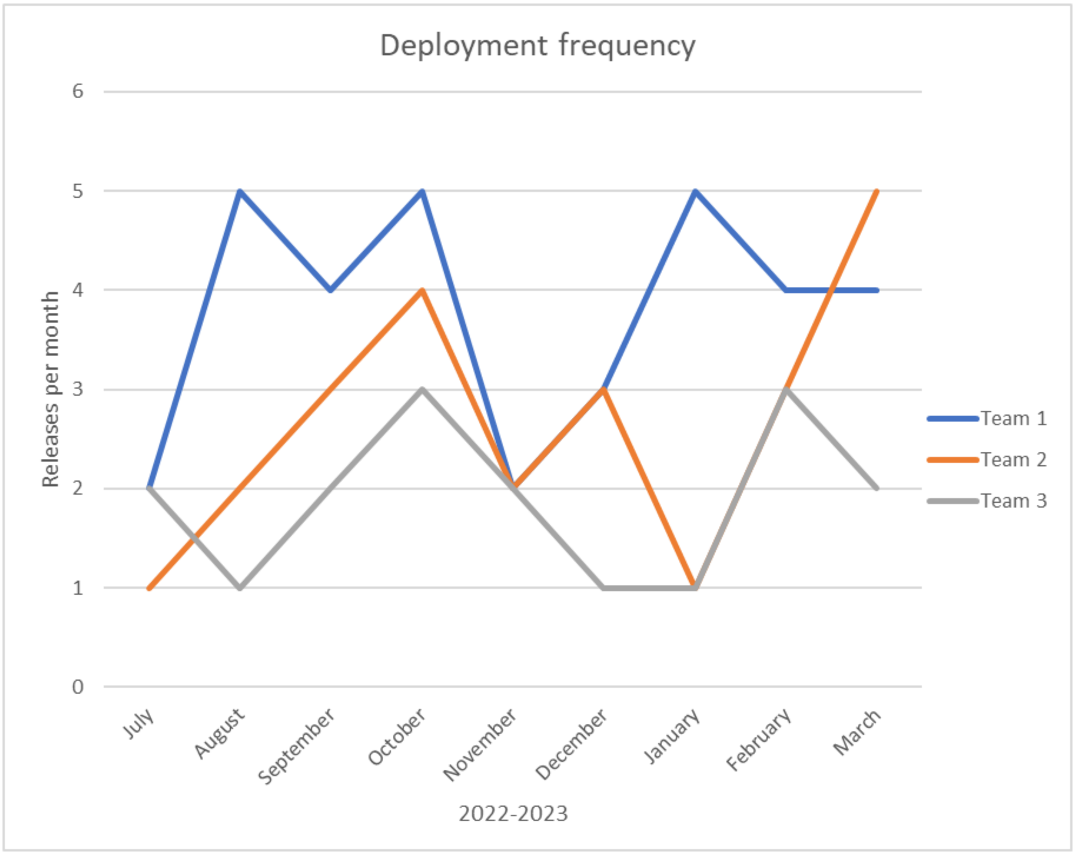


Figure 5.1: Deployment frequency metrics

Table 5.1 illustrates the deployment frequency metrics for the three teams from July 2022 to March 2023. The table shows numbers that represent the number of releases for each team from July to March. The first column represents the months. The table is grouped by columns.

Month	Number of releases for Team 1	Number of releases for Team 2	Number of releases for Team 3
July	2	1	2
August	5	2	1
September	4	3	2
October	5	4	3
November	2	2	2
December	3	3	1
January	5	1	1
February	4	3	3
March	4	5	2

Table 5.1: Deployment frequency metrics

5.2. Lead time for change metrics

Figure 5.2 illustrates the lead time for change metrics for the three teams from July 2022 to March 2023.

The Y-axis represents the number of days it takes to go from code committed to code running in production. We can note that Team 3 has the highest spikes in the graph, and Team 1 has the lowest spikes.

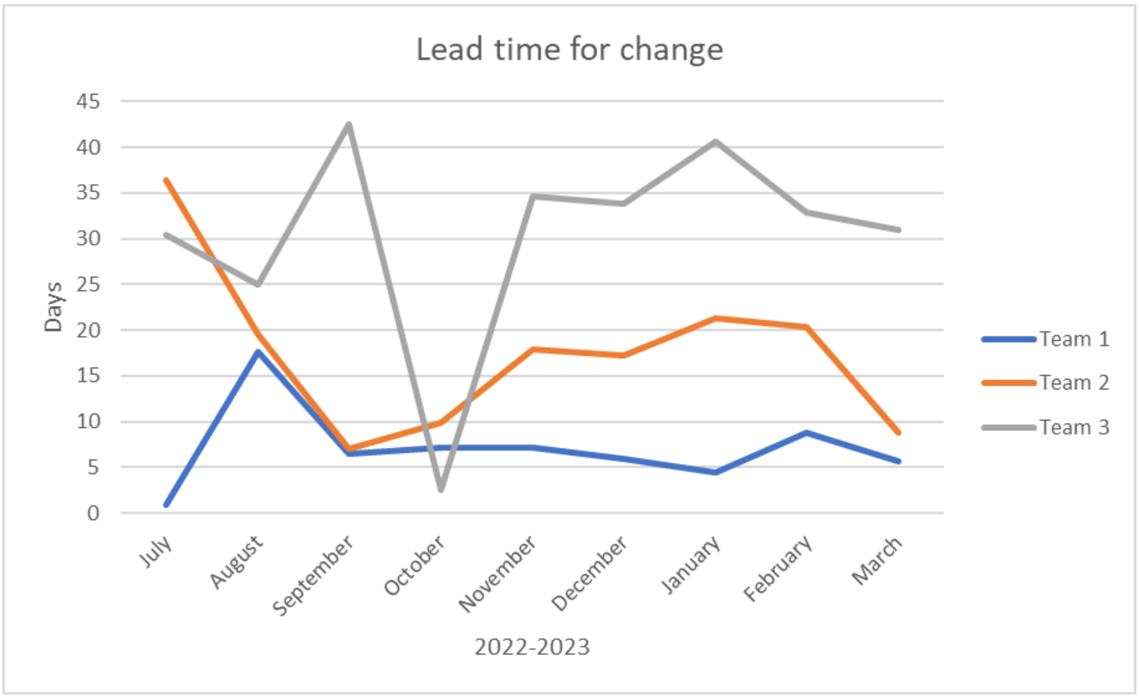


Figure 5.2: Lead time for change metrics

Table 5.2 illustrates the lead time for change metrics for the three teams from July 2022 to March 2023. The table represents how long it took for each team to go from code committed to code running in production during a given month. The table is grouped by columns, whereas the first column represents the months.

Month	Lead time for change in days for Team 1	Lead time for change in days for Team 2	Lead time for change in days for Team 3
July	0.83	36.36	30.39
August	17.56	19.48	25
September	6.5	7.07	42.53
October	7.15	9.89	2.56
November	7.14	17.89	34.66
December	5.85	17.23	33.84
January	4.42	21.27	40.6
February	8.77	20.3	32.81
March	5.61	8.83	31.01

Table 5.2: Lead time for change metrics

5.3. Interviews conducted by us

We let the software engineers look at the graphs together with us to get their insights into how the data can be interpreted and learn about factors which may impact it.

5.3.1. Interview with Team 1

Team 1's structure was described as follows:

- One product owner.
- A scrum master.
- Two to three UX designers.
- Six to seven software engineers.
- One person specialising in data collection from the application.

Their way of working can be summarised in the following points:

- Communication is facilitated through Slack and Team meetings.
- The team works in two-week sprints and has sprint-related meetings every other week.
- The teams hold weekly check-ins to review their progress.
- They manage their development work through a backlog.

Their release process can be described with the following points:

- They release features (completed and partially completed) and bug fixes weekly. They aim to ensure a lead time of one week for the majority of commits.
- Features that are not ready for release are hidden behind feature flags.
- They deploy changes to a production environment for testing purposes which resembles the development environment before the release.
- They conduct a weekly test session on Monday.
- They release on Mondays if the test session is successful, otherwise they resolve issues and postpone the release to Tuesday or Wednesday.

As for the DORA metrics, they stated that they don't give due attention to them. Some of the reasons are:

- Lack of knowledge and experience to comprehend the significance of the data and what it signifies.
- Lack of training in understanding what data is being collected and why it matters.
- Lack of awareness of the data's importance.

When asked about reasons for fluctuations in the deployment frequency report, they answered:

- If they don't release in a week, there are more releases the following week.
- The data was not accurately reflecting reality, probably due to changes to the DORA metrics script.
- Vacation time.
- The number of Jira IDs released, as some pull requests may not be linked to Jira IDs.
- The size of the tasks.

When asked about factors that cause variations in the lead time report, they answered:

- Changes to the DORA metrics script which could cause inaccurate statistics.
- During summer, the team likely did not release every week. There could be periods of up to five weeks without any releases.
- The team's workload.
- The team's release schedule.

However, despite these variations, the team's average lead time of eight days suggests that the team is meeting its targets for weekly releases.

5.3.2. Interview with Team 2

When the engineer in Team 2 was asked about their team structure, they said that there are three software engineers, one UX designer, one half-QA engineer (working part-time) and one product owner.

Their way of working could be summarised as follows:

- They follow a loose, Kanban-like sprint process.
- They follow Scrum routines but are not structured.
- They work in an investigative way and often open pull requests as drafts.

Their release process can be described in the following points:

- They try to keep their main branch clean and releasable.
- With their new QA engineer, they do integration testing and release testing.
- They have no fixed date for release but release when it makes sense.
- They aim to deploy to production as often as possible.
- They batch together a group of features and bug fixes in releases.

We both agreed that the DORA metrics assume that a team works in a very concrete way, where they pick up tasks, implement them and release them. It presumes that you need to know what to do from the first commit, and that is rarely the case in their team.

When asked if they regularly look at DORA metrics, they answered:

- They could not relate the numbers to what they had been doing during the previous few weeks.
- There could be an issue with how they report the data, or maybe they are just not used to the Power BI tool.

From looking at the graphs for deployment frequency and lead time, we identified the following:

- We noted that some weeks were omitted and interpreted it to mean that the omitted weeks are weeks where no release was done, which was aligned with reality according to the engineer.
- The lead time charts were aligned with reality, indicating that they release twice a month.

The engineer shared some general thoughts about DORA metrics which can be summarised as follows:

- A possible reason why many teams don’t look at the DORA data nor understand it can be that the data is only valuable if you have a continuous delivery approach, which is not the case for many of the existing products.
- If there is no continuous delivery setup, then the data will always be dependent on factors that are outside of the control of the engineers. When a feature is ready for release, it won’t reach production because it can be stuck for weeks due to the translation problem for example.

5.3.3. Interview with Team 3

The software engineer from Team 3 was also a member of the dedicated DevOps team and could provide answers to general DevOps questions concerning the organisation as a whole.

Their team structure was described as follows:

- The team is shrinking since it is in hibernation mode.
- Before hibernation, there were 4-5 software engineers, two business analysts, a product owner, a UX designer, a tester and an architect.
- They had two backends in C++ and Java.
- Their frontend was in JavaScript and they had web services in Python.

Furthermore, the engineer described their way of working as follows:

- They worked in Scrum up until hibernation, and then they switched to Kanban.
- The duration of a sprint was three weeks.

- After each sprint, they promoted a release branch, tested it for 2-3 weeks manually, and then released it to production.

When asked about their release process, they mentioned:

- Very often, they had one version unreleased, since they had already finished the previous sprint, and at the same time they were waiting for testing the current sprint but they hadn't yet released the old version.
- Their aim to deploy to production was once per month.
- Their aim for lead time for change was at least a week.

Some of the factors impacting the deployment frequency and lead time reports that the engineer identified were as follows:

- Loss of data as a result of issues with reporting the data with the correct environments.
- The bigger numbers show the state of normal sprints, and the downfalls are mainly hotfixes.
- The collected data was not only from the customer-facing application but also from the backend services. The backend had a different release cycle than the frontend.
- The manual testing process which could postpone releases.
- The translation problem, where dedicated translators need to translate texts to the different languages in the supported countries, which could take about a week.

The engineer's general thoughts about DevOps were as follows:

- Most of the teams don't look at the data that they have collected. The DevOps team found out that most of the teams don't understand the data that they have collected.
- The DevOps team aspires to conduct workshops with them to explain what it means.
- Two main factors for why teams don't study the metrics are a lack of understanding as well as doubting the value of the data.
- Just merely looking at the data might not bring value in itself, but if a team aspires to improve their deployment workflows and become more effective when deploying, then this data can be very relevant to track the progress.

5.4. Interviews conducted by the dedicated DevOps team

5.4.1. DORA usage statistics

Eleven software teams were asked by the DevOps team if they collect DORA speed metrics. Out of these eleven teams, six teams answered "Yes".
The eleven software teams were also asked if they use DORA speed metrics, in terms of looking at it frequently, interpreting it and learning from it. Out of the eleven teams, all answered "No".

5.4.2. Software release process statistics

During the interview, all eleven software teams were asked if they utilise continuous integration, and the answer was unanimously "Yes". Additionally, the teams were asked about their use of continuous delivery, which refers in this context to the ability to fully automate release processes. It does not mean that every change is deployed, but rather that a team can automatically deploy to a staging environment, and when a decision is made deploy automatically to production. Five teams confirmed using continuous delivery, while two teams said they do not. One team reported that they deploy manually, and another stated that they rely on manual testing. Unfortunately, the remaining teams did not provide adequate responses. The interview also inquired about continuous deployment, where the production deployment occurs automatically after merging a pull request, and only two teams responded positively.

5.4.3. How DORA metrics are used in software teams

One team responded that they have integrated data collection of DORA speed metrics through their pipelines. They also suggested that their lead time is naturally short since there are only two software engineers in their team.

Another team answered that they do not collect DORA metrics at the moment, and they experience problems with that since their product consists of two distinct projects (a frontend and a backend). The DevOps team responded that it is possible to collect the DORA metrics separately for each project.

One team stated that they have recently started to gather DORA metrics. They also felt that using DORA metrics can be useful even if their team is small.

One team said that they had been introduced to DORA metrics and the pipelines were set up in the repository. They don't talk about it but they know what it is about. They would like to understand what information they can extract from the reports.

Another team said that they don't use DORA metrics as their development is currently on hold, but they showed interest in knowing more about it in the future.

One team stated that they have not implemented DORA metrics as the project is in an early phase, but it is planned to be done in the future. Also, they do not yet deploy to production but only to a development environment at the moment.

One of the teams said that they know nothing about DORA metrics. They only know that they are sending some data somewhere, but nothing else.

Another team said that they are trying to look at the report but they are struggling to understand what they are looking at. They also hinted that they don't feel that the data in the DORA report reflects the truth. They also track releases on GitHub. The DevOps team suggested that if they are collecting data for testing environments as well as production, there is an issue where some data will be missing in production depending on the workflow.

One team said that they are not using the DORA metrics at all. They don't think that they have the data and they don't know where to check the data. They don't know anything about DORA metrics and would like some background information on why it is supposed to be used.

One of the teams mentioned that they are actively using DORA metrics. They use it to analyse the team's behaviour and possible problems in the way of working. They are interested in extending the DORA speed metrics with the DORA quality metrics (Mean Time to Recover and Change Failure Rate).

One team stated that they don't use DORA metrics at all. As with some other teams, they don't know where to find the data, and even if they would, they doubt the value that it provides.

5.4.4. Goals and suggestions regarding software release processes

One team stated that since they are missing a tester in the team, pull requests are open for some time until another software engineer gets time to review and test it. This stops their pipelines. The team is working on automatic testing to speed up the process, but it's not clear if their project will go on and it's difficult to prioritise. They suggested having shared testers that work in different teams which could help small teams to speed up the release process. Another suggestion was to conduct a GitHub training for non-engineers like product owners and testers, as they sometimes are scared to work on GitHub.

One team stated that their short-term goal is to have a functioning CI/CD solution for the backend. Their frontend team is fairly mature. They try to align more with the organisation's ways of working like adopting GitHub action workflows and using Azure Artefacts instead of Node Package Manager (NPM). Long term they are looking to cut down their release cycles. The backend is deployed manually with a lot of manual testing. A release usually takes one to two days. An issue with making it faster is that one of the countries has additional steps that make it complicated.

One of the teams stated that they now have a continuous delivery setup so that when a task is done, it's deployed to production. They used to deploy once a month, but since the tester left the team, they have moved to a faster deployment system. They are very happy with the new release process, smaller deployment packages and more often with fewer changes. When there are translations needed, an email is sent and a week or so is given to the translators to do their part. They suggested improving the end-to-end tests and the environment where they are running. They are also migrating from Azure to GitHub Actions.

One team stated that they are in general satisfied with their release process at the moment. Tagging and changelog generation is done manually, but they intend to automate this in the future. They are also interested in adopting best practices.

One team mentioned that they are having some issues with Azure in general and that it would be nice to have additional assistance and guidance going forward.

One team raised a problem with their release process, namely that they don't have a strong workflow for patch releases. They release from the main branch, but after that, they keep working on the main branch when introducing new features. When they find critical bugs, they experience issues with the changelog generator. They concluded that as long as they keep the main branch stable, they can always deploy new features with bug fixes. They believe that when they put features in the main branch, then it is releasable. They raised a concern that there is a constraint from the translators where they need two weeks to translate. This results in the team keeping the pull request alive despite being ready to deploy. They were aware that this impacts their lead time. They have a strong desire to release them as soon as they are ready, but currently, it is not possible due to the translation problem.

One team mentioned that they are looking to improve their self-hosted runners since they have substantial end-to-end testing. There is an interest in the *large runners* from GitHub. It would be beneficial both from a financial perspective and also to free up engineering time from the team. Currently, there is a fair amount of effort going into maintaining and upgrading the solution for their runners. When they expand it will become tougher and more expensive. Their current end-to-end test suites are much slower running on their self-hosted runners compared to running them locally.

Regarding their release process, they mentioned that their current workflow is that every pull request is tested automatically and also manually with the team. Every Monday, they create a new build that the team reviews. If it's approved, they release it the same day. They are not currently comfortable releasing automatically. Their test suite is good, but it's not extensive enough for them to feel comfortable deploying without manual verification. Their solution is fairly complex, and if they are to have tests for everything it will be cumbersome to maintain.

Another team mentioned that they have an automatic release system in place, but the testing process needs to be improved to have it fully automated because they need to be sure that no issues are deployed with a new version. The team is also interested in using the GitHub Actions cache to speed up the build time of the pipelines. They are also interested in uploading compressed files in Akamai to speed up the release process and to start using the Akamai retention package for cleanup.

One team stated that their release process is currently being improved, and it's aimed to be so continuously based on the team's findings, discussions and outcomes. One long-term goal is to have the process completely automated. Today, their production deployments are done manually. Developer items end up in the backlog, but not support requests. Items are typically refined and checked for unambiguity and clearness. Software engineers then pick the task from the top of the board, develop the feature, mark it as ready for test, and finally release it.

6. Analysis

From the interviews conducted and the data collected from the software teams, it can be stated that the overall ambition of the software organisation is to be data-driven and to have a DevOps mindset when releasing and delivering software. There is ongoing work and aspirations from the teams to continue collecting data about DevOps and DORA, as well as let teams start collecting data if they haven't already.

While a majority of software teams are collecting data on DORA metrics, they are not actively using this data for learning or improving purposes. This is primarily due to a lack of understanding of collected data, doubts about its value, and concerns that the generated report may be misleading or misrepresentative of reality. Moreover, a reason why many of the product teams could not make sense of the data is that they don't follow a continuous delivery approach, and therefore their data will always be dependent on factors that the software engineers can not control, thus making the data irrelevant for them as they can not impact it.

Some of the reasons why some teams don't collect DORA metric data are that they either have too little knowledge about what they are, they believe that it is complex to set up since they have different repositories for different services, or that their development is currently on hold.

6.1. Deployment frequency average

Table 6.1 shows the deployment frequency average values for the three teams. This table is grouped by rows. The columns show sum, average and variance values. We can note that Team 1 has the highest average value and that Team 3 has the lowest average value. Team 2 has the highest variance value which indicates that their data points are the most spread out compared to the other two teams.

Team	Sum	Average	Variance
1	34	3.777777778	1.444444444
2	24	2.666666667	1.75
3	17	1.888888889	0.611111111

Table 6.1: Sum, average and variance values for deployment frequency

Team 6.2 shows the ANOVA results for the deployment frequency between and within the groups. It shows a set of different values related to the tests which are described in the captions. In our analysis, we are mainly concerned with the P-value as it will aid us in rejecting or accepting the null hypothesis. We can reject the null hypothesis as the P-value is less than our significance level of 0.05.

Source of variation	SS	df	MS	F	P-value	F crit
Between groups	16.222	2	8.111	6.394	0.006	3.403
Within groups	30.444	24	1.2685			
Total	46.667	26				

Table 6.2: ANOVA results for deployment frequency. Source of variation = The factor being measured to determine the extent of variation. SS (sum of squares) = The total sum of the squared differences from the mean for each variation source. df (degrees of freedom) = The number of independent pieces of information used to calculate the sum of squares, which is equal to the number of groups minus one for between groups and the number of observations minus the number of groups for within groups. MS (mean sum of squares) = The average sum of squares calculated by dividing the sum of squares by the degrees of freedom. F (overall F-value) = a statistical value that measures the ratio of the mean sum of squares between groups to the mean sum of squares within groups. P-value = The probability value associated with the overall F-value, indicating the statistical significance of the observed differences. F crit (critical F-value) = The critical value of F that corresponds to a significance level of α = 0.05, used to determine whether the observed differences are statistically significant or due to chance. [20]

6.2. Lead time for change average (days)

Table 6.3 shows the average lead time values for the different teams. The table is grouped by rows, where each row represents a team. We learn from the table that Team 1 has the lowest average value, whereas Team 3 has the highest average value. Both Team 2 and Team 3 have high variance values which indicate their spread out data points, which can also be noted from the earlier Figure 5.2 which represents a graph of lead time for change.

Team	Sum	Average	Variance
1	63.83	7.092	20.355
2	158.32	17.591	77.961
3	273.4	30.378	136.445

Table 6.3: Sum, average and variance values for lead time for change

Table 6.4 shows the ANOVA results for lead time for change between and within groups. The caption gives brief explanations of what each column means. As with deployment frequency, we are mainly concerned with the P-value of lead time for change to reject or accept the null hypothesis. As we can observe, the P-value is less than 0.05 and we can therefore reject the null hypothesis.

Source of variation	SS	df	MS	F	P-value	F crit
Between groups	2447.828	2	1223.914	15.640	0.000045	3.403
Within groups	1878.091	24	78.254			
Total	4325.918	26				

Table 6.4: ANOVA results for lead time for change. Source of variation = The factor being measured to determine the extent of variation. SS (sum of squares) = The total sum of the squared differences from the mean for each variation source. df (degrees of freedom) = The number of independent pieces of information used to calculate the sum of squares, which is equal to the number of groups minus one for between groups and the number of observations minus the number of groups for within groups. MS (mean sum of squares) = The average sum of squares calculated by dividing the sum of squares by the degrees of freedom. F (overall F-value) = a statistical value that measures the ratio of the mean sum of squares between groups to the mean sum of squares within groups. P-value = The probability value associated with the overall F-value, indicating the statistical significance of the observed differences. F crit (critical F-value) = The critical value of F that corresponds to a significance level of α = 0.05, used to determine whether the observed differences are statistically significant or due to chance.[20]

6.3. RQ1 Is there a significant difference in the DevOps speed metrics between teams?

From the data presented for deployment frequency in Table 6.2, we note that the P-value was 0.0059 which is less than the significance level (0.05). Based on this, we can reject the null hypothesis and accept the alternate hypothesis which is that there is a significant difference in deployment frequency between the teams. As anticipated, Team 1 outperformed the other teams. This significantly higher rate of deployments suggests that Team 1 has excelled in efficiently delivering changes to the production environment, consistently maintaining a faster pace of deployment compared to other teams.

As we first anticipated, Team 1 performed the best out of the three in terms of deployment frequency. Team 2 had a relatively average frequency of deployments, and Team 3 had the lowest performance of all.

Based on the data presented for lead time for change in Table 6.4, we observe that the P-value is also smaller than the significance level (0.05). Based on this, we can also reject the null hypothesis for lead time for change. Team 1 exhibited the best performance with an average lead time of 7 days. This indicates that Team 1 has been most efficient in delivering changes, with a short lead time from initiation to deployment. Team 2 fell, as suspected, in the middle range with an average lead time of roughly 17 days. This suggests that they have a moderate level of efficiency, taking a slightly longer time compared to Team 1 but still demonstrating a reasonably prompt deployment process. In contrast, Team 3 had the highest average lead time of 30 days. This indicates that they have been the least efficient in delivering software changes, experiencing a longer lead time compared to both Team 1 and Team 2.

6.4 RQ2 What are some of the causes for fluctuations in DevOps speed metrics?

Factors that can affect the deployment frequency can be summarised as follows:

- The level of utilisation of automated processes.
- Absence of a well-defined release schedule.
- Dependence on factors that the software teams can not control.

Based on the findings, it can be concluded that various factors can influence deployment frequency, including the process of manual testing after each sprint and the lack of automated deployment workflows. According to the interviews, manual processes were found to be slower and more error-prone than their automated counterparts, resulting in delays in the deployment frequency. The manual testing process, in particular, was identified as a bottleneck, as it requires software teams to invest more time in confirming that the code is release-ready. Additionally, the absence of automated deployment workflows contributes to the need for a manual release process, further slowing down the deployment frequency. Overall, the findings suggest that the implementation of automated processes can help streamline deployments and minimise errors, leading to more frequent and reliable software releases.

Furthermore, the findings of the study suggest that having a well-defined release schedule is a crucial factor that affects deployment frequency. Teams that lack a well-defined release schedule tend to release software inconsistently, resulting in difficulties in understanding the reasons behind fluctuations in deployment frequency. If a team decides to release every third week for example, but the graphs show that the team has not followed this release schedule, then it could be easier to investigate why the deployments happen less often. Moreover, the absence of team members due to vacation time can also have a significant impact on the deployment frequency, as it can lead to challenges in maintaining a consistent deployment schedule. Therefore, software teams need to establish a clear release schedule and plan for potential disruptions, such as team member absences, to ensure a consistent deployment frequency.

A recurring issue that arose was the reliance of the data on factors beyond the control of the software teams. One prominent example is the involvement of external translators responsible for translating application texts before the version can be released in different countries. The translators were only able to provide translations once a week at most, which prevented daily production deployments. However, this was a common issue across all teams as they are all within the same organisation, and the translation process follows the same pattern for each of the three teams.

Factors that can affect the lead time for change can be summarised as follows:

- The level of utilisation of automated processes.
- Working methodology.
- Team size and workload.
- The size and complexity of code commits.
- Pull request handling.

The findings indicated that the lead time for change is influenced by various factors that can affect the workflow and productivity of the software development teams. One of the primary factors is the manual testing process, which often results in pending releases and delays in implementing changes. Manual testing can introduce bottlenecks in the development cycle as it requires human intervention which is both time-consuming and error-prone. The need for manual testing can slow down the lead time, especially when there are limited resources available or when there is a high volume of changes to be tested. Moreover, manual testing is susceptible to human error, which can result in longer lead times if issues are not detected and rectified promptly. This can have a cascading effect, causing subsequent delays in the release cycle and overall project delivery.

Another important factor is the team's working methodology, where a structured approach such as Scrum can lead to better planning and time management, resulting in a shorter lead time for change. Moreover, it is also essential to maintain an optimal team size and manage workload efficiency. While a larger team often delivers more software, it does not necessarily imply better lead time. A small development team with a short code review and testing process can lead to a lower lead time for change. When there are fewer engineers in a team, it is easier to coordinate and manage the code review process. Additionally, a shorter testing process can be beneficial for a small team, as it reduces the time required for testing and deployment. However, it is important to note that a smaller team can also lead to a lack of diversity in skills and perspectives, which can affect the overall quality of the code being produced. Therefore, it is important to find a balance between team size, testing process, and code review to ensure a consistent lead time for change while maintaining code quality. Additionally, lead time can also be influenced by factors such as vacations and other time off. By establishing effective processes and ensuring proper knowledge sharing within the team, the absence of team members can be managed more effectively. This allows for smoother operations and helps mitigate any significant disruptions to workflow and project timelines.

Moreover, fluctuations in lead time for change can be affected by changes to the reporting scripts used to generate reports. This emphasises the importance of regularly reviewing and validating the accuracy of data reporting to avoid misrepresenting the actual lead time.

Furthermore, the workload of the team can also affect the lead time for change, especially when many urgent tasks need to be completed. This can lead to delays in implementing other tasks, increasing the lead time for change. Therefore, the software team must prioritise and manage their workflow efficiently to reduce lead time for change.

Additionally, smaller changes tend to be deployed more quickly, which could explain the shorter lead time for certain changes. Additionally, teams may sometimes start deployment on a specific feature but later prioritise other tasks, leading to the initial commit remaining inactive and resulting in a higher lead time for change. Therefore, while the average lead time provides valuable insights, it is essential to consider the context of each change. Other facts, such as the complexity of the change, prioritisation decisions, and external dependencies, can also impact the lead time. However, the teams need to keep these factors in mind. To enhance their lead time, they should consider breaking down complex features into smaller components, enabling an increase in the number of deployments. This approach will ultimately lead to reduced lead time and improved development agility.

Another significant factor that can affect lead time is the way pull requests are handled. If pull requests are opened as drafts at an early stage of an investigation, the time taken to implement the changes may not accurately reflect the overall lead time. This is because the time will be measured from the first commit which may only be a work-in-progress commit, thus not being relevant for measuring the overall time it takes to implement a task. On the other hand, one could argue that this data is still valuable, even if it is measured from the first commit in a draft pull request, as this indicates the start of a work item, and measuring from that point of time is more accurate for studying how long tasks take from the time of the investigation. It could also be aligned with the common Agile methodology *stop starting, start finishing*, and act as a way of measuring work-in-progress items.

6.5 RQ3 What are some of the challenges with measuring DevOps metrics and learning from the data?

The main challenges that the teams explicitly mentioned that they had faced when measuring DevOps metrics are:

- Issues with reporting the data in a manner that reflects reality.
- Issues with understanding the collected data.
- Doubts regarding the usefulness, value and significance of the DevOps metrics.

We found that data collection and interpretation can pose challenges, leading to inaccurate reports that fail to accurately reflect the current reality. The inaccuracy in the collected data was in some scenarios an effect of the problems in the reporting scripts used to generate reports. Moreover, the link between issues and deployments is crucial for accurate reporting of deployment frequency. If this link is missing, the reports may not accurately reflect the actual deployment frequency.

Additionally, the lack of understanding of the data was a general problem for the teams. The accuracy of the data was affected because the deployment of backend services and customer-facing applications were mixed. However, since the study was focused on the deployments that directly reach the end users, the mixed data was not relevant to the research and caused confusion.

7. Discussion

The results from the case study highlighted the importance of understanding the technical and organisational challenges that software teams face when trying to adopt a DevOps approach. The findings suggested that these challenges can have a significant impact on software delivery performance, such as deployment frequency and lead time for change. Although the study was limited to one organisation, the findings are similar to what we observed in the State of DevOps report [2].

The State of DevOps report [2] further states that teams that rated higher on continuous delivery are more likely to have a higher frequency of deploying code to production and shorter lead time for changes, which can be supported by our findings as we noted that Team 1 who had the best scores of all the three teams were also the team that had the most stable release process of the three teams with a regular schedule of releasing every week and a process for testing.

Through our research, we gained insights that measuring the four key metrics (lead time, deployment frequency, time to restore service, and change failure rate)[2] alone may not provide a comprehensive understanding of a team's throughput. It became evident that other factors need to be taken into account to obtain a more accurate assessment.

Additionally, we observed that the metrics collected from the investigated software organisation closely aligned with the metrics presented in the State of DevOps report [2]. This finding indicates that the metrics commonly used in computer science are consistent and applicable to different organisations, reinforcing their validity and usefulness in assessing software development and deployment processes.

Furthermore, a comparison with other related work reveals that our findings align closely with the results reported in the existing literature. Our findings share similarities with the findings of Sallin et. al. [1] in several aspects. Like Sallin et al, we found that all teams worked in agile practice. The inherent flexibility of agile methodologies allows teams to adapt and perform optimally in diverse contexts. Additionally, we identified the importance of comprehensive documentation and efficient communication as crucial factors for achieving positive outcomes. One of our findings indicated that a well-defined release schedule was a key factor for the lead time for change and deployment frequency.

Sallin et al. concluded that the metric measurements are generally considered to be valuable, whereas in our case study, we can confirm that many teams were, in general, interested in learning more about the value of the DevOps metrics, while some teams were not completely convinced of its value and usefulness. Sallin et al. also mentioned that software teams, in general, realise that the four key metrics do not cover all aspects which are considered as important for quality, speed and DevOps in general, which we can also confirm in our case study, since we learned that at least one team believed that the metrics don't reflect the overall software delivery performance of a team due to it being measured on terms that do not match their reality, and the teams are aware that the bare measurement of DevOps metrics is not sufficient for improving software delivery performance [1].

Furthermore, Sallin et al. mentioned that some software practitioners believe that a team has to have already the right mindset to get value from the measurement, which can be confirmed by our findings in the case study where at least one practitioner believed that if a team does not work in a structured agile way with a DevOps mentality, then the metrics will not be of much value [1].

We also discovered that none of the teams in our case study paid active attention to the DORA metrics that they had collected. The absence of knowledge and, in many cases, lack of interest in teams disregarding their metrics, inhibits them from gaining a comprehensive understanding of their strengths and weaknesses.

One common reason that all respondents cited was a lack of understanding of the data and how to use it effectively. It may also be possible that the teams do not see the value of collecting and analysing this data, or that the generated reports are not providing useful insights.

Another reason for this could be that the teams are overwhelmed with their daily work and do not have the time to invest in analysing the data. In such cases, we suggest that organisations should create a culture of continuous improvement where reviewing and analysing DORA metrics is an integral part of the software development process.

To encourage the teams to pay more attention to their DORA metrics, it may be useful to provide training and support in data analysis, make the reports more accessible and understandable, and integrate the metrics into the team's daily workflow. This way, the team can gain valuable insights into their processes, identify areas of improvement, and continuously optimise their software delivery performance.

Sallin et. al. [1] arrived at a similar finding, as they highlighted the significance of giving attention to the metrics. They noted that the measurement of these metrics is often conducted manually and through surveys, resulting in a limited number of data points. This manual and survey-based approach to measurement restricts the granularity and accuracy of the collected data. It relies on self-reported information, which can introduce biases and inaccuracies. Moreover, the limited data of data points may not provide a comprehensive and representative view of the actual performance of the teams. In other words, Sallin to concluded that it is crucial for teams to actively review tier data and metrics to optimise their performance and achieve better long-term outcomes.

Another interesting aspect to consider is the impact of company culture on software delivery performance. Our case study provided insights into the importance of organisational factors, such as collaboration, communication, and management support, in achieving successful outcomes in software development. These factors are often deeply embedded in a company's culture, and changing them can be challenging.

While it may be difficult to generalise our findings due to the unique factors that can impact software delivery performance for each organisation, the study's results are still valuable for the software industry as a whole. It is crucial to raise awareness of the challenges faced by software teams, as only then can solutions be designed to overcome these obstacles.

Furthermore, the factors that can impact software delivery performance can vary a lot depending on the technical setup of a software team, the organisational factors that affect them, business requirements and so on. It is therefore not easy to generalise this type of findings, but the findings are nevertheless interesting for many software businesses in today's industry. This finding can be backed up by the State of DevOps report [2] which also mentions that the effects on software delivery performance depend on the broader team context such as a team's processes, strengths, constraints, goals and work environment.

Even though we believe that three software teams are not that sufficient to be able to give an overall overview of the process of measuring software delivery performance, we still think that the findings are useful for other organisations aiming to start measuring deployment frequency and lead time for change.

Furthermore, the research questions defined in Chapter 1 have been answered, and multiple factors that impact software delivery performance and more specifically deployment frequency and lead time for change have been identified. The challenges faced by software teams when measuring this performance have also been identified, which can range from data collection to data interpretation. Although the study was limited to three software teams, the findings are still useful for organisations looking to improve their software delivery performance.

8. Conclusions and future work

This study focused on investigating the challenges of measuring software delivery performance. We defined the following research questions: ***RQ1*** *Is there a significant difference in the DevOps speed metrics between teams?* ***RQ2*** *What are some of the causes for fluctuations in DevOps speed metrics?* ***RQ3*** *What are some of the challenges with measuring DevOps metrics and learning from the data?*

To answer RQ1, we performed a statistical analysis of collected metrics from three software teams and concluded that there was a significant difference in both deployment frequency and lead time for change. We answered RQ2 by stating that some of the causes that can affect deployment frequency and lead time for change are the size and structure of the development team, the complexity of the codebase, and the efficiency of the testing and release process. Similarly, the lead time for change can be influenced by the testing process, team methodology, pull request handling, team size, and workload management. To optimise these metrics, it is important for software development teams to regularly review and improve their processes while balancing efficiency and code quality. We answered RQ3 by stating that some of the main challenges that teams faced were issues with reporting the data, understanding the data as well as doubting the value and usefulness of the data.

Throughout the project, we have found interesting insights about the way that a software organisation collects data regarding software delivery performance. By looking at existing data from the software teams as well as interviewing software practitioners that work in some of the teams, we further took note of the problems that the organisation faces in the path of working in a data-driven way.

Additionally, we found that there were many challenges with collecting data about software delivery in an accurate manner that reflects reality. We believe that the results are relevant for the software industry in general as they highlight interesting challenges that could be further investigated. For example, a recurring problem was that of data reporting and which can sometimes lack accuracy depending on what data is being reported, and in what context, environment. Another recurring problem was the problem of data being dependent on factors that the software teams can not control, such as external translators that need to translate application texts before the version can be released to the different countries. The teams also faced pure technical challenges such as having to deal with a complex setup and do manual work for releasing and testing as a result of that which prevented them from continuous delivery.

We learned that working in a data-driven manner requires much active studying, effort and regular follow-ups on the collected data.

We believe that we have bridged the knowledge gap to some extent, as there were no concrete case studies made on specific software organisations concerning DORA metrics, and this case study gave some insights into the problems that a software organisation could run into when adopting DORA metric collection.

We believe that our case study was heavily focused on identifying the bottlenecks, problems and challenges that the software organisation was facing, and not so much on developing solutions and strategies to make the organisation even more data-driven and opt for better releasing and testing processes.

We are also aware that the data we used to perform our study is data that has been collected over some time and we can therefore not control it. If we had time, we would try to adapt the metrics to the way teams work to get more accurate data.

A suggestion of future work in this regard would be to work on developing a model for continuous deployment and delivery and a set of routines that teams can adopt to actively optimise and automate their data collections as well as study and learn from their data to improve their ways of working and the outcome of their software deliveries.

Another suggestion for future work would be to investigate further how the metrics should be measured depending on a team's ways of working in terms of software development. For example, we found that teams that don't work in a structured, typical Scrum manner but in a more Kanban-like way may not benefit much from measuring lead time for change as suggested in the general DORA guidelines. It would be interesting to analyse if there are other potential ways to measure this type of metric to see how a team performs in terms of software delivery.

Since we chose to limit our focus to deployment frequency rather than deployment volume, another suggestion for future work is to expand on this research by examining the relationship between deployment frequency and other metrics, such as deployment volume and change failure rate. By exploring these additional factors, a more comprehensive understanding of the software development process and its efficiency could be gained.

It would also be interesting to investigate if there are other metrics apart from the four defined key metrics that can be valuable to measure to further learn about a software team's delivery performance.

The impact of documentation quality on the DevOps metrics can also be worth investigating further, as the State of DevOps reports from 2021 [21] and 2022 [2] gave contradicting results and can therefore be an interesting area of research.

It can also be worth expanding the investigation on the usefulness and value of the four defined metrics from the work of Sallin et al. [1] with new surveys asking a broader population of software practitioners. One area of investigation is the measurements of the technical practices mentioned in the State of DevOps report from 2021 like loosely coupled architecture, trunk-based development, continuous testing and the use of open-source technologies [21].

Research on ways of working and how that can impact these metrics could also be of interest, like analysing how the size of user stories may impact how quickly changes reach production, and if it is more efficient to either break down stories into smaller pieces and deploy more often or group them and deploy them in one bigger chunk.

To improve the generalisability of the findings, future research should investigate the challenges faced by software teams in different organisations and contexts.

9. References

[1] M. Sallin, M. Kropp, C. Anslow, J. W. Quilty, and A. Meier, ‘Measuring software delivery performance using the four key metrics of DevOps, in *Agile Processes in Software Engineering and Extreme Programming: 22nd International Conference on Agile Software Development, XP 2021, Virtual Event, June 14--18, 2021, Proceedings*, 2021, pp. 103–119.

[2] ‘2022 state of DevOps report’, *Google Cloud*. https://cloud.google.com/devops/state-of-devops (accessed Apr. 06, 2023).

[3] H. van Merode, *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. Apress, 2023.

[4] N. Forsgren, J. Humble, and G. Kim, *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution, 2018.

[5] Atlassian, ‘Atlassian survey 2020 - DevOps trends’, *Atlassian*. https://www.atlassian.com/whitepapers/devops-survey-2020 (accessed May 22, 2023).

[6] ‘16th State of Agile Report | resource center | digital.Ai’, Apr. 2023, Accessed: May 22, 2023. [Online]. Available: https://digital.ai/resource-center/analyst-reports/state-of-agile-report/

[7] ‘Technology Radar’, *Thoughtworks*. https://www.thoughtworks.com/radar (accessed May 22, 2023).

[8] P. Runeson and M. Höst, ‘Guidelines for conducting and reporting case study research in software engineering’, *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, Apr. 2009.

[9] A. Debbiche, M. Dienér, and R. Berntsson Svensson, ‘Challenges When Adopting Continuous Integration: A Case Study’, in *Product-Focused Software Process Improvement*, 2014, pp. 17–32.

[10] V. Ionzon and S. Jägstrand, ‘A Company Case Study: Examining criteria in cross-platform evaluation frameworks’, 2022. Accessed: Mar. 24, 2023. [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1673341

[11] Y. Gebrewold, *dora_metrics_interviews*. Github. Accessed: May 22, 2023. [Online]. Available: https://github.com/Yamo93/dora_metrics_interviews

[12] H. W. Thompson, R. Mera, and C. Prasad, ‘The Analysis of Variance (ANOVA)’, *Nutr. Neurosci.*, vol. 2, no. 1, pp. 43–55, 1999.

[13] M. Shahin, M. Ali Babar, and L. Zhu, ‘Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices’, *IEEE Access*, vol. 5, pp. 3909–3943, 2017.

[14] N. Forsgren, M. C. Tremblay, D. VanderMeer, and J. Humble, ‘DORA Platform: DevOps Assessment and Benchmarking’, in *Designing the Digital Transformation*, 2017, pp. 436–440.

[15] E. Wolff, *A Practical Guide to Continuous Delivery*. Addison-Wesley Professional, 2017.

[16] D. G. Portman, ‘Use Four Keys metrics like change failure rate to measure your DevOps performance’, *Google Cloud Blog*, Sep. 22, 2020. https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance (accessed Apr. 08, 2023).

[17] S. Bick, A. Scheerer, and K. Spohrer, ‘Inter-team coordination in large agile software development settings: Five ways of practicing agile at scale’, *Proceedings of the Scientific Workshop*, 2016, [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2962695.2962699

[18] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, ‘DevOps’, *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016.

[19] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*. Addison-Wesley Professional, 2015.

[20] Zach, ‘Complete Guide: How to Interpret ANOVA Results in Excel’, *Statology*, Nov. 30, 2021. https://www.statology.org/interpret-anova-results-in-excel/ (accessed May 13, 2023).

[21] D. Smith, ‘Announcing DORA 2021 Accelerate State of DevOps report’, *Google Cloud Blog*, Sep. 21, 2021. https://cloud.google.com/blog/products/devops-sre/announcing-dora-2021-accelerate-state-of-devops-report (accessed May 15, 2023).

A Appendix

A.1 Interview questions

1. Please explain your team structure.
2. Please briefly explain your work process from sprint planning to release.
3. How often do you aim to deploy to production?
4. How quickly do you wish the time to be from code commit to the code running in production (lead time for change)?
5. Do you regularly look at DORA metrics? How do you act accordingly? If not, what may be possible reasons for not looking at it?
6. How do you explain your deployment frequency report? (show report)
7. How do you explain your lead time for change report? (show report)
8. What are factors that may impact your deployment frequency?
9. What are factors that may impact your lead time for change?