INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




DevOps Approach to Measure Product Quality and Developers Efficiency

JOÃO GONÇALO MESQUITA LEITE DE PINHO
Junho de 2024

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

DevOps Approach to Measure Product Quality and Developers' Efficiency

João Gonçalo Pinho

Master of Science, Specialisation Area of Cybersecurity and System Administratiton

Supervisor: Dr. Nuno Pereira

Porto, June 28, 2024

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, June 28, 2024

Abstract

To prosper in today's competitive technology landscape, companies must rapidly release high-quality features, driving the widespread adoption of DevOps practices. DevOps integrates development and operations teams, fostering cross-functional collaboration, enhancing developer efficiency, and accelerating product delivery, thus product quality.

Metrics play a crucial role in the DevOps environment by facilitating informed decision-making and allowing continuous improvement. Effective metric collection helps to address the challenge of maintaining market competitiveness by allowing companies to assess their practices that impact product quality and developer efficiency. However, collecting these metrics presents challenges, including data collection and security issues, as metrics are often dispersed across various tools with different security models, and may include sensitive information.

This project, conducted within a company-specific context, made use of a state of the art open-source tool DevLake that helped centralize data from different tools, automating data collection, and displaying metrics. Contextual factors encountered during the project required thorough research and adaptation. This allowed the collection of 30 metrics, where most of them were tailored to the company's needs, with potential applicability for benchmarking in other organizations. After employing these metrics, their effectiveness was assessed through six case studies.These case studies demonstrated that the metrics used are effective in assessing changes and providing insightful information for data-driven decision-making.

Keywords: DevOps, Metrics, Product Quality, Developers' Efficiency

Resumo

A prática de DevOps, que une equipas de desenvolvimento e operações, promove a colaboração e a integração contínuas, já comprovou que resulta numa maior eficiência e qualidade de produto. Este modelo não atende apenas às necessidades por inovação rápida, mas também facilita a adaptação a mudanças constantes no mercado global. No contexto de DevOps, a recolha e análise de métricas desempenham um papel muito importante. As métricas permitem às empresas avaliar o seu desempenho e comparar o seu progresso com as demais empresas, permitindo assim identificar áreas de melhoria e tomar decisões baseadas em dados. Mas, um dos grandes desafios de recolher métricas numa empresa é o facto da fonte dos dados das mesmas estarem espalhadas entre diversas ferramentas.

Durante este trabalho foi conduzido um estudo sobre do estado da arte onde foram apresentadas 18 capacidades de DevOps essenciais e frequentemente adotadas por empresas de alto desempenho, divididas em aspetos técnicos, processuais e culturais. Nove métricas foram apresentadas como fundamentais para medir tanto o sucesso das práticas DevOps, como a qualidade do produto e eficiência dos desenvolvedores.

Entre as métricas apresentadas, destacam-se as métricas DORA, que demonstraram correlação positiva entre a frequência de implantações e a estabilidade do ambiente de produção. A importância destas métricas destacam-se através da apresentação de dois casos de estudo, destacando-se o da Capital One, onde se demonstrou que com a recolha destas métricas foi possível identificar pontos de melhoria. Através deste conhecimento procederam à melhoria de determinadas práticas DevOps e por meio da monitorização destas métricas conseguiram identificar aumentos significativos na eficiência operacional sem comprometer a qualidade e estabilidade.

Numa abordagem inicial ao problema, feita uma análise da organização relativamente às ferramentas utilizadas, sendo que estas são as principais fontes dos dados das métricas. Tendo conhecimento das ferramentas usadas diariamente pela empresa, a seleção da ferramenta para a recolha dos dados, análise e apresentação de métricas tornou-se mais fácil. A ferramenta selecionada, DevLake, é uma ferramenta de código aberto, que permitiu a automatização da recolha de dados e a criação de dashboards personalizados, facilitando a visualização e interpretação das métricas relevantes para a empresa e para o projeto. No que se refere às métricas que foram criadas, muitas são específicas ao contexto da empresa, mas mesmo assim podem ser úteis para outras que pretendam avaliar o seu desempenho relativamente à qualidade do produto e à eficiência dos desenvolvedores. A criação das métricas exigiu um pensamento crítico e ponderação para garantir que fossem facilmente interpretáveis. Durante a implantação da ferramenta, a segurança esteve presente devido à criticidade dos dados, porque facilmente estes identificam colaboradores e o trabalho dos mesmos.

Por fim, houve a oportunidade de avaliar a utilidade destas métricas através das mudanças ocorridas na organização onde o projeto foi realizado. Esta avaliação resultou em seis casos de estudo que demonstraram que as mudanças recentes foram benéficas para a empresa,

além de confirmar a utilidade das métricas. Destacando um dos casos de estudo, o mesmo demonstrou que as recentes mudanças tomadas pela equipa de gestão impactaram positivamente na eficiência do desenvolvimento e planeamento.

Acknowledgement

First, I express my deepest gratitude to my family, and girlfriend for their unwavering support and stability throughout my academic journey. Their encouragement has been a cornerstone, for which I am profoundly thankful. Additionally, I express heartfelt appreciation to my friends, whose camaraderie has been invaluable.

A special acknowledgment is owed to Emvenci, where I was warmly received and provided with the necessary resources to conduct my thesis, from its inception to the final review. Within the company, I extend gratitude to Alexandre Aniceto, the company supervisor whose invitation made this possible and also to Miguel Bernardes consistently offered assistance, and toVasile Parasca provided invaluable support as a teammate.

I am also grateful to the Instituto Superior de Engenharia do Porto for equipping me with the skills essential to navigate the professional landscape and simplifying my journey into the working world.

Lastly, I extend sincere thanks to my supervisor, Dr. Nuno Pereira, for his unwavering support, guidance, and commitment throughout this process, from our countless meetings to his thorough reviews. His mentorship was instrumental in shaping this thesis.

Contents

List of Figures

List of Tables

Glossary

API	Application Programming Interface.
AWS	Amazon Web Services.
CAB	Change Advisory Board.
CD	Continuous Deployment.
CI	Continuous Integration.
DevOps	DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations (Ops). It emphasizes communication and collaboration, continuous integration, quality assurance, and delivery with automated deployment, utilizing a set of development practices..
DOES	DevOps Enterprise Summit.
DORA	DevOps Research and Assessment.
HTTPS	Hypertext Transfer Protocol Secure.
IaaS	Infrastructure as a Service.
IaC	Infrastructure as Code.
IT	Information Technology.
LDAP	Lightweight Directory Access Protocol.
LISA	Large Installation System Administration Conference.
NIST	National Institute of Standards and Technologies.
PAM	Privileged Access Management.
PII	Personally Identifiable Information.
RBAC	Role-Based Access Control.
SDLC	Software Development Lifecycle.
SDO Performance	Software Delivery and Operational Performance.
SSL	Secure Sockets Layer.
URL	Uniform Resource Locator.

USA	United States of America.
VCS	Version Control System.

Chapter 1

Introduction

Organizations today face fierce competition transcending national boundaries. It is not just about pricing or product quality; it encompasses various aspects such as technology, customer service, branding, and supply chain efficiency. Thus, organizations must focus on promoting and delivering high-quality products and services to remain competitive [1]. Software organizations, in particular, are required to accelerate the release of high-quality features within condensed time frames, which motivated the widespread adoption of practices to automate and integrate the processes between development (dev) and operations (ops) teams, commonly known as DevOps [2]. These practices aim to enable organizations to build, test, and release software faster and more reliably.

A key concept in DevOps is monitoring the software production process to understand performance, identify bottlenecks, and continuously improve processes. Central to this process are the several metrics collected. Effectively using these metrics helps teams make data-driven decisions, track progress, and ensure that their practices align with business objectives. However, collecting metrics about the software production process can present several challenges in data collection and security. For example, the sources of these metrics can be spread throughout the organization in different repositories and systems, which can have different permissions and security models, and the data itself might be sensitive as, for example, contains personally identifiable information about developers. Additionally, the context of the particular organization must be considered when selecting relevant metric and how these can be effectively presented to stakeholders.

This work aims to tackle these challenges and take advantage of a tool to automate data collection from various sources, centralize data, and display metrics conveniently and securely. A critical aspect of the work is to select relevant metrics not only from the state of the art but extract others from discussions with the stakeholders and apply them in the context of the specific organization. After designing and implementing such a tool, it is essential to demonstrate its utility to stakeholders. This work includes a set of case studies that leverage some early data collected to provide several concrete insights.

1.1 Problem

This work addresses the challenges of collecting metrics about an organization's software production process. Today, the hosting organization tracks some metrics by performing manual queries to the issue tracking and project management system. Although this may be effective in some cases, it cannot scale. As the organization grows, more complete and systematic data gathering is needed. Automating the collection of metrics will save time and enhance user-friendliness, help the company understand what may be affecting product

quality and developers' efficiency to make data-driven decisions and evaluate the impact of changes within departments.

1.2 Objectives

To address the problem described in the previous section, the following objectives were defined:

- **Objective 1: Survey practices, metrics, and technologies:** Perform a comprehensive review of the state of the art on pertinent topics to acquire the fundamentals of DevOps practices today, metrics, and technologies employed in gathering data from DevOps tools to distill insights for engineering excellence.
- **Objective 2: Assess Organizational Environment:** Comprehend the tools and practices of the company to identify relevant data sources.
- **Objective 3: Design and implement a solution and collect metrics:** Select relevant metrics in the organization's context, design a data collection architecture, and finally set up and configure the tool to collect data and generate the chosen metrics automatically.
- **Objective 4: Evaluate metrics and derive insights:** After collecting metrics, analyze them to extract valuable insights, with the most relevant findings highlighted as case studies.

1.3 Ethical Considerations

This document was prepared with a commitment to ethical considerations, aiming to uphold academic principles of honesty and integrity. The following section discusses the most pertinent ethical considerations.

Generative artificial intelligence (AI): Paragraphs of this thesis have been rephrased using AI tools, by only using contained and self-validated contributions from these tools. This usage was conducted carefully to ensure that no sensitive information was leaked in the processed text.

Access to sensitive data: This project required access to Personally Identifiable Information (PII), which required the following considerations:

- **Consent:** This was done with consent, and the organization was fully informed about the collection, use, and storage of the information.
- **Data privacy and security:** All information is secure against unauthorized access and disclosure.

Confidentiality and review process: No information was exposed within this document without the prior review of a representative member from the organization.

1.4 Document Structure

This document is organized in seven distinct chapters, which are:

- **Introduction:** This chapter introduces the topic and provides more details of the problem. Additionally, it presents the methodology and objectives to conduct the project and discusses the ethical considerations involved in writing this document.
- **State of The Art:** This chapter describes the details related to the methodology of the conducted research, together with the presentation of the results, providing fundamental knowledge needed to achieve the objectives.
- **Organizational and Tool Analysis:** This chapter describes the tools currently used by the company, states which was the selected tool for gathering data, and presents it.
- **Design:** This chapter describes the architecture of the system and explains how the dashboards are organized. Furthermore, it presents the gathered metrics and discusses their relevance.
- **Implementation:** This practical chapter provides an in-depth explanation of the chosen tool, detailing its setup and the construction of the metrics.
- **Case Studies:** This chapter analyzes the collected metrics, correlating them to present insightful case studies proving the usefulness of the chosen metrics.
- **Conclusion and Future Work:** This closing chapter provides a final overview of the work as a whole and discusses potential directions for future work.

Chapter 2

State of The Art

To gather the foundational knowledge necessary to address the objectives identified in Section 1.2, three research questions were formulated. These questions specifically target important topics relevant to the project. The resulting research conducted in this chapter contribute to **Objective 1**.

The research questions are as follows:

- **RQ1: What are the predominant practices utilized in DevOps?**
  - **Rationale:** Since DevOps highly correlates with product quality and developers' efficiency, understanding what high-performing companies practice within DevOps will, not only, empowered the author to build some knowledge about the practices but also take them into account during the implementation.
- **RQ2: Which metrics are available for assessing DevOps?**
  - **Rationale:** Comprehending the metrics commonly utilized to evaluate DevOps performance is fundamental due to the significant correlation between DevOps and product delivery, automation, and developers' efficiency. However, it is essential not only to identify metrics closely linked to DevOps but also to acknowledge other types of metrics that may prove value assessing product quality and developers' efficiency.
- **RQ3: What tools can be employed to collect metrics effectively?**
  - **Rationale:** Given the nature of automation in DevOps work, gaining insights into the tools that can effectively and automatically collect data to generate the metrics is indispensable.

To address these inquiries, a systematic review of the literature was performed. A systematic literature review is conducted to identify, evaluate and interpret all available research relevant to a particular research question, topic, or area of interest [3]. The search approach will involve querying various scientific databases, including ACM, Science Direct, and Google Scholar. The inclusion of Google Scholar was crucial for comprehensive coverage, as it queries multiple databases, allowing to include references from a wider group of databases.

After adhesion testing of queries, those that retrieved a greater number of relevant scientific studies and research articles were included in Table 2.1.

After numerous queries and a brief analysis of several documents, it became clear that the query retrieved a high volume of documents. Therefore, it was necessary to define inclusion criteria:

Query number	Query used
Q1	"DevOps" AND ("pipeline performance" OR "pipeline") AND ("metrics" OR "kpi" or "measurement") AND since 2010
Q2	"DevOps" AND ("commonly used capacities" OR "most utilized practices" OR "key capabilities" OR "widely adopted features" OR "popular strategies") AND since 2010

Table 2.1: Queries used

- **Filter 1**: Publication Date — Limited to content published after the year 2010, unless it is something widely referenced and fundamental to explain something related to the topic.
- **Filter 2**: Language — The content must be in Portuguese or English to ensure that content is comprehended and accurately analyzed.
- **Filter 3**: Format — Limited to books, articles, thesis, and conferences.
- **Filter 4**: Credibility — Requires a minimum of 5 citations to establish a minimum level of recognition and influence.
- **Filter 5**: Relevance — Content must be related to the topic and discuss at least one metric, one capability, or one tool to gather metrics.

The previously mentioned criteria was fundamental, since it allowed to provide credibility to the document and to understand what the core literature needed to be analyzed. However, in the metrics section and on the tools to gather them, including other sources of information such as tools websites, technical websites, and others, it was vital to have a wider view of the topic. Research failing to meet any of the inclusion criteria was excluded from consideration in this document.

After this introduction regarding the research methodology used to conduct the state of the art review, the results will be presented. This remainder of this chapter is composed by the following sections:

- **DevOps:** Describes what in what involves DevOps, examines its influence on the software development lifecycle, and provides a definition analysis.
- **DevOps Capabilities:** Enumerates the core capabilities identified within DevOps.
- **Metrics:** Presents some metrics that were commonly referenced in the literature review, and presents some case studies involving a group of metrics.
- **Tools:** Evaluates tools that assist in collecting data to generate metrics from various sources.

2.1 DevOps

DevOps is a term that has been a subject of discussion for over a decade, but the implementation of it has showcased a range of advantages. The advantages range from the increase of organizational Information Technology (IT) performance and productivity, cost reduction across the Software Development Lifecycle (SDLC), enhanced operational efficacy and efficiency, elevated quality of software products to the improved business alignment between development and operations teams, but still lacks a universally accepted definition [4, 5].

A study conducted by Jabbari et al. [6], which aimed to explore how DevOps it is defined in peer-reviewed literature, concluded the following definition: "DevOps is a development methodology aimed at bridging the gap between Development (Dev) and Operations (Ops). It emphasizes communication and collaboration, continuous integration, quality assurance, and delivery with automated deployment, utilizing a set of development practices.". The definition not only captures the essence of DevOps but also showcases its holistic nature and still considers the various phases integral to the development and deployment lifecycle.

DevOps enhances the SDLC by introducing a shift in mindset, significantly enhancing efficiency and communication between development and operations teams [7].

The main improvements on the SDLC are presented below:

- **Collaborative Development and Continuous Integration:** Collaborative development brings together the business, development, and quality assurance teams to deliver fast, innovative, and quality changes to software in the hands [8].
- **Automation:** DevOps technological capabilities are responsible to automate different trivial and non-trivial tasks, seeking to eliminate human action from all repetitive processes, thus reducing the human error probability and delivery time and increasing quality and security [8].
- **Continuous Testing:** Developers are required to write unit test cases to integrate their software frequently. These tests are executed automatically in the continuous integration pipeline, explained in more detail in a later section [8].
- **Continuous Monitoring:** Automates and optimizes the ability to monitor and manage the performance and availability of applications and infrastructure continuously. Logging is demanded to effectively perform a deep problem analysis to find the root cause and provide quality feedback to the development teams [8].

Concluding that DevOps is more than a practice, is a mindset that fosters a continuous improvement environment. Furthermore, the absence of a singular, universally accepted definition can be attributed to the continuously evolving and diverse nature of DevOps.

2.2 DevOps Capabilities

With the fast-paced investigation of the DevOps field, numerous articles highlight various capabilities. DevOps capabilities can be defined as a variety of engineering processes supported by cultural and technological enablers. These capabilities define the procedures that an organization should be able to carry out, allowing a fluent, flexible, and efficient way of working [9]. With such variety of capabilities associated with DevOps, it is hard to define what are the ones defined as core. However, a particular book, widely referenced, stands out for delineating the essential DevOps capabilities – "Accelerate: Building and Scaling High Performing Technology Organizations" [10]. This book relates the research of five years in the DevOps field by assessing multiple companies worldwide and its authors experience. Furthermore, an article [11] that conducted an in-depth analysis of these capabilities, identified those classified as core that took into consideration the aforementioned book and other research. Upon comparing both analyses, it becomes evident that they highlight the same capabilities, even though some with different names. This alignment facilitated the organization of the following sections.

The capabilities are separated into three main ones, technical, process, and cultural.

2.2.1 Technical Capabilities

DevOps technical capabilities are related to the specific skills, tools, and practices that individuals and teams within an DevOps environment possess or adopt in order to effectively implement DevOps principles. These principles enable the collaboration between development and operations teams, as well as the automation of processes throughout the SDLC [10].

The capabilities are logically organized from foundational concepts to more advanced practices.

Version Control

Version Control System (VCS) is a system that facilitates collaboration on file access, creation, updates, and deletion across teams and organizations [12]. These types of systems are closely tied to enabling efficiency and productivity [11]. Some examples of these types of systems are Git and Subversion.

Test Automation

Automated tests minimize the occurrence of defects in a software system, these tests must guarantee the correct functionality not only of individual services or packages but also the harmonious collaboration of individual components as an integral and coherent whole [13]. Reliable automated tests ensure software quality, increasing the team's confidence in its readiness for release. Once automated tests are in place, it is important to run them regularly. Ideally, when someone commits a change, the software should build and run a set of fast and automated tests to quickly assess the impact of the changes [10, 11].

Test Data Management

To ensure that the automated tests validate realistic scenarios, it is fundamental to supply the tests with production-like data. This requirement extends to various types of tests, including both automated and manual ones. Given the importance of test data in automated testing, it is essential to adopt effective practices, such as having adequate data to run the test suit, acquiring necessary data on demand, conditioning test data in the pipeline, and ensuring that data does not constrain the number of tests to be executed. Teams should minimize, whenever possible, the amount of test data needed to run automated tests [10, 11].

Code Maintainability

In today's technology landscape, complex systems demand extensive codebases. The capability of teams to effectively maintain their code stands out as an important technical practice contributing significantly to the success of continuous delivery. When achieving an effective code maintenance ability, team members will effortlessly locate examples, reuse code from others, and modify team-maintained code as necessary. Furthermore, incorporating new dependencies or migrating to updated versions becomes a smoother process for the team, enhancing stability and minimizing disruptions in the code [14]. In essence, ensuring code maintainability translates to making the codebase easily comprehensible and facilitating uncomplicated modifications.

Database Change Management

Database Change Management is a very simple yet highly effective solution related to managing changes of the databases, including both structure (tables, columns, and relationships) and the data itself. These changes should be stored as scripts in version control, and changes to these files must be managed the same way that the production application changes are managed. When companies follow these practices, database changes do not slow down or cause problems when performing code deployments [15, 16].

Trunk-based Development

Trunk-based development is a practice that has proven to contribute to better software delivery performance, but developers accustomed to Git-Flow are still skeptical about it. In this version-control branching model, developers collaborate in a single branch known as 'trunk'[1], resisting the urge to create additional long-lived development branches to avoid merge complications and build disruptions [17, 18].

For a more lucid comprehension of this approach, let's compare it with an alternative branching model.

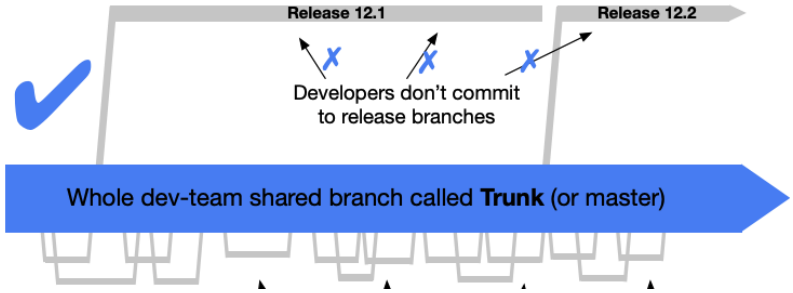

Figure 2.1: Trunk-based development example[17]

As illustrated in Figure 2.1, beneath the blue arrow denoting the main branch, it is possible to observe short-lived feature branches. Each of these branches, typically managed by a single person over a few days at most, undergoes a pull request style code review and build and test automation process before being "integrated" (merged) into the main branch.

[1] 'trunk' was used on Subversion, 'main' was adopted by the Git community in 2020 (Referred to as 'master' previously, but this terminology was revised due to its unsavory connotation)

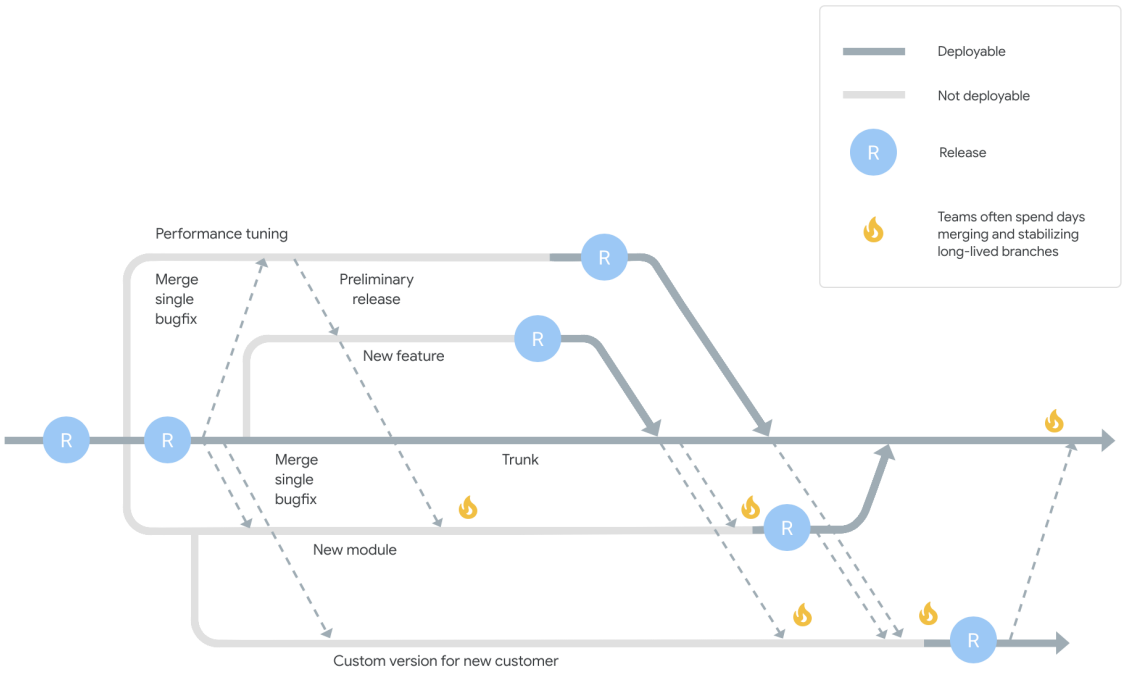

Figure 2.2: Non-trunk-based development example[19]

In Figure 2.2 it is depicted the Git-Flow branching strategy, where developers make changes to long-lived branches, leading to more extensive and intricate merging procedures in contrast to trunk-based development, as demonstrated in Figure 2.1 and previously explained. This approach also requires additional supplementary efforts for stabilization, often involving periods of "code lock" or "code freeze", leading to the creation of additional long-lived branches to ensure software functionality. Large merges lead to the known "merge hell" and frequently introduce bugs or regressions, as a result, the post-merge code needs to be tested thoroughly and increases the frequency of bug fixes.

Continuous Integration

Continuous Integration (CI) is a development practice in which team members integrate and merge development work frequently. This practice comprises automated software building and testing resulting in shorter and more frequent release cycles improving software quality and reliability, increasing team productivity, and reducing the cost of ongoing software development and maintenance [20, 21].

As Rossel states in his book [22], "Continuous Integration is ensuring that the software is in a deployable state all the time", in other words, the code compiles and the quality of code is reasonably good.

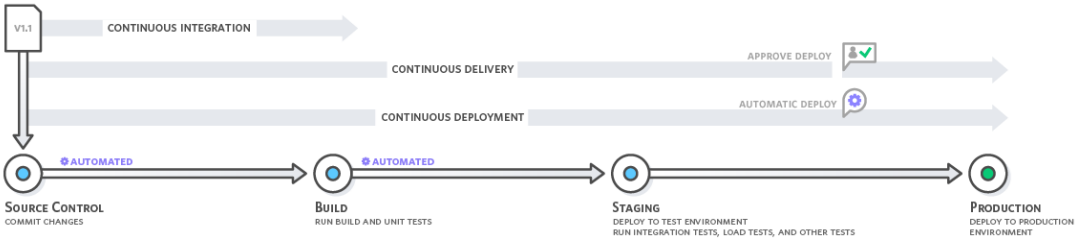

Figure 2.3: Continuous integration, delivery, and deployment [23]

Amazon Web Services (AWS) illustration, depicted in Figure 2.3 perfectly demonstrates how does CI work, where it only fetches the code from source control triggered by a commit and then builds and run the automated tests against the changes.

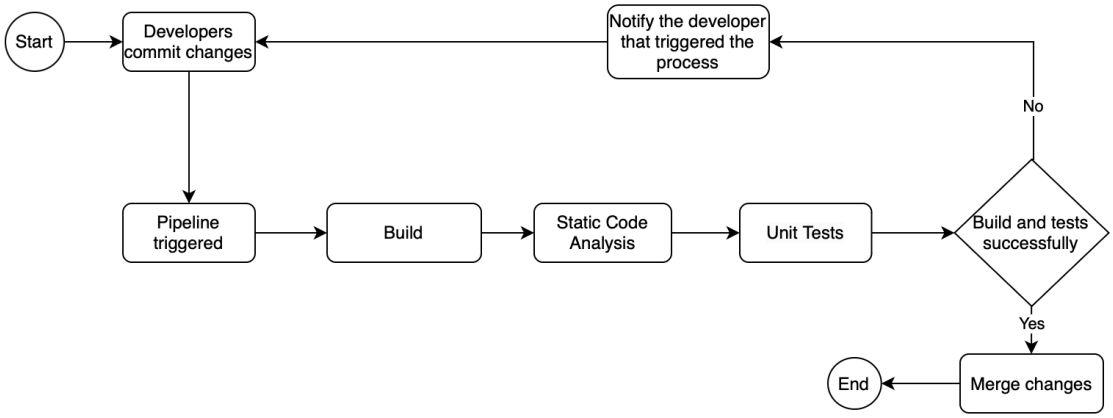

Figure 2.4: Continuous Integration Diagram

A more detailed CI process is exemplified in the flow diagram featured in Figure 2.4. As elucidated in the earlier definition, this procedure involves the automated activation of a pipeline that retrieves the code from the version control tool. This code undergoes a building, static code analysis, and testing to evaluate the effects of newly introduced changes. Upon receiving positive test results, the changes are integrated with the main code. Otherwise, in the event of negative test outcomes, the developer receives a notification, enabling a review of outputs and the resolution of identified problems.

Continuous Delivery

Continuous Delivery, which does not have any acronym otherwise would conflict with Continuous Deployment (CD), involves the steps of CI with the addition of the capability of automatically deploying it to a production-like environment at any time, but only gets deployed to a production environment with a manual approval. To achieve this, the software should be in a deliverable state, meaning that it should pass all the steps of the CI process. This practice allows a rapid response to client requests in a more controlled manner [22].

The efficacy of Continuous Delivery is closely tied to the outcomes of the CI process. If the CI results indicate failure, the software is not considered deliverable, preventing deployment to both staging and production environments. This interdependency showcases the important role of CI in maintaining the integrity and reliability of the software throughout the pipeline.

Figure 2.3, perfectly demonstrates that it depends on CI to then stage and eventually deploy to production upon a manual approve is accepted.

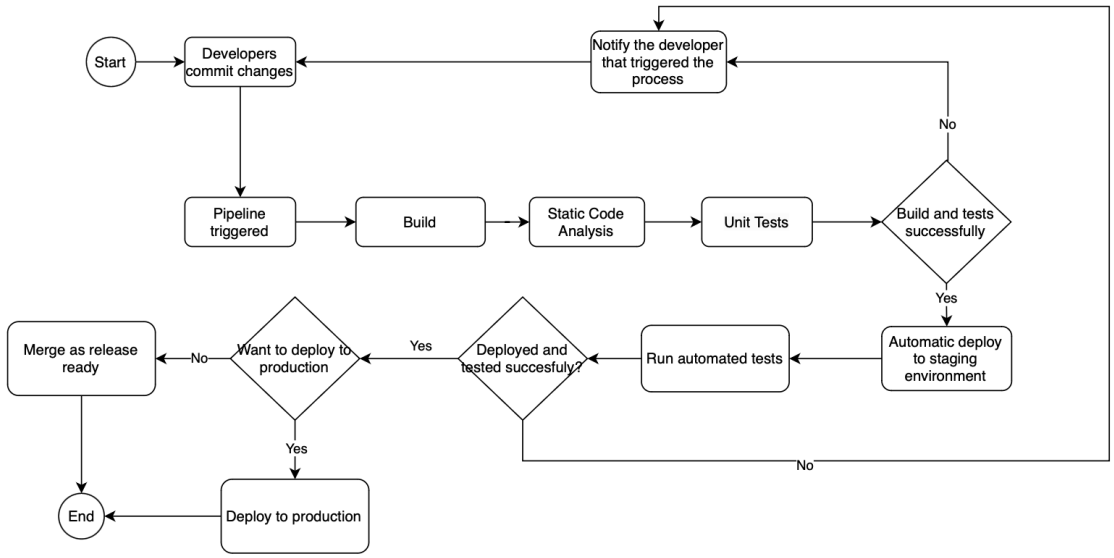

Figure 2.5: Continuous Delivery Diagram

A more detailed example is shown in Figure 2.5, the process extends the CI steps by automatically deploying to a staging environment, enabling the execution of more intricate automated tests. If the deployment and tests in the staging environment encounter problems, the developer who triggered the pipeline is notified to resolve them. Otherwise, a manual step to deploy to the production environment arises, if accepted the automated production deployment begins; otherwise, the code is merged with the main codebase.

Continuous Deployment

Achieving to implement CD it is important to know that attaining the two previous stages in a solid state is mandatory. As aforementioned, in Continuous Delivery, every successful check of the source code triggers a deployment to a production-like environment. Fully automating the deployment to production as soon as the code meets specified criteria can potentially accelerate problem detection and resolution. By deploying in smaller batches, problems surface more quickly, enabling faster remediation. This approach not only enhances the speed of identifying and resolving problems but also contributes to a more smooth and efficient deployment workflow [22].

Figure 2.3 perfectly explains the differences between the Continuous Delivery and CD, where CD requires no manual intervention in order to changes deploy to production.

To provide a more detailed example, the process of continuous deployment is illustrated in Figure 2.6.

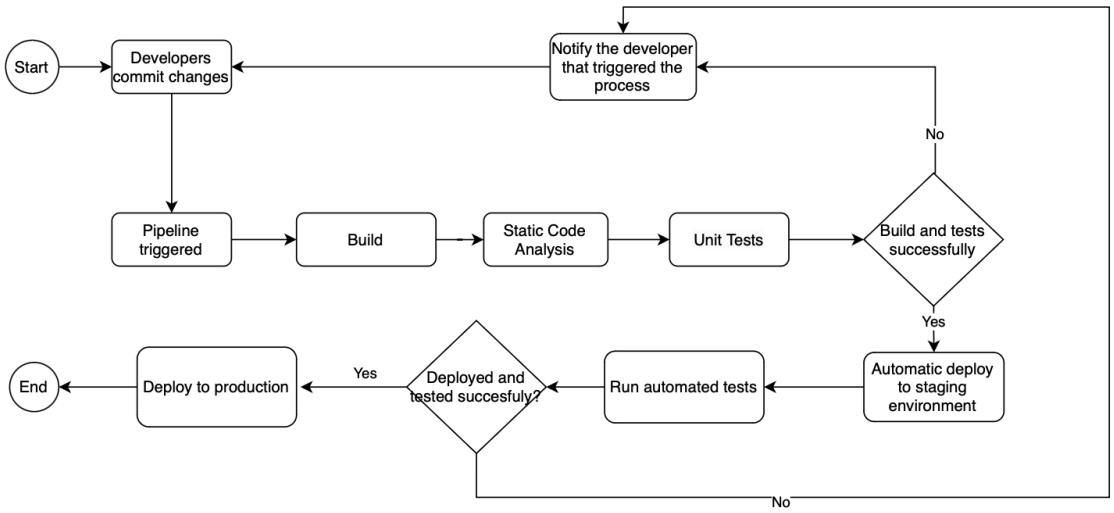

Figure 2.6: Continuous Deployment Diagram

The previous Figure 2.6 resembles the continuous delivery process, with a key distinction: there is no manual step for deploying to production. Every code change intended for the main branch undergoes the triggered pipeline, and if all steps are successful, it is automatically deployed to production.

Monitoring and Observability

Comprehensive monitoring and observability solutions are one of the practices that positively contribute to the previously mentioned continuous delivery [24].

Monitoring plays an important role as it enables teams to comprehend the real-time condition of their systems. This involves the systematic collection of predetermined metrics or logs through the utilization of tools, software, or technical solutions [24]. Beyond the traditional focus on understanding the system stack at various levels, monitoring extends to include insights from the tools employed. Acknowledging that these tools possess both failure potential and opportunities for performance enhancement is crucial for comprehensive system oversight [25]. Consequently, the analysis of both perspectives not only enriches the understanding of system dynamics but also emphasizes the need to evaluate and optimize the performance and reliability of the monitoring tools themselves.

Observability, on the other hand, enables teams to acquire insights into the internal operations of a system, particularly in complex and dynamic environments where predicting all metrics in advance may not be possible. This capability allows teams to actively debug their systems and gain a comprehensive understanding of their behavior [24].

Monitoring metrics must be checked often to understand the system's behavior. Establishing and configuring thresholds becomes crucial to ensure timely alerts for DevOps teams, allowing fast actions in case of abnormal activities within the system. Recognized as a fundamental capability within DevOps, the broad-reaching impact of monitoring and observability on other practices underscores its classification as a core component. When it comes to observability, as previously stated, the implementation of it allows proactive issue detection and improvement.

Flexible Infrastructure

This capability is frequently associated with Cloud Computing, especially the Infrastructure as a Service (IaaS) model, which enables the ability to provision computing resources, on-demand, via code. This is more commonly referenced as Infrastructure as Code (IaC) [26]. The United States National Institute of Standards and Technologies (NIST) [27] defines five essential characteristics of cloud computing that are also the key points of this capability:

- **On-Demand Self-Service**: The user can provision computing capabilities according to his needs.
- **Broad Network Access**: The provided capabilities can be accessed from various platforms such as mobile phones, tablets, laptops, and workstations.
- **Resource Pooling**: The provider resources are pooled in a multi-tenant model, where both physical and virtual resources are dynamically assigned on-demand. The customer has the flexibility to designate the location at a more abstract level, such as country, state, or data center.
- **Rapid elasticity**: Capabilities can be elastically provisioned on demand, on other words, can either increase or decrease the computing capabilities as needed. This creates the impression of limitless availability and ability to acquire in any quantity at any given time.
- **Measured service**: By default cloud systems control, optimize, and report resource usage based on the type of service such as storage, processing, bandwidth, and active user accounts.

In sum, the key aspect of this capability lies in its ability to dynamically adjust computational resources, enabling cost-effective scalability by increasing or decreasing resources as needed.

Empowering Teams to Choose Tools

The term for this capability is self-explanatory; but, in other words, it implies that companies should permit teams to select tools according to their needs [10]. In typical organizational settings, engineers are usually constrained to the use of tools and frameworks from a previous approved list for the following reasons:

- Reduce the complexity of the environment.
- Ensure that the team possesses the necessary skills to manage the technology throughout its lifecycle.
- Increasing purchase power with vendors.
- Ensure compliance with licenses for every used technology.

However, this lack of flexibility prevents teams from choosing the most suitable tools for their specific requirements, limiting their creativity and problem-solving abilities. The engineers who develop and deliver software and manage complex infrastructures select the tools based on what is best for completing their work and supporting their users. Giving the freedom to teams to choose the tools is a way better approach than forcing them to use preselected tools for the stakeholder's convenience. This capability not only enhances software delivery performance but also fosters a sense of empowerment among the team members, leading to increased job satisfaction and, consequently, improved organizational performance [10].

As Dr. Forsgren et al. highlighted [10], "When the tools provided make life easier for the engineers who use them, they will adopt them of their own free will."

2.2.2 Process Capabilities

Process capabilities refer to the set of practices, procedures, and methodologies employed to optimize the software development and delivery lifecycle [10].

The following processes are logically organized, from the foundational concept to the most advanced practice.

Documentation Quality

Documentation is a fundamental aspect of software development, since it enables current and future employees to understand the system. Inadequate documentation is akin to its absence, showcasing the need for clear guidelines on internal documentation to maintain consistency and quality within the company [28].

Efficient Change Approval

Every organization follows a process to update its production environment. It varies from a simple approach, often seen in startups where developers review each other's code before production changes, to a more complex one in larger companies. In the latter, changes undergo review by a Change Advisory Board (CAB), a team responsible for approving external changes. This, combined with team-level code reviews, can extend the timeline to weeks, impacting delivery speed [10]. Implementing change approvals through peer review during development, along with technical tools like continuous integration monitoring, and observability, fostering the early detection and prevention of problematic changes [10, 11].

Shift-left on Security

Shifting left on security means integrating security objectives into everyday work practices. Often, individuals say that engaging in "security activities slow them down" less attention to this aspect or, in extreme cases, put it as a secondary consideration. It is only when a security issue arises in production that the true significance of incorporating security practices becomes apparent. The objective is to incorporate activities that address both quality and security without compromising the swift pace of delivery [29].

Constructing a secure environment is undoubtedly challenging, consistently implementing security measures in the initial stages of the SDLC may initially seem time-consuming. However, in the long run, the investment proves to be invaluable.

Loosely Coupled Architectures

A loosely coupled architecture is an architectural design approach that focuses on minimizing dependencies between different components. This capability aims to achieve a high degree of isolation, fostering independence among team members and facilitating the concurrent development with the overarching goal of prioritizing deployability, testability, supportability, and modifiability [30, 31]. Aligned with DevOps principles, this capability shows as one of the most significant contributors to DevOps success [32].

2.2.3 Cultural Capabilities

The cultural capabilities, as defined by Westrum, is "the organization's pattern of response to the problems and opportunities it encounters" [33].

Well-being

Well-being exhibits the individuals' happiness and job satisfaction. Well-being predicts organizational performance and employees' job tenure [34].

Generative Organizational Culture

It is not new that a good organizational culture is a factor that optimizes information flow and enhances organizational performance in technology. This was based on Dr. Ron Westrum's research that found that exists three typologies of organizational cultures, presented in Table 2.2.

Pathological	**Bureaucratic**	**Generative**
Power oriented	Rule oriented	Performance oriented
Low cooperation	Modest cooperation	High cooperation
Messengers "shot"	Messengers neglected	Messengers trained
Responsibilities shirked	Narrow responsibilities	Risks are shared
Bridging discouraged	Bridging tolerated	Bridging encouraged
Failure leads to scapegoating	Failure leads to justice	Failure leads to inquiry
Novelty crushed	Novelty leads to problems	Novelty implemented

Table 2.2: Westrum Organizational Cultures [33]

In Westrum's work, he noted that a generative culture influences the way information flows through an organization, providing three characteristics of good information:

- The information provides answers to the questions that the receiver needs to answer.
- It is timely.
- It is presented in such a way that the receiver can use it effectively.

Similar to Dr. Westrum's findings, DORA showed that a high-trust, generative culture predicts software delivery and organization performance in technology [10, 33].

In conclusion, having a generative culture in an organization is an important factor when implementing DevOps, it is so important that on the previously given DevOps definition, one of the parts is "It emphasizes communication and collaboration [. . .]" allowing to classify this as an important DevOps capability.

2.3 Metrics

Measuring performance in the domain of software has always been a hard procedure — unlike manufacturing, inventory is invisible [10]. Metrics refer to quantitative measures of the degree to which a system, component, or process possesses a given attribute [35]. Metrics objectively help teams distinguish between improvements and unproductive changes [36]. However, it is important to note that metrics can degrade performance when used as

targets, as Goodhart's law says "When a measure becomes a target, it ceases to be a good measure". In this section, is discussed isolated metrics and provide a more in-depth view of the metrics identified by the DORA.

2.3.1 Isolated Metrics

Essentially, the metrics included in this section were collected from diverse sources, and the ones chosen are the ones that align most closely with the objectives. Table 2.3 displays a variety of metrics, providing their names, definitions, relevance, and calculation methods.

Name	Definition	Relevance	Calculate
Build Failure Rate	Proportion of failed builds compared to all builds [37]	Allows to understand how frequently the builds fail and which pipeline fails the most [38]	Number of failed builds divided by the total number of builds multiplied by 100
Build Duration	Time that the successful pipeline jobs take to build [37]	Overall performance and efficiency of the building process [39]	Sum of all times of successful builds divided by the number of builds
Lines of Code	Total of lines of the source code	Quick impression of the size and complexity of the software, and it is used to evaluate code quality [40]	Number of lines of code
Number of Code Duplication	Number of duplicated snippets of code	Provides a quick overview of the maintainability, reliability, and readability of the codebase [41, 42]	Counting the number of duplicated code snippets
Pull Request Frequency	Number of pull requests that merged to the main branch	Provides a quick feedback on developers velocity and code quality [43]	Number of merged pull requests divided by the number of all pull requests in a specific range
Test Coverage	The amount of code that is covered by the unit tests	Enables assessment of untested code, prioritizing testing to enhance overall coverage and reduce the risk of undetected defects in software [44]	Subtract the uncovered lines from the total number of lines to cover, then divide it by the total number of lines to cover and multiply by 100.
Defect Escape Rate	Number of defects that were not detected in a testing environment [37]	Underlines the importance of the importance of testing and monitoring to detect any defects [45]	Number of defects found in production divided by the total number of defects found and multiply by 100
Code Debt	Effort to fix all code smells	Provides insights about areas needing improvement, and minimizing future challenges in code maintenance [46]	Total code debt divided by the number of working hours
Service Availability and Uptime	Percentage of service availability on a time period [37]	Quick view of how operational the service is [47]	Divide the number of hours your service is up and running per year per 8,760 hours (total hours in a year) and multiply by 100

Table 2.3: Isolated Metrics

2.3.2 DORA Metrics

DevOps Research and Assessment (DORA) which is described by its creators as the largest and longest-running research program in this category. It looks forward to understanding the capabilities that influence software delivery and operations performance and helping teams to apply them, consequently leading to better organizational performance [48]. In other words, as stated by Pietrantuono et al. [49], DORA can also be described as a "[. . .] research that provides a comprehensive and up-to-date analysis of observed trends, meant as a reference by companies for benchmarking".

These metrics are mainly referred to as DORA metrics or in some literature "Accelerate metrics", "Four Key Metrics" or even "Google Metrics" since they bought DORA. The "DORA metrics" nomenclature stems from the fact that the individuals responsible for the research and assessment were also the ones who created the company DORA and, the "Accelerate Metrics" nomenclature have its origins from the book Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations.

This metrics recognition is getting higher, a survey conducted by Atlassian [50] in 2020 revealed that nearly half of the inquired individuals use the DORA metrics to assess the DevOps in their companies, showcasing that the metrics really provides good insights and companies are adopting them.

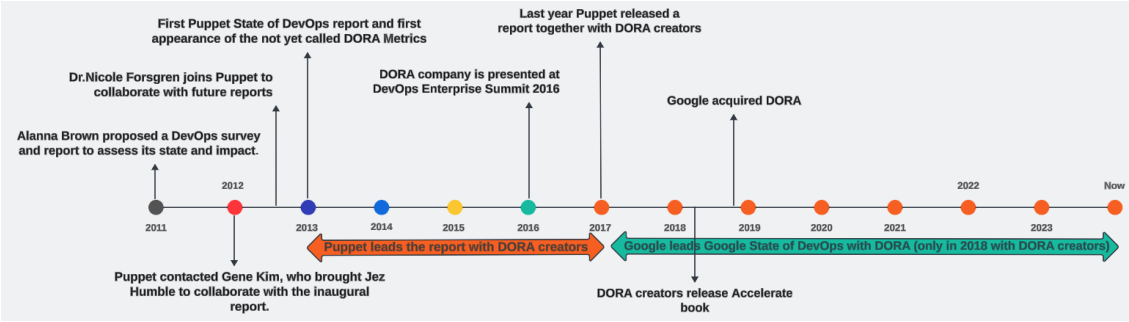

Figure 2.7: DORA metrics timeline

Figure 2.7 depicts a timeline that explains the history how DORA got formed through key events, ranging from 2011 until early 2024. The evolution of this topic is explained more in-depth in section 2.3.2.

Metrics

These metrics are separated into two types of indicators, throughput, and stability as shown below.

Throughput indicators:

- **Deployment Frequency:** Deployment frequency is the frequency that changes are introduced into production. It is highly correlated to batch size since reducing batch size reduces cycle times and variability in flow, accelerates feedback, reduces risk and overhead, improves efficiency, increases motivation and urgency, reduces costs, and fosters growth. Measuring software batch size has proven to be challenging due to the absence of visible inventory, therefore deployment frequency serves as an indirect measure of the batch size since it is easier to measure and typically has low variability.

A release, denoting a change set for deployment, involves various commits unless the organization has achieved the flow where each commit can be released to production, as previously stated as continuous deployment[10].

- **Change Lead Time:** Change lead time is defined as the time that it takes for a customer to request a change until the change is satisfied. In the context of product development, there are two parts to lead time: the time that it takes to design and validate a product or feature, and the part to deliver it to the client. This indicator focuses more on the part of the delivery to the client — the time it takes for work to be implemented, tested, and delivered — since it is easier to measure and has lower variability. Shorter lead times in product delivery are favorable as they facilitate faster feedback on the development process and allow the enhancement of the code. Additionally, shorter lead times prove crucial when a delivery of a fix is necessary. This metric is quantified by measuring the time it takes to transition from commitment to the successful execution of code in the production environment [10].

Stability Indicators:

- **Mean Time To Recover:** Mean time to recover is defined as how long it takes to restore service for the primary application or service when an incident occurs, such as an unplanned outage, service impairment [51].
- **Change Fail Rate:** This is the percentage of changes to an application or service that result in a failure in production, leading to a service impairment or outage, that requires a hot-fix, a rollback, a fix-forward or patch [10].

These indicators look forward to measuring the performance of software teams that focus on global outcomes in DevOps. Meaning, in the basic sense of DevOps, prioritizing the measurement that avoids pitting development against operations, by rewarding development for throughput and operations for stability and secondly, focusing on outcomes, not output [10]. Essentially, the approach is to discourage rewarding people for their volume of work, as this may result in increased output at the expense of quality and security that does not help achieve organizational goals, but rather assess results that contribute tangible business value. The DORA metrics distinguish between low, medium, high, and elite performers. In 2023, the Google State of DevOps [52] has presented the yearly delivery performance depicted in Table 2.4.

Table 2.4: Software delivery performance 2023

Performance Level	Deployment Frequency	Change Lead Time	Change Failure Rate	Mean Time to Recover
Elite	On Demand	Less than one day	5%	Less than one hour
High	Between once per day and once per week	Between one day and one week	10%	Less than one day
Medium	Between once per week and once per month	Between one week and one month	15%	Between one day and one week
Low	Between once per week and once per month	Between one week and one month	64%	Between one month and six months

The provided 2023 table allows companies to benchmark against the industry and identify the most important areas of focus.

History

To comprehend the evolution of this topic, it is essential to explore the origins of DORA and understand its evolution. For visual guidance, follow the text together with the Figure 2.7.

The initiative of this research program started without the existence of DORA, originating in 2011 when Alanna Brown, working with Puppet, conceived the idea of creating a survey and report to investigate the state and actual impact of DevOps practices [53].

In 2012 Puppet reached out to Mr.Gene Kim, the author of The Phoenix Project book, to collaborate with the inaugural report. Gene, in turn, brought in Mr.Jez Humble, the author of the Continuous Delivery book [53]. The 2012 survey was completed by over 4,000 technical professionals from over 90 countries, a stunning response rate for the time, leading to the release of the 2013 report. On it, is possible to observe the first mention of the not yet called DORA metrics. They presented it as two types of indicators, Key Agility Performance Indicators and Key Reliability Performance Indicators, with the same metrics mentioned on the previous section.

In the same year, Puppet crossed paths with Dr. Nicole Forsgren, an IT impacts specialist, at Large Installation System Administration Conference (LISA). Impressed by her expertise, Puppet extended an invitation for her to join the research. Dr. Forsgren accepted the offer, contributing to the production of the State of DevOps reports for 2014, 2015, 2016, and 2017 together [53].

In 2016, after working with two clients measuring the key capabilities of their teams and benchmarking them against the industry using capabilities that were predictive of performance outcomes and seeing strong results, Mr.Gene Kim and Mr.Jez Humble convinced

Dr.Nicole Forsgren to run the company full-time. After convincing her, in October 2016, DORA was presented at DOES 2016 in San Francisco with statements from Capital One and Verizon about the success of their assessment [54]. The case study of Capital One will be presented in the next section.

Puppet and DORA last collaborated on a report in 2017, collecting responses from 3200 IT professionals globally. During that year, Dr.Nicole Forsgren, Mr.Jez Humble, and Mr.Gene Kim wrote Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations based on all the previous reports and additional research. The book was published at 27 of March 2018.

On 20 December 2018, DORA was acquired by Google, since then the reports have been led by Google's team and are now part of the Google Cloud [55]. During the same year, they expanded the model to include the availability metric. This addition improved the ability to elucidate and predict organizational outcomes, providing a more comprehensive view of developing, delivering, and operating software. This new construct is called Software Delivery and Operational Performance (SDO Performance). This new analysis allows us to offer even deeper insight into the benefits of implementing DevOps [56]. Availability can be defined as "the accessibility of software to perform operations when required by a user. It is measured as the percentage of time the software is accessible for use when required. Availability can also be expressed as the probability that software will not be brought down (or becomes non-functional) for maintenance purposes when it is needed by the user" [57].

In 2021 Google, replaced availability with reliability stating that "Historically we have measured availability rather than reliability, but because availability is a specific focus of reliability engineering, we have expanded our measure to reliability so that availability, latency, performance, and scalability are more broadly represented." [58].

Case Studies

As previously mentioned, these metrics are gaining recognition within companies. This is due to their ability to offer valuable insights about the company, enabling to compare them among the other market players. In order to showcase the real value of these metrics, two case studies were analyzed and presented.

As expected, the first case study [59] is the one that was conducted by the DORA creators at Capital One, that due to its incredible results had to be mentioned. According to Forbes [60, 61], Capital One is the ninth largest bank in the United States of America (USA) in 2023, it has about 469 billion dollars in assets held almost entirely in the USA. Capital One is headquartered in McLean, Virginia, and serves customers in the United States, Canada, and the United Kingdom. Senior Director Adam Auercach, Leading DevOps transformation at Capital One, was proud of the technical team's current progress but was in search of the next improvement opportunity to boost the delivery of new features to their core banking systems. The team believed in that the use of measurement to guide insights and drive action was the right choice, but did not have the right tool to implement it in a way that would take into account their unique strengths and capabilities. In simpler terms, Capital One needed an assessment tool that would measure them as a whole and benchmark them against the industry and identify the most important areas for them to focus on, and that was where DORA joined. After using DORA Metrics to assess, they found two key capabilities that they needed to implement in order to improve: a more streamline change approval process and trunk-based development. The implementation of those capabilities resulted in

astonishing outcomes, with 20 times more number of releases and some applications had deployed more than 30 times to production with no increase of incidents.

The second, and last, case study was led by Sallin et al. [62] conducted at the IT unit of Swiss Post Ltd. with 330 software developers that run only projects for internal customers. The author's prior work, the development of an automatic measure dashboard of the DORA metrics, was applied to the company, and the team decided to track metrics weekly.

Through surveys with a Likert scale, six of the ten team members responded, revealing that four were familiar with DevOps metrics. Participants expected long-term improvements in software delivery attention, reduced deployment pain and anxiety, a more relaxed environment, encouragement of technical practices, some pressure on team members, and a source of motivation to improve. All participants considered these metrics valid for assessing software delivery velocity, stability, and DevOps adoption. However, they noted that the DORA metrics did not capture important aspects like the speed of infrastructure provisioning. Despite the team's experience with agile, DevOps, and software engineering, they moderately agreed on knowing which practices influence the metrics. This suggests that less mature teams require guidance to improve on the metrics raised.

In conclusion, the study found that it is possible to measure the DORA metrics automatically and the software developers team sees it as valuable, thus the prototype was going to be applied in all development teams at Swiss Post.

2.4 Tools

Name	Paid	Metrics	Integrations
Swarmia	Yes	DORA metrics and other industry best-researched metrics	GitHub, GitLab, Bitbucket, Azure DevOps, Jira, Linear, Asana, Shortcut, ClickUp, Slack, Jenkins, CircleCI, and Buildkite but adapts to the client's needs
Apache DevLake	No	DORA metrics, other predefined metrics but allows making our own	GitHub, GitLab, Jira, Jenkins, Bitbucket, TAPD, Teambition, Zentao, Gitee, PagerDuty, Feishu, AE, Sonarqube, Bamboo CI, and Azure DevOps but if data source is not directly supported, they provide a webhook to create our own.
Code Climate Velocity	Yes	Various predefined metrics but classify the DORA metrics as a key feature	Links data from all systems.
LinearB	Yes	DORA metrics, pipeline delivery, quality metrics, team throughput and adapts to clients needs	Slack, Microsoft Teams, Jit, Circle CI, Jenkins, Swimm, Azure Repos, GitLab, BitBucket, GitHub, Jira, Shortcut and SonarCloud

Table 2.5: Tools to gather data from different sources

Having presented some metrics that are widely used to assess product quality and developers' efficiency, it is important to gather data in an automated and discrete for the professionals.

This automation eliminates manual intervention and enhances consistency [36]. In the following sections, some tools will be briefly presented, with the focus of presenting them to the stakeholders. The key aspects of the tools are: if it is paid, what metrics does it allow visualizing and what integrations it offers. To simplify the view of all the tools, they are summarized in Table 2.5.

2.4.1 Apache DevLake

Apache DevLake [63] is an open-source platform designed to gather, analyze, and visualize data from various DevOps tools. Its flexibility accommodates small and large teams, with the ability to extend support for new data sources, metrics, and dashboards. The dashboards are constructed using Grafana.

DevLake integrates with multiple data sources like GitHub, GitLab, Jira, Jenkins, Bitbucket, TAPD, Teambition, Zentao, Gitee, PagerDuty, Feishu, AE, Sonarqube, Bamboo CI, Azure DevOps and Circle CI. If a data source is not directly supported, they provide a webhook to import deployments and incidents from unsupported data integration to calculate DORA metrics, and others. Apache DevLake centralizes data for analysis, for example, a company can use Jenkins for CI/CD, GitHub for source code management to define issues, and SonarQube to ensure the quality of the code and with the variety of connectors they offer it is possible to unify their information in one place. Existing data sources have predefined dashboards for easy metric gathering.

2.4.2 Code Climate Velocity

Velocity [64], it is a paid software engineering intelligence platform that allows, to gain the needed comprehensive insights to boost productivity and efficiency and maximize engineering impact. It automatically and securely, ingests, cleans, and links data from all systems where engineers work allowing to make more informed, data-backed information. In shorter terms, it allows seeing how work moves through the software development life cycle. They permit the analysis of various predefined metrics but classify the DORA metrics as a key feature, stating that they “Balance Speed and Stability”.

2.4.3 LinearB

LinearB [65] is a paid platform, free with restricted functionalities, that allows visibility and automation across the organization, allowing to streamline the operations to increase predictability, reduce costs and improve the quality of each release. It provides flexibility in which metrics the client wants but has a predefined set of metrics such as DORA metrics, pipeline delivery, quality metrics and team throughput. As the writing of this document LinearB allows the following integrations Slack, Microsoft Teams, Jit, Circle CI, Jenkins, Swimm, Azure Repos, GitLab, BitBucket, GitHub, Jira, Shortcut and SonarCloud.

2.4.4 Swarmia

Swarmia [66] is a paid platform, free for companies with up to 14 developers, that helps software organizations improve business impact, engineering productivity, and developer experience. To achieve that, they provide a dashboard with engineering metrics such as DORA metrics, and other industry best-researched metrics. Their platform already has connectors

to GitHub, GitLab, Bitbucket, Azure DevOps, Jira, Linear, Asana, Shortcut, ClickUp, Slack, Jenkins, CircleCI, and Buildkite but adapts to the client's needs.

Chapter 3

Organizational and Tool Analysis

This chapter presents the actions taken to address **Objective 2** and partially **Objective 3**, focusing on the analysis of the organizational environment and the chosen tool. The process began with an assessment of key performance indicators: deployment frequency, lead time for changes, mean time to recovery, and change failure rate, using the DORA Quick Check. Next, the tools and technologies used daily within the company, especially those for data collection and analysis, were examined. Finally, the selected tool was presented in more depth.

This chapter is composed by three sections:

- **Dora Quick Check:** Analyzing the responses of a questionnaire that allows companies to benchmark themselves against industry standards.
- **DevOps Tool Chain:** Presents the DevOps tools that the company uses on a daily basis.
- **DevLake:** This section explains the selected tool, DevLake, used for gathering data and creating metrics covering key aspects of the tool.

3.1 DORA Quick Check

Dora Quick Check [67] is a questionnaire that allows companies to benchmark themselves against industry standards. It uses straightforward questions to identify the values of each DORA metric:

- **Lead time**: For the primary application or service, how long does it take to go from code committed to code running successfully in production?
- **Deploy frequency**: For the primary application or service, how often does your organization deploy code to production or release it to end users?
- **Change fail percentage**: For the primary application or service, what percentage of changes to production or released to users result in degraded service and require remediation (e.g., hotfix, rollback, fix forward, or patch)?
- **Mean time to recover**: For the primary application or service, how long does it take to restore service after a change in production or release to users results in degraded service and requires remediation (e.g., hotfix, rollback, fix forward, or patch)?

The possible answers to each question are listed in Table 2.4.

The stakeholder and some employees completed the questionnaire, and they provided the following responses:

- **Lead time:** One to six months.
- **Deploy frequency:** Between once per month and once every six months.
- **Change fail percentage:** Between 0% and 15%.
- **Mean time to recover:** Less than one day.

For each metric, all respondents provided the same responses. The results of the DORA Quick Check indicated that the company has a score of 5.1 out of 10. This score is slightly above average, indicating there is room for improvement.

In Chapter 6, there is a case study that examined the automatically gathered DORA metrics and compare them with the answers provided to this questionnaire. Comparing these initial responses will show to what extent the employees and stakeholders' views match the actual data.

3.2 DevOps Tool Chain

This section demonstrates the tools the company uses in its DevOps processes, including Jira, Bitbucket, Jenkins, and SonarQube. These tools will be the source of the data that will be used in the metrics. The role of each tool is briefly explained.

3.2.1 Jira

Jira Software [68] serves as an agile project management solution that enables teams to plan, track, release, and support software with confidence. The company relies on Jira precisely for these purposes.

The company uses a workflow based on the agile methodology that is divided into two phases: planning and development.

Planning Phase

The image 3.3 illustrates the stages of the planning phase.

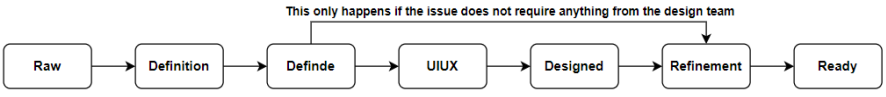

Figure 3.1: Planning phase stages

- **Raw**: An issue was identified and was created, but has not been worked on yet.
- **Definition**: In this initial stage, the issue is identified, and its requirements are outlined.
- **Defined**: The issue is clearly described, and its scope is understood.
- **UIUX**: User interface and user experience considerations are addressed for optimal design and usability.

- **Designed**[1]: The issue's design is developed and finalized.
- **Refinement**: Further enhancement and optimization of the issue occur.
- **Ready**: The issue is prepared for assignment to a sprint for development.

Development Phase

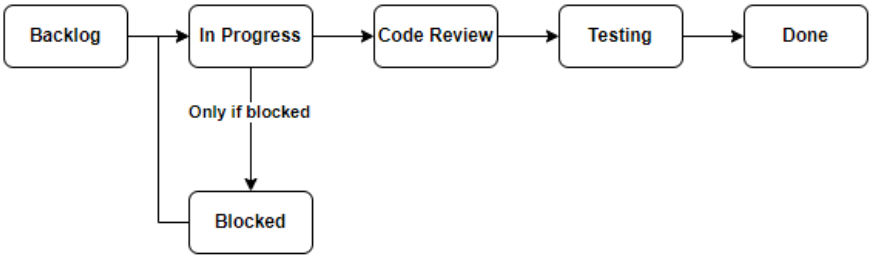

Figure 3.2: Development phase stages

- **Backlog or To Do**: Tasks are listed for implementation.
- **In Progress**: Active work is undertaken on the issue.
- **Blocked**: Development halts due to dependencies on other tasks.
- **Code Review**: Submitted pull requests await evaluation and approval (or rejection).
- **Testing**: The testing team assesses changes for bugs and problems.
- **Done**: Successful completion of the issue.

It is important to mention these stages from both phases because some metrics are filtered based on them.

The word "issue" on this document typically represent individual work items such as big features, user requirements, and software bugs. On other project management tools can also be referenced as tickets.

3.2.2 Bitbucket

Bitbucket [69] is a git-based code hosting and collaboration tool, it offers integration with Jira, bringing the entire software team together to execute a project. Providing one place for a team to collaborate on code from concept to cloud, build quality code through automated testing, and deploy code with confidence. Even though Bitbucket offers pipeline solutions, the company opted to use Jenkins, presented in the next subsection.

3.2.3 Jenkins

Jenkins is a widely used open-source automation server, the company uses this for continuous integration, deploys and, automation of other jobs.

[1] Only issues requiring design go through the UIUX and Designed stages, as depicted on Figure 3.1

3.2.4 SonarQube

SonarQube is an open-source platform that performs automatic code quality reviews with static code analysis to detect bugs and code smells in various programming languages. The company uses this to track the unit test coverage, and to perform static code analysis in all projects.

3.3 DevLake

The combination of tools employed by the company establishes a solid foundation for planning, developing, and delivering high-quality software. Recognizing the stack of tools in use made the tool selection process more straightforward. In Section 2.4, various tools were presented as potential options for the gathering of metric data, but ultimately DevLake was chosen because of its open-source nature, vast connectors including integration with the company's existing tool set, user-friendly interface, and a growing community of both users and contributors since it is open-source software.

DevLake is an open-source platform that ingests, analyzes, and visualizes data from fragmented DevOps tools. It was first released in 30 September 2021 [70], it is currently maintained by 130 contributors, and it has a growing community. The version that was used was 0.19.0.

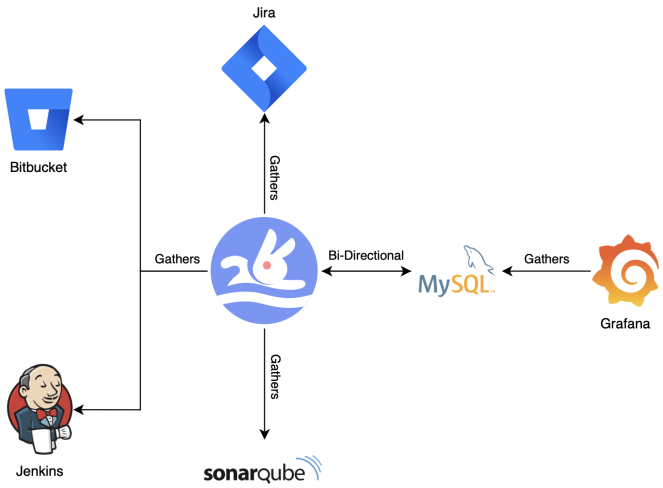

Figure 3.3: Architecture Illustration

Figure 3.3 provides a simplified illustration of the expected system architecture, a more in-depth one is available in Section 4.1. DevLake is responsible for collecting metrics from Bitbucket, Jenkins, SonarQube, and Jira, and subsequently stores the data in a MySQL database. Grafana will then utilize the MySQL database data to generate the metrics.

DevLake has concepts that can be used on the Config UI or via API to set up the gathering process. These will be explained in detail in the following subsections.

3.3.1 Data Source

A data source is a specific DevOps tool from which it is possible to extract data. At this moment, DevLake supports the same data sources as was presented in Chapter 2.

3.3.2 Data Connection

One data connection is a specific instance of a data source. It stores the necessary access information, such as the endpoint Uniform Resource Locator (URL) and authentication credentials, to establish a connection to that data source.

A single data source can have one or more data connections associated with it. Allowing to connect to and retrieve data from different instances or installations of the same data source. For example: Bitbucket data source can have two connections, one for Project A and another for Project B.

3.3.3 Data Scope

A data scope is the top-level 'container' in a data source. For example, a data scope for Jira is a Jira board, for Bitbucket and SonarQube is a repository, for Jenkins is a Jenkins job, and so on.

It is possible to have multiple data scopes in a data connection to determine which data to collect.

3.3.4 Scope Configuration

A scope configuration is related to the configuration of a data scope. Describes the specific data entities to be collected and the transformations to be applied to those data.

Each data scope can have at most one scope configuration associated with it; while a scope configuration can be shared among multiple data scopes under the same data connection.

A scope configuration consists of the following two parts:

Data Entities

Data entities refer to the specific data fields that are collected from different data domains. Data entities are categorized into six data domains in DevLake:

- Issue Tracking
- Source Code Management
- Code Review
- CI/CD
- Code Quality
- Cross-Domain

As previously given as a reference GitLab is one data source that when configuring the scope configuration it is possible to select Source Code Management, Code Review, Issue Tracking, CI/CD data entities.

These domains will be explained more in-depth together with the database as it organizes the tables using the same logic.

Transformations

Transformations are configurations for users to customize how DevLake transforms raw Application Programming Interface (API) responses to the domain-layer data.

Although configuring transformation rules is not mandatory, certain pre-built dashboards, such as DORA metrics, require them to display the metrics accurately. If you leave the rules blank or have not configured them correctly, only a few data source dashboards will be displayed, as expected.

Transformations do not apply to all data sources, only for specific ones.

3.3.5 Project

DevLake projects can be viewed as real-world projects or product lines. It represents a specific initiative or endeavor within the software development domain. In more in other terms, it is a way of organizing and grouping data from different domains. DevLake uses various data scopes, such as repositories, boards, CI/CD scopes, and Code Quality projects, as the "container" to associate different types of data to a specific project. A project has multiple blueprints.

Projects are very similar to blueprints, the major difference is that in a Project it is possible to enable DORA metrics.

3.3.6 Blueprint

A blueprint serves as the plan for synchronizing data from data sources with the DevLake platform. Creating a blueprint consists of three steps:

- **Adding data connections:** You can add one or more data connections to a blueprint, depending on the data sources you want to sync with DevLake. Each data connection represents a specific data source, such as GitHub or Jira.
- **Setting up the data scope:** When adding a data connection, you can choose to collect all or part of the configured data scopes of the data connection.
- **Setting up the sync policy:** You can specify the sync frequency and the time range for data collection.

These steps will be reviewed in Chapter 4.

3.3.7 DevLake Database

After collecting data from various sources using DevLake, all information will be stored in a MySQL database. This database is divided into six domains, data entities on the DevLake UI, which can be correlated depending on the data source. The five domains are:

- **Issue Tracking:** Contains data from sources that manage issues, consisting of fourteen tables.
- **Source Code Management:** Involves commits, tags, and branches, related to a repository, consisting of ten tables.
- **Code Review:** Relative to pull requests, consists of four tables.

- **CI/CD:** Relative to jobs and build of tools that allow automation, consists of six tables.
- **Code Quality:** Currently, only SonarQube provides data for this domain, consisting of four tables.
- **Cross-domain:** Maps entities from different domains to integrate data, consisting of nine tables.

In addition to these forty-seven tables, each tool has its own specific tables for their data. These forty-seven tables centralize the data, making it easier to perform queries across different sources and correlate the data.

3.3.8 Use Cases

Knowing the key concepts, let's visualize DevLake's use case diagrams, which illustrate the possible actors in the system and the actions they can perform. Due to readability concerns, these diagrams have been divided into two, the use case diagram of connections and the use case diagram of projects and blueprints.

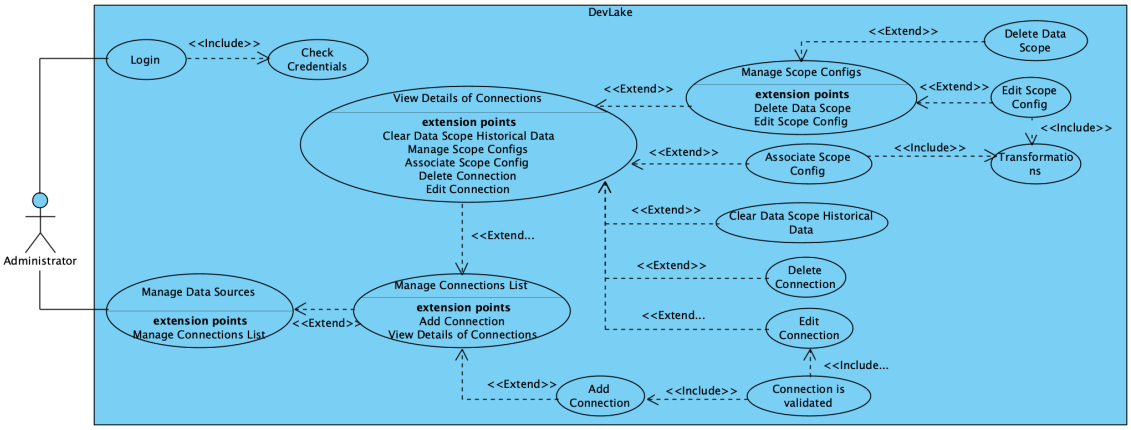

Figure 3.4: DevLake connections use case diagram

For DevLake, there is only one actor: the administrator. This is because the DevLake Config UI contains sensitive information that, if exposed to unauthorized individuals, could pose significant risks to the company.

In Figure 3.4, it is possible to observe that the administrator can log in and navigate to the data source page. On this page, when the actor clicks one of the data sources he can manage the connections, meaning that he can create new connections or view details of already created connections.

Still, with respect to existing connections, the administrator can also edit, delete, clear historical data on the scope of gathered data, associate a scope configuration, and manage associated scope configurations, if they exist. In either case of creation or edition of a connection, they are validated by establishing a connection to ensure they are correct.

Additionally, the administrator can delete or edit the scope configurations and, for certain data sources, perform transformations when editing or associating a scope configuration.

Figure 3.5 illustrates the use cases regarding projects and blueprints.

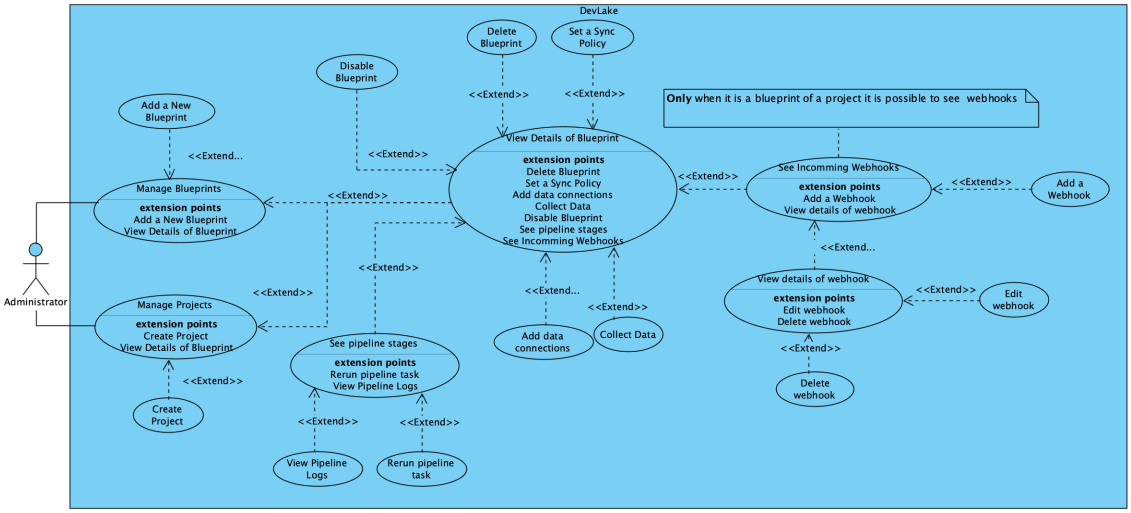

Figure 3.5: DevLake blueprints and projects use case diagram

The system's sole actor can manage blueprints, meaning that he can view or create new ones. In the blueprint details, the administrator can add a data connection, set the synchronization policy, which defines when the blueprint will run the pipeline to gather data from the specified data source or data connection, manually trigger the data collection process, and disable or delete the blueprint. Additionally, within the blueprints, it is possible to visualize the pipeline stages, where the administrator can view logs or rerun specific pipeline tasks or the pipeline as a whole.

Regarding projects, he can create a new project or view the details of an existing one. Inside projects, he can perform every action made in the blueprints with the addition of the creation of webhooks to receive data from other sources that are not integrated into the tool.

This integration is beneficial for calculating metrics such as DORA (DevOps Research and Assessment) metrics. For example, using Elasticsearch Heartbeat, a webhook can be triggered when a service goes down, classifying it as an incident in DevLake. On the webhook page, the administrator can create a new webhook, delete an existing one, view webhook details, edit a webhook, or delete a webhook.

3.3.9 Grafana

Grafana is the tool DevLake uses to present metrics. During the early phases of implementation, Grafana was attached to DevLake according to the documentation. However, during implementation, the company adopted Grafana for other purposes, resulting in a different version from the one used in DevLake version 0.19.0. The Grafana version used was the open-source version 10.3.3.

Grafana comprises multiple components, but only the most frequently used key components will be explained.

Dashboard

A Grafana dashboard is an adaptable, interactive panel that presents data from various sources visually. Grafana dashboards are employed to monitor, analyze, and visualize metrics, logs or both.

Visualization Panel

The panel is the fundamental unit of a Grafana dashboard. Each panel can showcase data in multiple formats such as graphs, tables, heatmaps, and more.

Queries

Queries define how data is retrieved from a data source. Grafana provides a query editor customized for each data source type, allowing users to build queries for filtering, aggregating, and transforming data. The query language depends on the data source being queried; for instance, Structured Query Language (SQL) is used when querying a MySQL database.

Variables

Grafana variables, also referred to as filters, boost dashboard interactivity and customization by serving as dynamic placeholders in queries and visualizations. There are various types of variables, including query variables, custom variables, interval variables, and data source variables. These variables enable users to adjust the data scope dynamically without altering the dashboard configuration.

Use Cases

Although Grafana offers many functionalities, the use case diagrams will only depict the most commonly used ones.

Grafana can have as many actors as needed, in this specific case, there are four actors, Administrator, Frontend Developers, Backend Developers, and Project Managers. The use case diagrams for Grafana are divided by the existing actors.

Due to the high number of use cases for the Administrator actor, the diagram was split into dashboard management and management of data sources, plugins, and organizations. Starting with dashboard management, as depicted in Figure 3.6, these actions are primarily performed by the administrator.

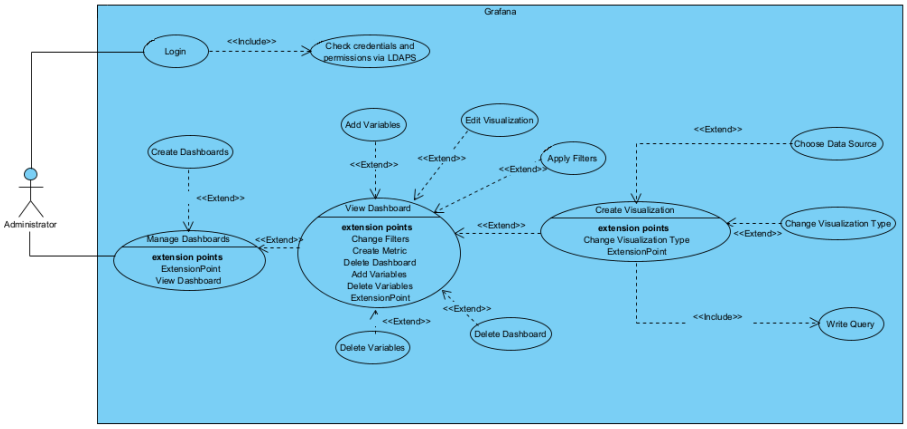

Figure 3.6: Grafana use case diagram administrator manage dashboard

All actions in Grafana require user authentication. Administrators have the capability to manage dashboards, which includes creating new ones and viewing existing ones along with

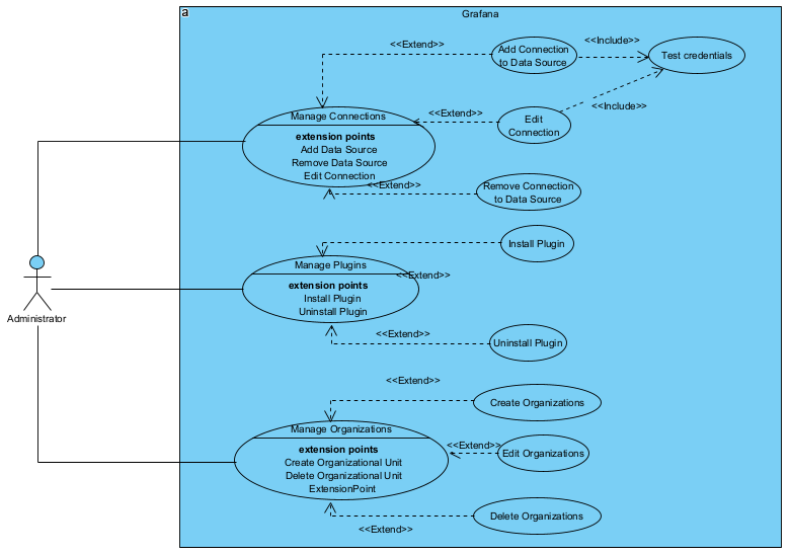

Figure 3.7: Grafana use case diagram administrator manage data sources, plugins, and organizational units

their visualizations. Within a dashboard, they can also delete them, add variables, remove variables create, edit, and delete visualizations, select data sources, and write queries.

Figure 3.7 illustrates actions regarding data sources, plugins, and organizations; all of these actions are only possible to perform by an administrator.

As an administrator, it is possible to manage data sources. When managing them, it is possible to add a new connection and edit or delete an existing one. Whenever a connection is added or edited, it is always tested to see if the provided credentials and configuration are correct. Furthermore, the administrator also has the capability to manage plugins, meaning that he can install and uninstall plugins.

Finally, the last use case that the administrator can perform is to manage organizations, meaning that he can create new organizations or edit and delete already existing ones.

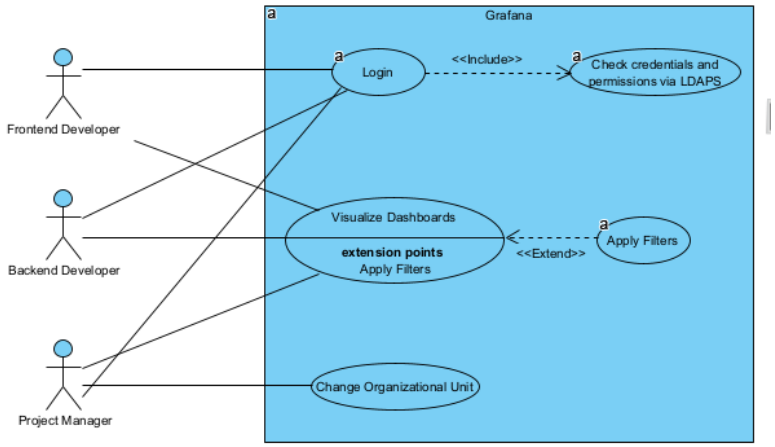

Figure 3.8: Grafana use case diagram for frontend and backend developers and project managers

Figure 3.8 illustrates the actions of the remaining actors in Grafana.

These roles share the same use cases, except for the project manager, who has the additional capability to change the organizational unit. DevLake design will be explored in more depth in Chapter 4.

Chapter 4

Design

This chapter aims to explain the design of the solution, including the selection of the metrics, the rationale behind each one of them and how was Grafana organized. Contributing to the still partially achieved **Objective 3**.

This section is composed by five sections:

- **Architecture:** This section explains all the components of the DevLake setup and details the connections between them.
- **Metrics:** This section presents a summary of the selected metrics.
- **Metrics Relevance:** This section explains the relevance and reasoning for each selected metric.
- **Dashboard:** This section explains how the dashboards will be organized.
- **Security Aspects:** The final section explains the security considerations taken into account when installing and configuring the tool.

4.1 Architecture

The component diagram in Figure 4.1 illustrates the DevLake setup that was implemented.

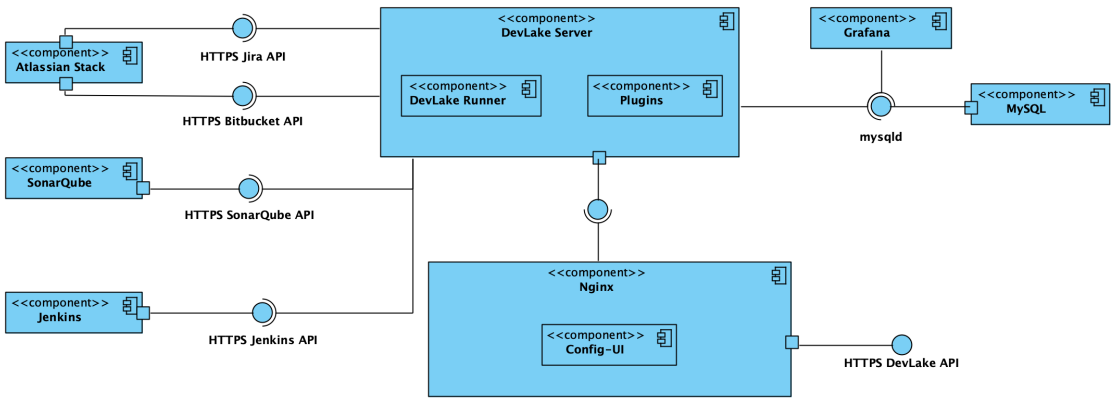

Figure 4.1: Component Diagram of DevLake Installation

DevLake is composed of many components, including:

- **DevLake Server:** The DevLake server is the programmatic interface of DevLake that provides a REST API handling requests and managing data pipelines. This API ends up being available through Nginx, that acts as a web server and gateway. Communicates directly with all components except for Grafana. But, inside this DevLake server, there are two other components:
  - **DevLake Runner:** Is the component that handles the execution of tasks, performing heavy data processing. Operates within the DevLake Server or as a temporal-based runner for production environments.
  - **Plugins**: Enables DevLake to collect and analyze data from various DevOps tools with accessible APIs. These plugins will be later called, Data Sources.
- **Nginx:** Is the web server that hosts DevLake and makes the DevLake API and the UI available to the public.
  - **Config UI:** Provides a user-friendly interface for creating, triggering, and debugging Blueprints. These concepts will be further explored. Communicates with the DevLake Server.

  Other components that DevLake requires to work are:
- **Database (MySQL):** Stores DevLake's metadata and user data collected by data pipelines, and Grafana uses it as a data source to provide metrics.
- **Grafana:** It serves as the official dashboard tool for DevLake, already provides some pre-built dashboards, but it is also possible to build other metrics. In this implementation, it is external to DevLake. Interacts with the MySQL database to perform the queries and generate the metrics.

Finally, on the left side of the diagram, are depicted the DevOps tools that the company uses that provide their APIs so DevLake plugins can extract them, and these are:

- **Atlassian Stack (Jira API, Bitbucket API):** Provides APIs for Jira and Bitbucket, used by DevLake Worker for data collection and analysis. Communicates with the DevLake Server via Hypertext Transfer Protocol Secure (HTTPS).
- **SonarQube (SonarQube API)** Offers an API for collecting code quality metrics, utilized by DevLake Worker. Communicates with the DevLake Server via HTTPS. **Jenkins (Jenkins API):** provides an API for the collection of CI/CD data used by DevLake Worker. Communicates with the DevLake Server via HTTPS.

4.2 Metrics

The table provided below showcases the names, descriptions, filters, types, and sources of the selected metrics. Notably, a column for metric type is included to prevent misinterpretation.

There are six types:

- **Commitment**: These metrics assess the involvement of the development team or the developer and the dedication and integration to a project.
- **Efficiency**: These metrics evaluate the efficiency of the development team.
- **Issue Tracking**: These metrics help understand the status and progression of issues.

- **Maintainability**: These metrics gauge the ease of maintaining company projects.
- **Reliability**: These metrics measure the reliability of projects and processes.
- **Security**: These metrics pertain to security concerns.

4.2.1 Product Quality

The following table of metrics is directly related to the quality of the product, incorporating the metrics that significantly influence it.

Table 4.1: Summary Product Quality Metrics

Name	Description	Filters	Type	Source
Number of Vulnerabilities	Quantity of vulnerabilities	Repository, Time Range	Security	SonarQube
Code Coverage	Percentage of code that is covered by unit tests	Repository and Time Range	Reliability	SonarQube
Code Smells	Quantity of traces in the code that indicates deeper issues	Repository and Time Range	Maintainability	SonarQube
Code Debt	Cost of reworking code due to identified issues	Repository and Time Range	Maintainability	SonarQube
Percentage of Duplicated lines	Proportion of duplicated lines compared to total code	Repository and Time Range	Maintainability	SonarQube
Successful Build Rate	Proportion of successful build compared to all builds	Pipeline and Time Range	Reliability	Jenkins
Build Success Rate per Month	Percentage of Successful Builds Per Month	Pipeline and Time Range	Reliability	Jenkins
Deployment Frequency	How often does the company deliver changes to production	None	Efficiency	Jenkins
Mean Build Duration Time	Average duration for pipeline job completion	Pipeline and Time Range	Efficiency	Jenkins
Change Failure Rate	How often do the changes to production fail	None	Reliability	Jenkins
Bug Priority Evolution	Number of bugs per priority per month	Time Range	Issue Tracking	Jira
Average Age of Bug	Age of bugs by priority	Time Range	Issue Tracking	Jira
Bug Evolution	Total number of bugs per month	Time Range	Issue Tracking	Jira
Raw Issue Distribution	Number of raw issues distributed across all months	Time Range	Issue Tracking	Jira

4.2.2 Developers Efficiency

The following table will list metrics that affects both developer efficiency and those tailored towards management, facilitating informed decision-making to manage the developers.
small

Table 4.2: Summary Developers Efficiency Metrics

Name	Description	Filters	Type	Source
Pull Request Size	Average Number of lines of code per pull request per month	Repository and Time Range	Efficiency	Bitbucket
Number of contributions	Number of commits of a developer and all developers per month	Developer, Repository and Time Range	Commitment	Bitbucket
Pull Request Review Depth	Average number of comments on pull requests per month	Repository and Time Range	Commitment	Bitbucket
Pull Request Merge Rate	Percentage of merged pull requests	Repository and Time Range	Efficiency	Bitbucket
Pull Request Status Distribution	The status of all pull requests in each month	Repository and Time Range	Efficiency	Bitbucket
Mean time to merge pull requests	Average time for pull requests to be merged	Repository and Time Range	Efficiency	Bitbucket
Number of pull requests	Number of pull requests in each month	Repository and Time Range	Efficiency	Bitbucket
Ratio of non-merging pull requests	Percentage of non-merged pull requests in each month	Repository and Time Range	Efficiency	Bitbucket
Issue State Distribution	Number of issues in each stage of the planning and development phases	None, Sprint and Time Range	Issue Tracking	Jira
Average Time to plan	Average time an issue takes to go from Definition to Ready in each month	Issue Type, Priority and Time Range	Efficiency	Jira
Average Time to Solve Issues	Average time that it takes to an issue to go from In Progress to Code Review in each month	Issue Type, Priority and Time Range	Efficiency	Jira
Average Time to Test Issues	Average time an issue takes to go from Testing to Done in each month	Issue Type, Priority and Time Range	Efficiency	Jira

Average Time Issues Stay in Raw State	Average time an issue takes to start to be picked up in each month	Issue Type, Priority and Time Range	Efficiency	Jira
Reopens Evolution	Number of issues that go from Testing to In Progress in each month	Sprint and Time Range	Efficiency	Jira
Development Time Variance	Comparison between estimated and actual development time	Developer and Sprint	Efficiency	Jira
Reopens Percentage	Percentage of issues that went from Testing to In Progress	Time Range	Efficiency	Jira

4.3 Metrics Relevance

When determining metrics with stakeholders and project management, relevance was the main concern, because the objective of the dashboards that will be created is to be easy to analyze and straight to the point. Similar to our previous approach of dividing tables into two main types, we maintained this consistency here.

4.3.1 Product Quality

Number Of Vulnerabilities: By tracking the number of vulnerabilities over time, development teams and stakeholders can evaluate the effectiveness of their security measures and practices. A high number of vulnerabilities indicates weaknesses in the codebase, since the vulnerability count is provided by SonarQube.

Code Coverage: This metric provides insights into the thoroughness of testing efforts, indicating how much of the codebase these tests cover. High unit test coverage suggests a greater assurance of code quality and reliability, reducing the likelihood of undiscovered bugs or defects.

Code Smells: Code smells [71] are categories of issues in code that may lead to deeper problems over time, typically associated with poor coding practices. Identifying code smells early in the development process can help prevent the accumulation of technical debt and improve overall code quality.

Code Debt: Code debt represents the estimated time required to fix all the code smells. This metric is provided by SonarQube and the estimate can vary based on its configuration, as the time needed to address each type of issue depends on specific settings. Assessing code debt is essential for managing and mitigating it, which is crucial for maintaining the health of a software project.

Percentage Of Duplicated Lines: This metric is important because high values suggest possible inefficiencies or redundancies in the code, which can result in maintenance difficulties, reduced readability, and a higher likelihood of errors. Fewer duplicate lines means cleaner, more maintainable code, improving overall software quality and developer efficiency.

Successful Build Rate: The build success rate is a key indicator of the overall health and reliability of the software development process. By monitoring the success rate of

builds, the operations team can assess the effectiveness of their practices and infrastructure, identify potential problems or areas for improvement, and ensure that software releases are consistently of high quality.

Build Success Rate Per Month: Tracking the build success rate monthly allows the operations team to identify trends and patterns in build performance over time. By analyzing variations in the success rate from month to month, it is possible to identify potential factors affecting build reliability and take proactive measures to address them, ensuring consistent and reliable software delivery.

Mean Build Duration Time: The mean build duration time offers insight into the efficiency of the build process. By analyzing the average time taken to complete builds, it is possible to identify bottlenecks or inefficiencies in build infrastructure and workflows, allowing to optimize the processes for faster and more reliable builds.

Bug Priority Evolution: It enables the company to monitor patterns. For instance, a sudden surge in high-priority bugs could signal recent erroneous codebase alterations or functional regressions. Additionally, it aids in resource allocation; excessive high-priority bugs may necessitate reallocating resources to tackle urgent matters or reassessing project timelines.

Bug Evolution: It enables the identification of periods characterized by increased bug discovery, thereby indicating potential problems within the development or testing phases. This prompts examination into why many bugs are only being identified at that particular moment, questioning their presence over an extended duration.

Average Age of Bugs: It helps in identifying areas for enhancement within the bug resolution process, facilitating prioritization of efforts accordingly.

Raw Issue Distribution: This metric illustrates the distribution of issues created that are still in the raw state in different months. Understanding when issues in the raw state were created provides valuable insights into patterns of issue creation over time. This information can help identify trends, such as peak periods of issue creation, which may indicate potential challenges or opportunities for process improvement.

4.3.2 Developers Efficiency

Pull Request Size: Can serve as a good indicator of the efficiency, risk, and overall quality of the software development process. Smaller pull requests signify that developers are working on issues in more manageable units, making reviews easier and reducing the likelihood of introducing bugs.

On the other hand, larger pull requests suggest that developers are building substantial changes into a single pull request. While this may indicate progress on complex features, it often poses challenges such as reviewers facing difficulty comprehending all the changes, potentially resulting in overlooked bugs or quality problems, increasing the risk of destabilizing the codebase.

Number of contributions: Monitoring contributions offers insights into individual or team engagement with the repository. It helps measure activity levels, spot development patterns, and track progress. A high number of contributions suggests greater developer commitment, which could lead to higher quality project contributions. However, it is crucial to consider other metrics like bug count, vulnerabilities, and code smells alongside this indicator.

Conversely, a low number of contributions from a developer may signal a need for assistance from project managers and/or technical leads to help them understand the requirements or code and get back on track.

Pull Request Review Depth: Tracking the average number of comments per month in pull requests provides a rapid assessment of developer engagement with the product. A high number of comments may signal active commitment in product enhancement. However, it could also suggest problems with understanding requirements, leading to longer development times.

In contrast, a lower number of comments may indicate smooth pull request processes or minimal developer engagement in the project.

Pull Request Merge Rate: Monitoring the pull request merge rate helps development teams identify areas for improvement in their workflow, identify potential blockers, and ensure a smooth and efficient development process. A high pull request merge rate suggests that developers are actively contributing and that their work is being integrated into the project quickly.

On the other hand, a low pull request merge rate might signify problems such as a backlog of unreviewed pull requests, code review bottlenecks, or communication problems within the team. It could also indicate a lack of engagement or productivity among team members.

Pull Request Status Distribution: This metric provides valuable insights into the efficiency and overall health of a team or organization's software development process. Reviewing the status of pull requests for a specific month allows for an understanding of the progress of development within a project. This understanding is crucial for assessing progress and identifying areas for improvement in the development workflow.

Mean Time To Merge Pull Requests: Monitoring the mean time to merge provides insights into the efficiency of code review and integration processes and helps to identify bottlenecks. Lower values signify efficient workflows with prompt code review and integration. However, excessively low values coupled with high reopens and bugs may imply superficial pull requests reviews for speed rather than quality.

Conversely, higher values indicate review process bottlenecks, potentially hindering development progress. Elevated values often correlate with heavy developer workloads, large pull request sizes, and limited test automation percentages. Fluctuations in the metric may suggest seasonal variations in workflow dynamics.

Number of Pull Requests: This metric indicates development activity and team efficiency, helping to evaluate the project's momentum by showing the volume of contributions over time. A higher number of pull requests suggests active development and ongoing improvement. Tracking this metric aids in resource allocation and highlights periods of high or low development activity. It also helps identify productivity patterns.

Ration of non-merging pull requests: The proportion of non-merging pull requests measures the proportion of pull requests that are not merged, indicating the viability of the code review process. A high ratio may point to quality problems, inefficiencies within the review process, or misalignment in team communication. This metric is significant for distinguishing potential problems in the workflow and areas where the team needs to improve their practices.

Issue State Distribution: It offers a snapshot of issue distribution throughout the planning and development process, helping teams to be aware of their current workload status. This enables the identification of potential bottlenecks or areas of issue accumulation. Furthermore, it enhances communication among team members by fostering a shared understanding of the status of issues.

Average Time to Plan Issues: It provides valuable data for predictability and serves as a benchmark for evaluating the efficacy of process enhancements or changes to improve planning activities.

Average Time to Solve Issues: This metric provides valuable insights into the efficiency of issue resolution processes, allowing teams to assess their performance and identify areas for improvement. Focusing solely on completed issues within a specific period, offers a more accurate representation of actual resolution times, enabling teams to make informed decisions and optimize their workflows accordingly. A high average time to solve issues can suggest that the issues are not well refined, when planning the sprint the project manager did not consider the developer capacity or even the lack of commitment of developers on the project.

On the other hand, lower values represent the inverse of higher values. However, it is important to note that if the development team rapidly resolves issues but experiences a rise in metrics such as reopen percentage, number of bugs, and vulnerabilities, it may indicate a tendency to prioritize speed over the best practices.

Average Time to Test Issues: This metric offers valuable insights into the efficiency of the testing process. By tracking this metric over time, teams can identify trends and patterns in testing durations, allowing them to optimize their testing workflows and improve overall efficiency. Additionally, it helps teams ensure timely delivery of quality-tested features or fixes by highlighting any areas where testing may be taking longer than expected. A high average time to test issues could indicate that the testing process is encountering challenges or inefficiencies, such as being overwhelmed or lacking resources to test changes.

On the other hand, a low average time to test issues can suggest a well-resourced and efficient testing team, capable of effectively validating changes. However, important to guarantee that this efficiency does not compromise the thoroughness of testing, as overlooking critical issues can have negative effects on product quality and user satisfaction.

Average Time Issues Stay in Raw State: This metric helps analyze the average time taken for issues to be picked up each month. By tracking this metric, it is possible to gain insights into the responsiveness of the planning team and identify any delays in the workflow. Understanding these patterns allows for better planning and resource allocation, ensuring that issues are handled promptly and efficiently.

Reopens Evolution: Tracking the number of reopens per month provides valuable insights into the stability and effectiveness of the development and testing processes. A high number of reopens may indicate problems with the quality of testing or the completeness of development work. By monitoring this metric over time, teams can identify patterns and trends, allowing them to implement targeted improvements to reduce the frequency of reopens and enhance overall efficiency and quality assurance practices.

Development Time Variance: Provides valuable insights into the accuracy of project planning and estimation processes. The project manager can identify potential problems such

as underestimating issues complexity, resource constraints, or unexpected delays by analyzing the variance between estimated and actual completion times. This information enables the project manager to improve future estimations, allocate resources more effectively, and enhance overall project management practices to ensure better alignment between planned and actual project timelines.

Reopen Percentage: The percentage of reopens offers a measure of the quality and effectiveness of the development and testing processes. A high percentage of reopens may indicate inefficiencies or deficiencies in these processes, such as inadequate testing coverage or incomplete development work. By monitoring this metric, teams can assess the impact of reopens on project timelines and resources, identify areas for improvement, and implement strategies to enhance overall issue resolution and quality assurance practices.

4.4 Dashboards

To facilitate easy access to specific information, these metrics are logically organized within the dashboards in Grafana. Consequently, four distinct dashboards were created, each corresponding to a specific DevOps tool mentioned in Section 3.2. The rationale for separation was an imposed requirement.

In the Section 4.2, the 'Source' column in both tables indicates the exact dashboard where each specific metric can be found.

Figure 4.2: Bitbucket Dashboard

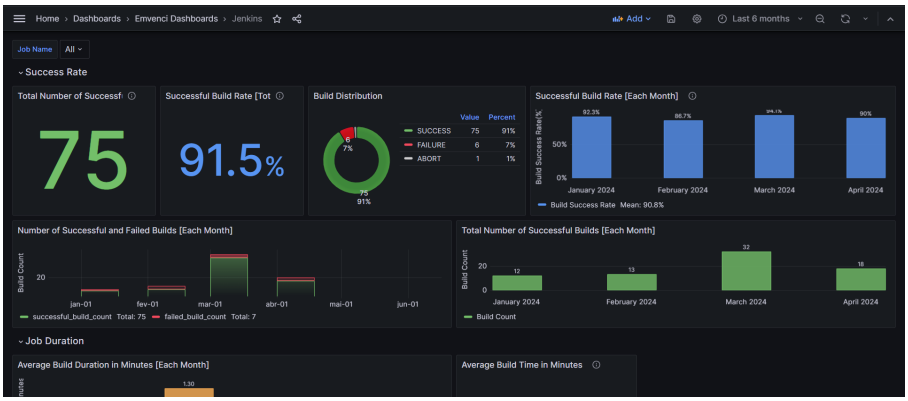

Figure 4.3: Jenkins Dashboard

Figures 4.2 and 4.3 illustrates some examples of the actual Grafana dashboards.

4.5 Security Aspects

When using a relatively new and open-source application that actively welcomes new contributors and accesses every internal tool, like it was done in this project, there are core security considerations to take into account. Most of the aspects are:

- **Machine Exposure:** The machine hosting the application is not exposed to the public and is strictly internal. However, a firewall is in place, and a SELinux profile is employed to enhance security, and it has restricted access.
- **Credential Permissions:** In DevLake, as it will be demonstrated, gathering data requires specifying the tool credentials. It is crucial to provide credentials with the least privilege necessary.
- **Data Encryption:** Since the project accesses PII, it is important to encrypt the database storing this data, even though the application is internal.
- **Authentication:** A requirement was the integration of Grafana with Lightweight Directory Access Protocol (LDAP) over Secure Sockets Layer (SSL). However, with the default configuration, it is possible to impersonate another user by simply changing one account attribute. Authentication validation must not rely on fields that the users can edit to identify users attempting to log in.
- **Secure Communication:** By default, these tools do not use HTTPS. Despite being an internal tool, HTTPS should be enabled to prevent the unencrypted transit of information.
- **Security Groups:** Not all users should have access to all dashboards. Security groups were identified, created, and assigned, and in Grafana, they were mapped to organizational units.

These aspects will be reinforced during implementation and sometimes explained with more detail, except for the machine exposure one.

Chapter 5

Implementation

This section explains all remaining steps taken to attain **Objective 3**, involving the implementation and configuration of DevLake and Grafana. Although the actual implementation order differed, this section presents the steps in the best sequence to replicate the process without jumping back and forth. This section not only covers the technical aspects, but also explains most of the security concerns mentioned in Section 4.5.

This chapter is composed of three sections:

- **DevLake Installation and Setup:** This section explains all the steps required to install and securely setup DevLake for gathering the data needed to create the metrics.
- **Grafana Setup:** This section details the configuration of Grafana to prepare it for securely displaying the dashboards containing the metrics and ensuring accessibility.
- **Metric Creation:** The final section discusses the process of creating a metric, including the critical thinking and logical aspects involved.

5.1 DevLake Installation and Setup

DevLake provides two methods to install DevLake, via Helm or Docker Compose. Given that the company already uses Docker and Docker Compose provides a quicker setup, the latter was used to install DevLake.

On every release, DevLake provides a pre-configured `docker-compose.yaml` file containing all necessary containers for its operation, and these are:

- `mysql`
- `grafana`
- `devlake-server`
- `config-ui`

In addition to the `docker-compose.yaml` file, a `.env` file is also provided. This file contains environment variables essential for the DevLake operation, such as the encryption secret used to secure information stored in the database. The encryption secret is set by the user, and DevLake recommends generating a 128-character string of uppercase letters using OpenSSL.

This key must be stored securely, as it is critical for encrypting sensitive information. If the key is lost, decrypting the information will be impossible. This key was stored in a Privileged Access Management (PAM) tool.

By default, the Config UI interface, used for generating API keys and setting up data source connections, is freely accessible. However, DevLake provides two environment variables that, when applied, whenever a user enters the DevLake Config-UI is prompted by the username and password that were inserted into those variables, adding security. These credentials were also stored on the PAM.

In the DevLake Docker Compose file, the environment variables for MySQL allow customization of the database name, root password, MySQL user, and MySQL password. It is a good practice to change these default values to improve security.

With this setup, it would already be possible to work with DevLake. However, during the implementation of DevLake, the company adopted Grafana for other specific needs, and security being the main concern, using Grafana while attached to DevLake would not be a good practice because Grafana version that is attached to DevLake is different from the last Grafana version and to ensure all the latest security updates, Grafana was separated from DevLake.

To dissociate Grafana from DevLake's Docker Compose setup, it was necessary to remove the container and volume configurations from DevLake Docker Compose and create a new one for Grafana. To avoid exposing the MySQL port, Grafana was connected to the DevLake Docker network. Additionally, to prevent incorrect redirects from DevLake Config-UI to Grafana, two environment variables were introduced in the Config-UI section of the Docker Compose file.

```
USE_EXTERNAL_GRAFANA: "true"
GRAFANA_ENDPOINT: "https://grafana.example.com"
```

The first variable is a flag to inform Config-UI that it is using an external Grafana. The second one is the URL of the external Grafana. Once the configuration is complete, use the Docker Compose command to run the setup. Ensure that the machine is listening to the application ports, and if a firewall is in place, it is necessary to add exceptions to allow access.

5.1.1 Configuring DevLake for HTTPS

To enable HTTPS communication for DevLake UI, which utilizes Nginx as its web server, modifications were made to its configuration. This process began by creating a Docker volume in the `docker-compose.yml` file. The volume was mapped to the Nginx configuration path `/etc/nginx/conf.d` in the Config UI service. To apply the volume, the container was restarted.

Within this configuration directory, a subdirectory named `ssl` was created to store the SSL certificate and key files.

The DevLake Nginx template configuration, `default.conf.tpl` was updated. This involved adding the `ssl` directive to the configuration and specifying the paths to the SSL certificate and key files to the server block.

After making these modifications, the Config UI service was restarted to apply the new Nginx configuration. This enabled the Nginx server to establish secure HTTPS communication.

5.1.2 Setup DevLake Connections

The first page of DevLake is the Connections page, as shown in Figure 5.1. This is where the connections to the DevOps tools are configured. There are various data sources, but as previously mentioned, it will only be: Bitbucket, Jenkins, Jira, and SonarQube.

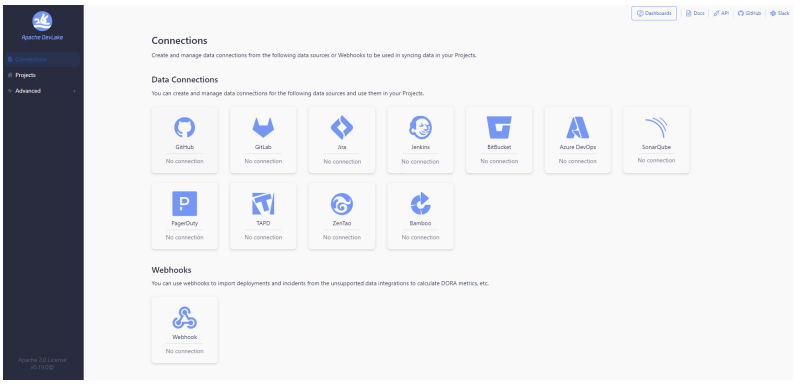

Figure 5.1: DevLake Config-UI Connections

When a DevLake data source is selected, a new page lists the existing connections. For the initial setup, a new connection is created by clicking a button below the list. This opens a form for setting up the connection, as shown in Figure 5.2 that happens to be for Bitbucket.

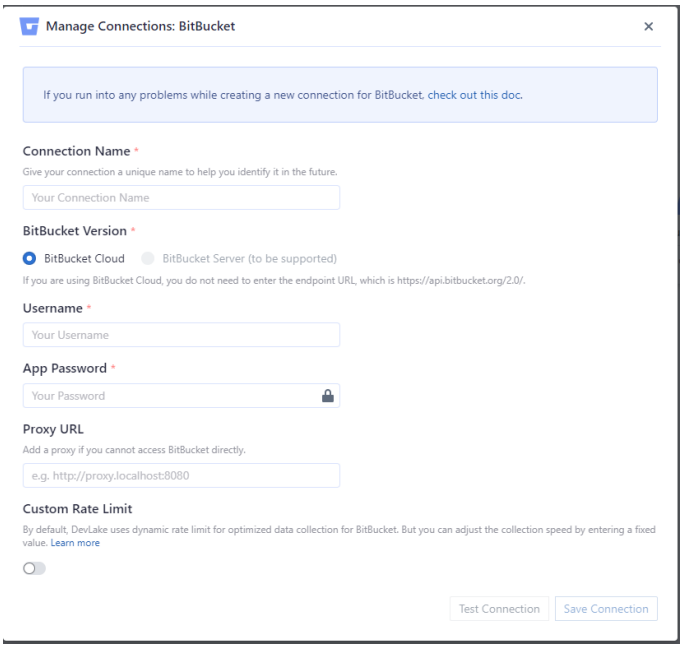

Figure 5.2: DevLake Config-UI Add Bitbucket Connection

As seen, the form contains five inputs, starting with the connection name. It is important to give connections meaningful names to easily distinguish them if more connections of the same type are added. The Bitbucket username is not the public name but rather the username under personal settings. For credentials, it is good practice to create an app password specifically for DevLake with minimal permissions, as recommended in the DevLake

documentation [1]. The proxy URL is optional and used if DevLake is behind a corporate firewall or VPN. A proxy server helps establish a connection to Bitbucket by routing traffic. Finally, the last input the Custom Rate Limit is another optional field that allows setting a dynamic rate limit for Bitbucket. When changing this, consider the API request limit of the data source [72].

After filling in all fields, test the connection to ensure it is ready to proceed.

In the event that a data source is self-managed and uses a self-signed certificate, it is necessary for the DevLake container to trust that certificate. Alternatively, if a certificate chain is used, the entire chain must be trusted to establish HTTPS connections. The current version does not support communication over insecure HTTPS connections. To trust a certificate chain, you need to copy the public keys of both the root and intermediary certificates to the SSL certificates directory, which is `/usr/local/share/ca-certificates`. Since DevLake is set up using Docker, the simplest method is to create a named volume and copy the certificates to that volume.

After creating a connection, the management page will show that no data scopes are associated. By clicking the button to add a data scope, a list of projects will appear to which the configured credentials have access. When selecting a project, it will also present the repositories within that project that the credentials have access to. This can be seen in Figure 5.3. In this case, the credentials have access to only one project "Emvenci", and only one repository "DevOps Testing". However, if the credentials had access to more projects and repositories, it would be possible to select multiple repositories from different projects.

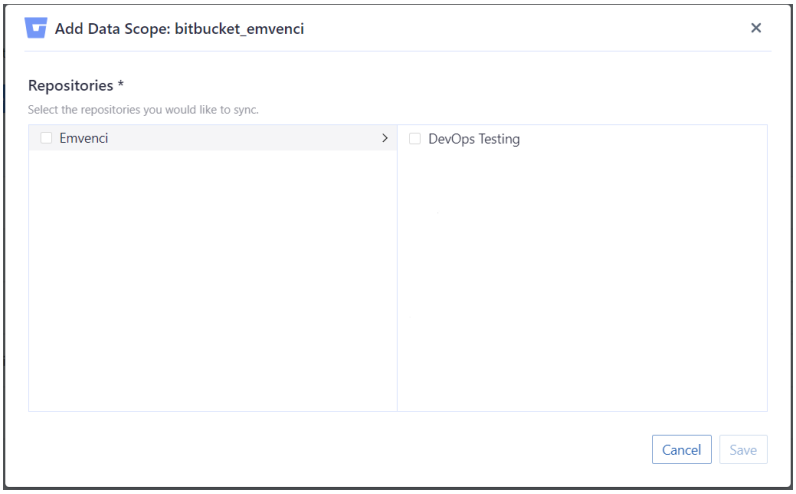

Figure 5.3: DevLake Config-UI Add Bitbucket Data Scope

After selecting the repositories, they will be listed on the connection management page. The "Scope Config" column shows "N/A" because a scope configuration is still needed. It is possible to create a scope configuration by selecting one or multiple projects and clicking the button to associate a scope configuration, then creating a new one. As shown in Figure 5.4, it is necessary to provide a scope configuration name and data entities. It is possible to have multiple data entities, with a minimum of one.

[1] If Bitbucket or any other data source adds new features and DevLake has not updated its documentation, use the least privilege principle.

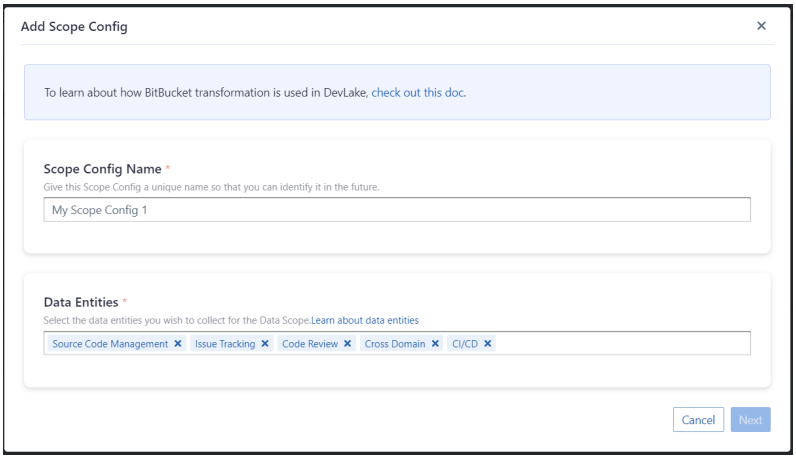

Figure 5.4: DevLake Config-UI Add Bitbucket Scope Configuration

For Bitbucket, after deciding the Data Entities, the connection will be fully set up. However, for other data sources, like Jira, additional transformations are necessary. Figure 5.5 shows the transformation needed for a Jira Board to complete the creation of a Scope Config.

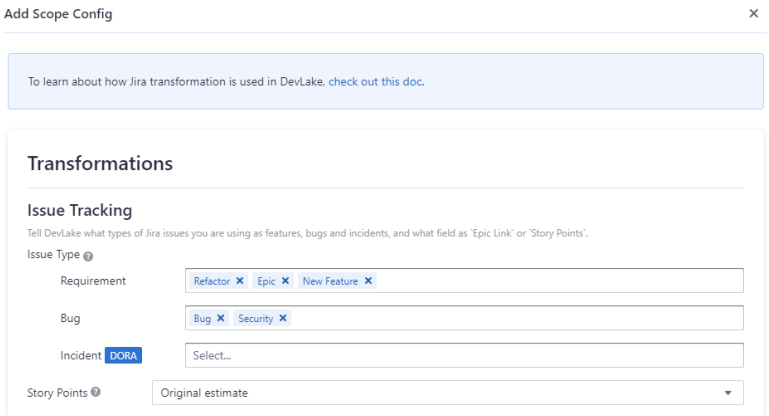

Figure 5.5: DevLake Config-UI Transformation

Transformation maps different issue types into three categories: Requirement, Bug, and Incident. The incident issue type is used to calculate some DORA metrics. In addition, it is necessary to specify the type of estimate being used. Even though it is not depicted in the figure, the actual used value is hours ("Original Estimate"), but it could also be story points or other units.

This covers all the steps for setting up a connection. These steps were repeated for all other DevOps tools used by the company. The next subsection will explain how to set up a blueprint or a project to begin collecting data.

5.1.3 Automate Data Gathering

Having configured connections, it is now possible to create Blueprints and/or Projects, this allows to setup a schedule to gather the data from the desired connections.

As mentioned in the previous section, a project represents a real-world project or product line and enables the collection of DORA metrics. Due to that, a project was created to

gather metrics rather than a blueprint. However, process of creating blueprints is similar, so the following steps can also be used to setup a blueprint.

In DevLake, projects have their own management page, where new projects can be created. When creating a project, there is a checkbox that when checked allows the calculation of DORA metrics for the project, if the necessary data is provided.

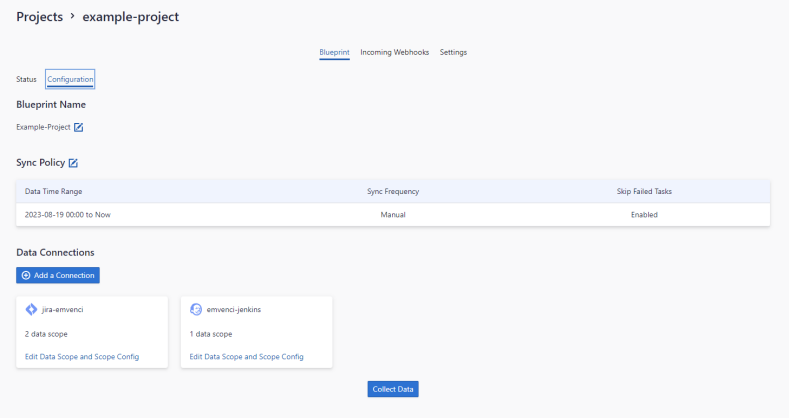

Figure 5.6: DevLake Project

After creating a project, connections can be associated with it. These associated connections will allow the project blueprint to collect data. As shown in Figure 5.6, the project already has two associated connections, one for Jira and one for Jenkins.

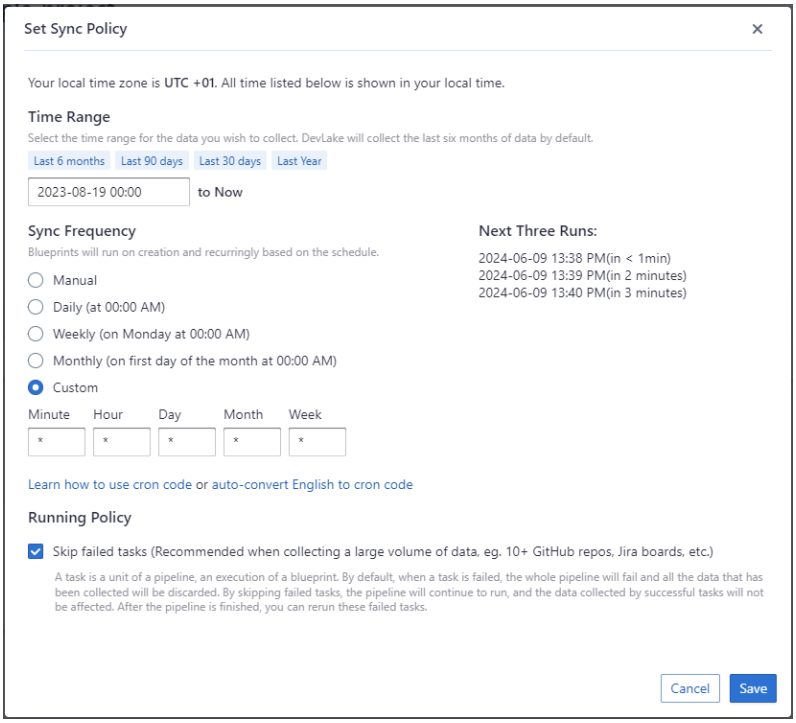

Figure 5.7: DevLake Project Synchronization Policy

Within the project, a synchronization policy can be defined to determine the frequency at which the DevLake runner executes to gather data. As depicted in Figure 5.7, configuring the synchronization policy requires specifying the time range, which is the interval of data

that the blueprint will collect, and the sync frequency, which is the defined schedule that the blueprint will run, the custom option allows defining a frequency based on cron code, which is the syntax typically used in cron expressions to schedule tasks. Additionally, to prevent other data connections from losing information, it is possible to skip failed tasks, allowing the pipeline to proceed with other tasks even if one fails.

With the data connections and a synchronization policy in place, it is possible to wait for the automation to run or manually run the process. When this process starts, a pipeline is triggered, which executes tasks to collect data from the different data connections that were established.

With data already gathered, predefined metrics can be examined, and specific ones can be built using Grafana.

5.2 Grafana Setup

This section explains every configuration applied to Grafana prior to the creation of dashboards and visualizations.

5.2.1 Configuring Grafana for HTTPS

By default, Grafana do not opearte via HTTPS. To enhance security, one of the first step involved configuring communication via HTTPS.

Starting by creating a named volume linked to the Grafana configuration file `grafana.ini` located at `/etc/grafana/`. Subsequently, either generate a self-signed certificate or obtain one from a certificate authority and move both the certificate and its corresponding key to the volume.

Next, navigate to `grafana.ini` and modify the configuration parameters specified as described in Appendix B.

After updating the configuration parameters with accurate information, restart Grafana container. Following this, Grafana will be accessible via HTTPS.

5.2.2 Customizing Grafana

As this tool will eventually be used by all employees within the company, it prompted consideration regarding customization options in Grafana to enhance its alignment with company branding. Specifically, there was a desire to integrate the company logo into the login page and modify background visuals and titles to create a more corporate feel.

By default, Grafana does not offer built-in functionality to customize the login page, icons, or titles. Nevertheless, through exploration of the Grafana source code and the use of `sed` command, it became possible to implement the desired alterations. This involved tasks such as replacing icons, updating the logo, adjusting the background, modifying titles, reviewing the login interface, and eliminating version indicators on footers.

It is important to consider that since this is not a built-in functionality on version updates, this may start working erroneously or even stop working at all. Apart from that, automating the process of customizing the Grafana on every version update was also important, so it was created a Dockerfile, visible in Appendix A, that contains all of these modifications.

If utilizing a Grafana image that is attached to DevLake, follow these steps:

1. Replace the image name in the `Dockerfile` with the DevLake Grafana image name.
2. Build a new image and give it a meaningful name.
3. Update the **docker-compose.yml** file by replacing the image name in the Grafana service section with the newly customized one.

In Figure 5.8, the modifications to the Grafana main login interface are evident.

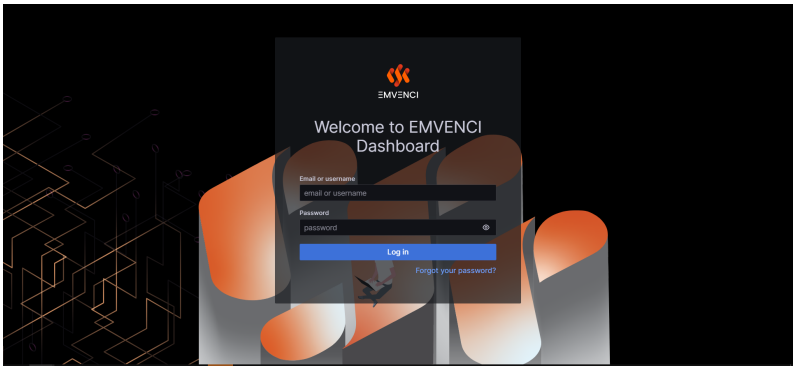

Figure 5.8: Customized Grafana Login

5.2.3 Grafana Organizations

An organization in Grafana helps isolate users and resources, such as dashboards and data sources, from others. The main purpose is to provide separate experiences, similar to multiple instances of Grafana, within a single instance. Managing multiple organizations is easier and cheaper than managing multiple instances of Grafana.

In this case, Grafana organizations were used as a workaround because the open-source version of Grafana does not support user mapping to teams, a feature available only in the enterprise version.

Three organizations were created: Administration, Backend, and Frontend.

Currently, only the Administration organization has dashboards, as the metrics are still stabilizing. However, as demonstrated in following sections, everything is pre-configured to create dashboards and select metrics for Backend and Frontend developers, who will have immediate access without the necessity of more configurations.

5.2.4 Grafana LDAP over SSL Integration

The company requested the integration of Grafana with the LDAP over SSL to centralize credentials, ensuring that all accounts and security groups are derived from a single authoritative source.

The first step to enable LDAP integration is to go to the `grafana.ini` file and enable LDAP just like it is shown in Appendix C.

After enabling LDAP, it is necessary to provide the actual configuration of LDAP to establish communication. That part is on `ldap.toml` file that is on the same path as the `grafana.ini` folder.

The first 23 lines of Appendix D are related to the setup of the connection. In this configuration, port 636 and SSL are used for secure communication. Additionally, since the LDAP server is signed with a intermediate certificate, both the root certificate and the client certificate are provided.

During the initial setup of LDAP, it is recommended to uncomment line 3 to receive more detailed logs related to LDAP. This makes it easier to identify and troubleshoot any configuration issues. After LDAP is working correctly, it is advisable to disable these logs because they expose some of the LDAP queries as well as DNS or IP addresses related to the LDAP server.

Next, the user bind is configured. This user, who performs LDAP queries on the server, must have administrator permissions. The distinguished name of the user is required for this parameter. Instead of storing the bind password directly in the code, it is stored in an environment variable. Storing the password in an environment variable helps protect it from being exposed in source code, reducing the risk of unauthorized access if the code is viewed or shared.

The search filter is the attribute that was used to search for users. For security reasons, the User Principal Name (UPN) is used, as it is not easily changed by users. Using the UPN helps prevent user impersonation because it is a unique identifier that typically remains constant, unlike other attributes such as email addresses or usernames, which can be changed more easily by users.

The search base domain name (DN) is the starting point for LDAP database searches.

Although group search filter parameters are usually commented, they are applied in this case because security groups can be nested within other groups. The application of the group search filter ensures an accurate retrieval of nested group memberships.

Finally, attributes are mapped between Grafana attributes and LDAP attributes.

Grafana in its UI allows testing of the configured authentication methods. It is possible to verify the LDAP connection and the user mapping that will be set up in the next step. Navigate to Administration > Authentication > LDAP. Note that the LDAP option will only be visible if it has been enabled previously.

After integrating Grafana with LDAP, the next step was to configure the access based on user security groups. However, since Role-Based Access Control (RBAC) is a Grafana Enterprise or Cloud feature, a workaround was needed.

Grafana's LDAP integration allows for mapping security groups using LDAP queries, but it can only map to organizational units. More granular mapping, such as to Grafana teams or dashboards, is a feature available only in the Enterprise or Cloud versions. Justifying the aforementioned organizations.

Four security groups have been established for users accessing Grafana.

- **grafana_admin**: This group has unrestricted access to all functionalities.
- **grafana_backend**: Members have access to the backend organizational unit and read privileges for all dashboards within it.
- **grafana_frontend**: Similar to Grafana backend but with access to the frontend organizational unit instead.

- **grafana_project_manager**: Granted visualization permissions for both frontend and backend organizational units.

In Appendix E, it is possible to observe the parameters that are needed to be filled in order to map the LDAP security groups to organizational units and roles.

- **group_dn:** Query to the path of the security group in question.
- **org_role:** The role of the users with the security group inside the organizational unit, can be: Viewer, Editor, or Admin.
- **grafana_admin:** When true, the users inside of this security group get Grafana server admin. A Grafana server admin has admin access over all organizations and users.
- **org_id:** allows the users with the security group to get access to the organizational unit with the specified id; if not specified, the default is 1.

5.2.5 Grafana Connections

Using the Grafana instance integrated with DevLake, the primary organization will already possess an established connection due to the configuration within the `docker-compose.yml` file. This setup includes four environment variables for the Grafana service, ensuring a default connection. However, if opting for an external Grafana instance or creating new organizations, it will be necessary to manually establish connections for each organization.

To set up a MySQL connection in Grafana:

1. Navigate to the connections management page by searching for **Connections**.
2. Search for and select **MySQL** as the data source.
3. Provide the following details:
    - **Host URL:** The connection string indicates the host and port. Since Docker is used and the MySQL port is not exposed, use `container-name:3306`.
    - **Database name:** The name of the database to query.
    - **Username:** The MySQL username, can be found in the Docker Compose file.
    - **Password:** The MySQL user password, available in the Docker Compose file.
4. Test the connection. If configured correctly, a message indicating that the database connection is successful will appear.

After establishing the connection, it is possible to start creating visualizations by querying the created data source.

5.2.6 Create Dashboards

Creating a dashboard requires logging into the application with a user with administrative permissions. Next, use the search bar to locate “Dashboards” and select “New Dashboard” This action will open a new dashboard where it is possible to create new visualizations, import an existing dashboard, or add a library panel as needed.

As previously explained, four dashboards were created, one for each DevOps tool that the company uses.

5.2.7 Variables

Grafana variables function as filters in visualizations, and any changes to them are reflected in the visualizations. To add a variable, go to the dashboards settings and select the option for variables from the left navigation bar. When creating the variable the following inputs need to be filled out:

- **Variable Type:** Type of query that would be used, can be many, but in this case only the "Query" type was used.
- **Name:** The name of the variable to be used in visualizations.
- **Label:** The display name for the variable.
- **Data Source:** The source to be queried.
- **Query:** The query that retrieves data.
- **Regex:** The regular expression to extract parts of the text.
- **Sort:** The method for sorting the variable's values.
- **Selection Options**: Enable multiselection of values, include all options, or both.

5.2.8 Visualization Panel

To create a visualization panel on a dashboard, there is a blue button with the label 'Add', and it is possible to create a new visualization.

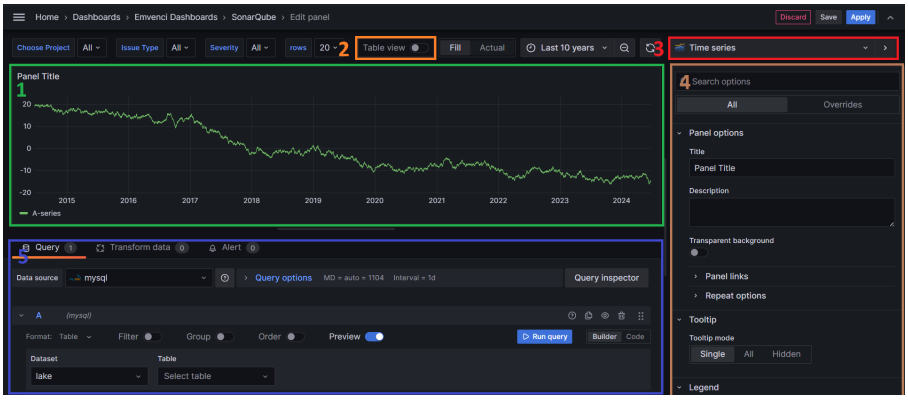

Figure 5.9: Visualization Creation Page

In Figure 5.9, demonstrates the page where a new visualization can be created, and each part is highlighted by a color and a number that will be referenced to explain:

- **Green and number 1:** This is where the result of the query will appear, either a graph or a table.
- **Orange and number 2:** During the query building phase, sometimes listing the values in a table view is useful, this is the option that will display them in that format.
- **Red and number 3:** This is where is possible to change the visualization type.
- **Brown and number 4:** It is possible to change the visualization panel title, description and other visualization specific options.

- **Blue and number 5:** Depending on the selected data source, this will change, It is possible to have multiple queries depending on the visualization type selected. On the bottom of this rectangle, when a query have errors, the error message is displayed. Some errors require access to Grafana logs to debug.

5.3 Metric Creation

Even though multiple metrics were created, this section will only explain three of them, each corresponding to a visualization type in Grafana:

- **Line Graph:** Average Time to Test Issues
- **Bar Graph:** Pull Request Size
- **Stat:** Vulnerabilities

This section demonstrates that the metrics were developed with critical thinking and a focus on simplicity and clarity. Although only three metrics are explained here, the same level of critical thinking was applied to all the metrics created.

5.3.1 Average Time to Test Issues

All metrics that included averages required careful consideration. For instance, when calculating the average time to test issues, it is crucial to decide whether to include all test iterations or just the last one, especially if multiple reopens occur. Excessively elaborate the metric by including all iterations can distort the results and make the analysis unnecessarily complex.

The primary goal of this metric is to evaluate the department's efficiency and identify areas for improvement. By considering only the last test iteration, the metric provides a clearer and more straightforward representation of the final testing phase before resolution. This approach simplifies the calculation and focuses on the most relevant data, ensuring that the analysis accurately reflects the department's performance and highlights opportunities for improvement.

The query operates on the issue table and focuses on measuring the average time it takes for issues to go from state "Testing" to the state "Done". To achieve this, it considers only the most recent transitions, thus excluding potential reopenings. This filtering is implemented through inner joins.

Additionally, the query allows for further refinement by filtering issues based on a dashboard variable that is the priority of the issue, which specifies the priority of the issues analyzed.

Regarding the output columns, the results are grouped by month, where each date is standardized to the first day of the month. The average amount of time taken is calculated in hours. To facilitate interpretation, the visualization configuration ensures that the output, initially in hours, is automatically converted into days or weeks when appropriate. The columns are only two to be able to be displayed in the time series visualization type.

To illustrate concretely, the query is possible to observe at Appendix F.

Figure 5.10, demonstrates how does a visualization of type line graph looks in Grafana.

Figure 5.10: Average Time to test Visualization

5.3.2 Pull Requests Size

When considering pull request size, it's important to determine which pull requests should be included. For instance, pull requests aimed at epic branches may not be relevant, whereas those targeting the main branch are. Additionally, commits made solely to synchronize branches that will be merged were filtered out so that would not hinder the average and mislead the person that is interpreting the metric.

Figure 5.11 illustrates what a bar chart visualization in Grafana looks like.

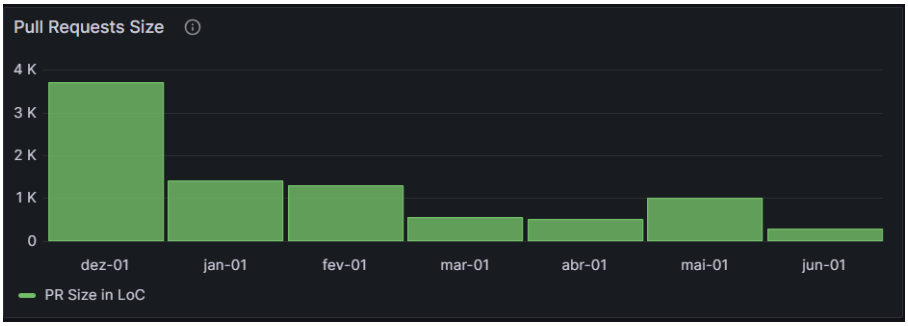

Figure 5.11: Pull Request Size Grafana Visualization

5.3.3 Vulnerabilities

This type of metric, which is only a number displayed, is the simplest; it simply queries the database with specific filters. The only thing to take into consideration is what filters can be applied to simplify the data visualization, and in this metric, it is possible to filter by repository and time range. This demonstrates that even some easy metrics can be gathered and yet be so useful.

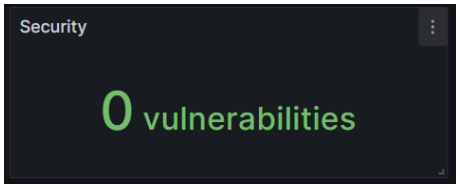

Figure 5.12: Vulnerabilities Grafana Visualization

Figure 5.12, depicts what a stat visualization looks like in Grafana.

Chapter 6

Case Studies

The process of gathering metrics for the software development process and operations has yielded a wealth of intriguing and valuable insights. This chapter explores some insights through a series of detailed case studies on data collected by the tool developed in this work. Each case study highlights important aspects of developers' efficiency and product quality. Remarkably, the timing of this implementation coincided with several changes in the company's management strategies, allowing for an immediate and practical assessment of these modifications using the newly gathered metrics.

Although the solution has not yet been fully integrated into the company's daily operations, it has occasionally been used to monitor progress and assess the impact of recent changes. The metrics span from November 2023 to April 2024, offering a comprehensive view of the period during which these strategic changes were made. By analyzing and correlating various data points, it is possible to derive significant insights into the effectiveness of these changes.

In this chapter, specific cases will be presented with brief introductions, followed by descriptions of the changes made and their impacts. These insights were discussed and confirmed during meetings with the project manager. The detailed analysis of these metrics reveals critical aspects of developer efficiency and product quality, providing a data-driven assessment of the company's recent management strategies and operational changes. Fulfilling the last **Objective 4**.

The case studies presented here are organized into sections, each focusing on different areas of improvement:

1. **Impact Assessment of Management Strategies**: This section evaluates the effectiveness of new management approaches by analyzing metrics such as the average time that issues stay in a raw state, the average time to plan and solve issues, and the number of reopens per month.
2. **Assessment of Development Optimization**: Here, it is explored how smaller pull requests and thorough reviews have improved the development process. Metrics such as pull request size, review depth, pull request status distribution, and code coverage are examined.
3. **Assessment on Enhanced Testing Efficiency**: This section discusses the impact of DevOps team improvements on testing efficiency, using metrics such as the number of bugs per month and the average time to test issues.

4. **Quick Response to High-Priority Bugs**: The company was assessed for its responsiveness to high-priority bugs by analyzing metrics such as the average time to plan, solve, and test high-priority issues.
5. **From Misclassification to Efficiency**: This case study highlights how metrics identified opportunities to enhance bug prioritization and classification, leading to improved issue management.
6. **DORA Metrics**: This case study consisted of comparing the responses to the DORA metrics mentioned in Chapter 3 and the values that were automatically collected by the system, with a focus on the frequency of deployment and the rate of failure of the change, to provide additional information on operational performance.

Each of these sections provides a detailed analysis of the relevant metrics, supported by visual data where appropriate, to illustrate the impact of the changes implemented. The case studies collectively demonstrate the power of data-driven decision-making in enhancing software development processes and achieving continuous improvement.

6.1 Impact Assessment of Management Strategies

In this section, the impact of recent modifications in management strategies on the development and planning efficiency is analyzed. The effectiveness of these changes is evaluated using the following key metrics:

- Average time issues stay in raw state
- Average time to plan issues
- Average time to solve issues
- Number of reopens per month

These metrics were selected because they not only provide a comprehensive view of the planning, execution, and quality assurance phases of the development process but were also the ones that clearly demonstrated the positive impact on the efficiency of developers and, consequently, the quality of the product.

6.1.1 Analysis and Discussion

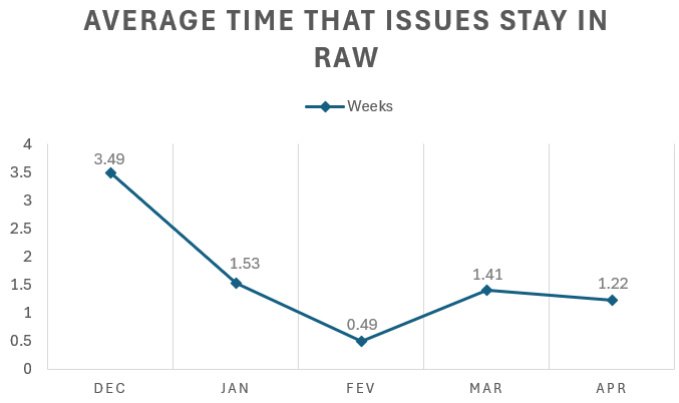

Figure 6.1: Average Time Issues Stay Raw

As shown in Figure 6.1, the average time that issues remain in raw state has significantly declined, meaning that the issues are being picked up faster. In December, the peak average was attributed to the Christmas holidays, but in the subsequent months, it decreased until it reached the lowest value of 0.49 weeks, or roughly 3.5 days. In March, it increased but remained lower than in January, and in April, it decreased once more, indicating quicker issue pick up. This demonstrates an improvement in management's planning efficiency, suggesting a more effective strategy for task delegation and issue prioritization.

Issues are picked up more quickly, but it is also important to determine if they are being planned faster. A quick glance at Figure 6.2 reveals that they are indeed planned faster. In December and January, the planning time for issues was longer. The peak in December can be attributed, once again, to the Christmas holidays, while in January it was slightly lower but still high. This may be due to the planning of older issues, resulting in a higher value. For the remaining months, the values decreased until March and April, where they seemed to stabilize, maintaining nearly the same value for two consecutive months.

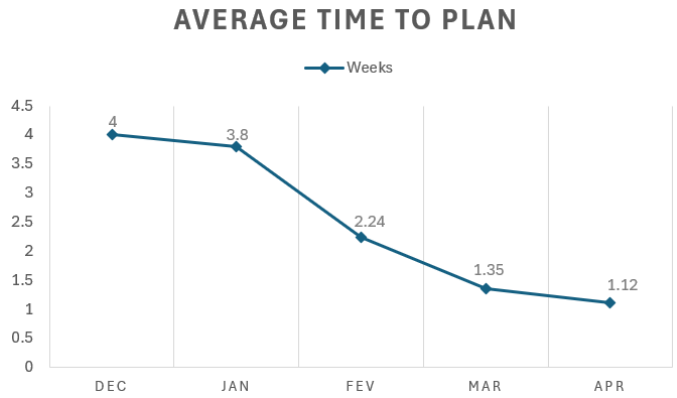

Figure 6.2: Average Time to Plan

After examining the planning side, another important aspect is if the faster planning is affecting the development efficiency, and Figure 6.3 demonstrates that it is positively impacting the development. Notably, there was a sharp decline from December to January, once again, because of Christmas vacations, followed by a gradual reduction until reaching a stable level. These improvements can be credited to a refined strategy of breaking down larger issues into smaller and more manageable tasks.

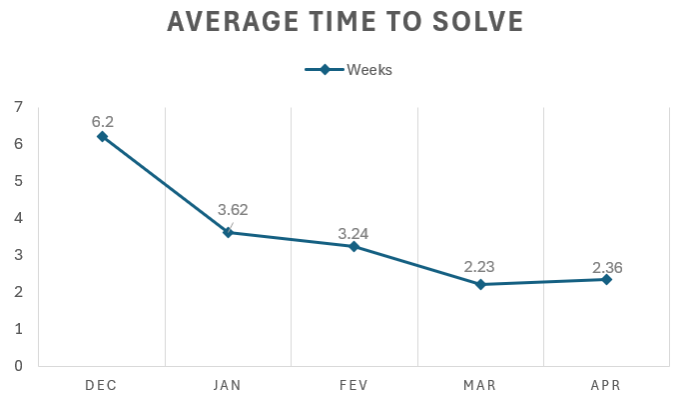

Figure 6.3: Average Time to Solve

This approach not only accelerated the planning and resolution processes but also enhanced the clarity and focus of each task. The metric that highlights the benefit of this approach is the number of reopens per month. There has been a noticeable decrease in reopens, indicating that issues are being better understood and executed correctly the first time. This suggests that the enhanced planning phase is providing developers with clearer and more detailed requirements, reducing the need for revisions.

6.1.2 Final Notes

Together, these metrics demonstrate a positive trend in planning and development efficiency. Data consistently show declining trends in time-related metrics and reopens, validating the effectiveness of management strategies.

6.2 Assessment of Development Optimization

The modifications made to improve developers' efficiency are closely tied to changes in the management team's approach. To evaluate the effectiveness of these alterations, the following metrics were analyzed:

- Pull Request Size
- Pull Request Review Depth
- Pull Requests Status Distribution
- Number of Reopens per Month
- Code Coverage

The management team implemented rules that resulted in working in smaller pull requests, this case study will demonstrate that if the developers adopted the practice and if it was beneficial.

6.2.1 Analysis and Discussion

As depicted in Figure 6.6, the size of the pull requests is decreasing, with December being the highest month and decreasing each month. This decrease comproves that in fact the developers started working in smaller batches.

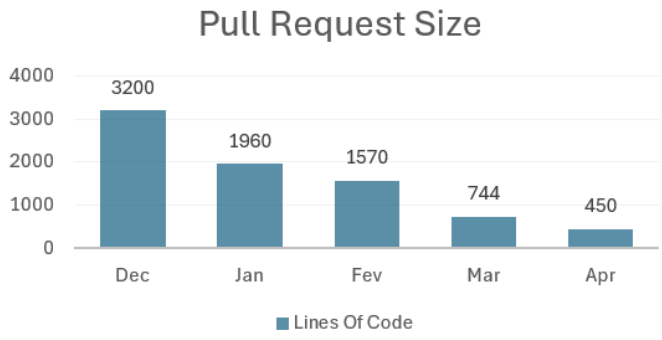

Figure 6.4: Pull Request Size Graph

The Pull Request Status distribution demonstrates that more pull requests are being merged, indicating that with a smaller size of pull requests, more work gets integrated into the code base, this is observable in Figure 6.5. By correlating the pull request status and pull request size, it is possible to observe that the smaller the average size, the higher the number of pull requests that are integrated.

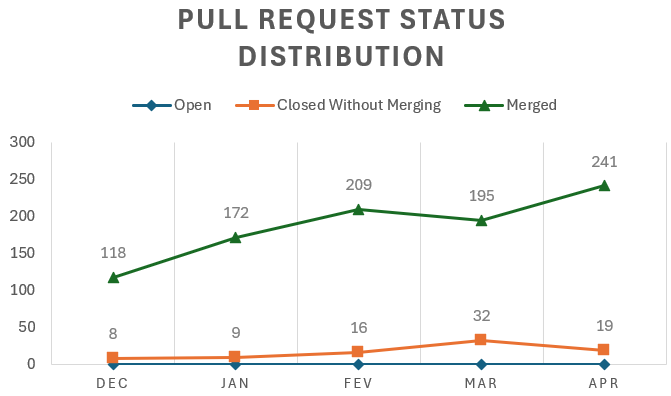

Figure 6.5: Pull Request Status Distribution

Despite the smaller size of the pull requests, as shown in 6.6, the review depth has not significantly decreased, indicating that developers remain committed to high standards and ensuring code quality. And by correlating again with the size of the pull request, both graphs showed a spike in December.

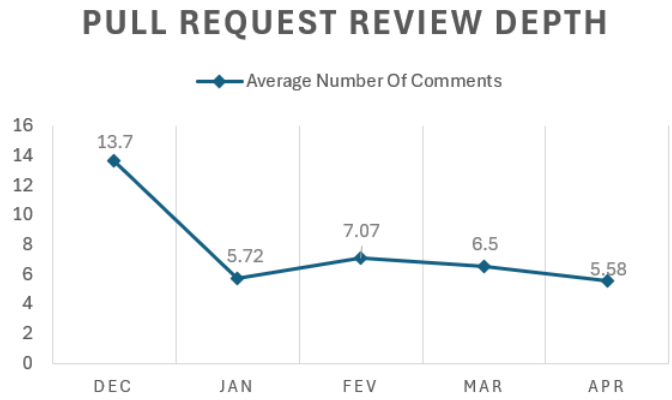

Figure 6.6: Pull Request Review Depth Graph

Although it is still early to definitively assess the impact on the number of bugs—since bugs from recent changes might only surface in the coming months— it was possible to observe a reduction in the number of reopens per month. This can be attributed to the previously mentioned change and the smaller scope of changes in each pull request, which allows reviewers to focus more effectively. With fewer changes to analyze, reviewers experience less fatigue and can examine each modification more thoroughly, leading to the identification of more issues during the code review phase before the code proceeds to testing.

Another change was the definition of a threshold for the minimum test coverage required for new code. This policy has led to an increase in overall code coverage, enhancing the quality and reliability of the final product. Higher test coverage ensures that more code is

being tested, which helps in catching potential issues early and improving the robustness of the software.

6.2.2 Final Notes

Overall, these metrics demonstrate that the changes implemented have positively impacted the development process. The transition to smaller increments and the emphasis on testing have led to more efficient workflows, improved code quality, and a decrease in rework. While long-term effects on bug counts will need further observation, the early indicators are promising. Furthermore, it was possible to effectively demonstrate the correlation between smaller pull requests and the number of merging pull requests.

6.3 Assessment on Enhanced Testing Efficiency

The following metrics were analyzed to assess the effectiveness of the modifications made to the infrastructure by the DevOps team:

- Number of bugs per month
- Average time to test issues

This case study analyzed the impact of the creation of multiple testing environments for the testing team.

6.3.1 Analysis and Discussion

The infrastructure improvements have significantly impacted testing efficiency, as reflected in the following metrics.

In Figure 6.7 demonstrates the graph of the average time to test issues metric has also been positively affected. With the flexibility of testing in various environments, testers can conduct more efficient and effective testing processes. This has resulted in a quicker turnaround for testing tasks, enhancing the overall productivity of the testing team.

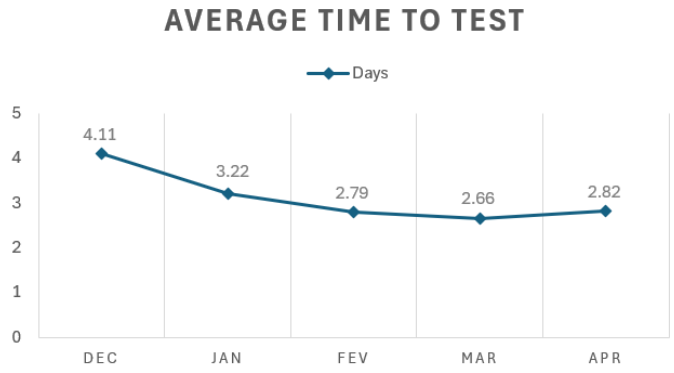

Figure 6.7: Pull Request Review Depth Graph

With more efficient and effective testing, the number of bugs per month metric demonstrated that in the month of implementation there was an increase of bug findings.

6.3.2 Final Notes

The positive trends in testing efficiency demonstrated the effectiveness of the enhanced infrastructure, contributing to higher product quality.

6.4 Quick Response to High-Priority Bugs

To assess the response to high-priority bugs, the following metrics were analyzed:

- Average Time to Plan (Filtered by Highest Priority)
- Average Time to Solve Issues (Filtered by Highest Priority)
- Average Time to Test (Filtered by Highest Priority)

6.4.1 Analysis and Discussion

These metrics revealed that the company has a fast response to the high-priority bugs. Data from April shows that the Average Time to Plan, Solve, and Test high-priority issues is approximately four days. This finding encompasses the entire process from planning through solution development to testing, demonstrating a prompt response to urgent problems.

The average time to plan, filtered by high-priority bugs, is decreasing and was already at a good values. This indicates that high-priority bugs are quickly identified and prioritized for action, and that the management team is becoming increasingly adept at handling these types of issues.

Similarly, the Average Time to Solve Issues filtered by high-priority bugs is minimized, reflecting the company's commitment to rapidly developing solutions for the most severe problems. This quick turnaround in solving issues demonstrates the efficiency and focus of the development team when dealing with high-priority bugs.

The Average Time to Test high-priority issues also remains low, but yet the stage that takes longer, showcasing the testing team's interest to verify fixes for high-priority bugs.

6.4.2 Final Notes

Overall, these metrics highlight the company's effective and timely response to high-priority bugs. Despite the inherent challenges and potential risks associated with high-priority bugs, the company's ability to rapidly plan, solve, and test these issues demonstrates a robust and responsive approach to maintaining software quality and security. Notably, the team takes on average of four days to resolve these complex issues, a timeframe that includes planning, development, and thorough testing. Given the high complexity of these bugs and the rigorous quality assurance required to ensure they are fully resolved, this four-day turnaround demonstrates their fast response. Additionally, it is important to note that such high-priority bugs do not occur every month, further underscoring the team's efficiency and readiness to tackle these rare but impactful challenges.

6.5 From Misclassification to Efficiency

While not all observations are positive, it is important to showcase how metrics can identify areas for improvement. This case study examines how metrics highlighted opportunities to enhance the prioritization and classification of bugs.

The metrics analyzed were:

- Average Age of Bugs by Priority
- Raw Issue Distribution

6.5.1 Analysis and Discussion

The Average Age of Bugs by Priority metric provided valuable insights. In March, it was observed that the time for issues marked as highest priority was higher than expected. This aroused the management team, and they investigated and refined the prioritization process to ensure that the issues were correctly classified.

The Raw Issue Distribution metric also offered important information. In November, there are a high number of issues that were not planned. This coincided with the onboarding of new developers, which temporarily shifted the focus of the management team towards integrating new team members. As a result, the issues created in that month started to be left behind.

6.5.2 Final Notes

The management team took proactive steps to review the issues that were created in that month and improve the classification and prioritization process. They reviewed and adjusted the priority levels of issues, ensuring that the most high-priority bugs received immediate attention. Additionally, clear guidelines were provided to enhance the accuracy of bug classification.

6.6 DORA Metrics

In Section 3.1, were presented the results of the DORA quick check. Now, let's observe the two metrics that it were possible to automatically gather using the tool:

- Deployment Frequency
- Change Failure Rate

These metrics are derived from the performance of our production pipeline. In short, Deployment Frequency measures the successful executions of the production pipeline, while Change Failure Rate tracks instances of pipeline failure.

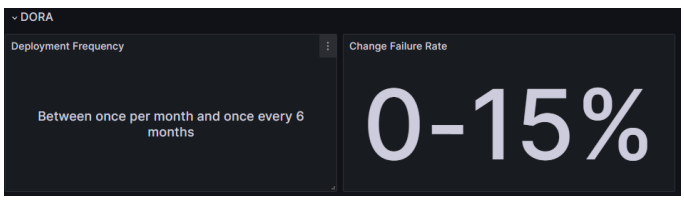

Figure 6.8: DORA metrics

The deployment frequency metric aligns with the response provided in Section 3.1, indicating a frequency ranging from once per month to once every six months. Similarly, the Change Failure Rate falls within the range of 0% to 15%. This consistency between the values collected and previously reported reflects a clear understanding of current operational performance. Both values are possible to observe in Figure 6.8.

6.7 Case Studies Conclusion

In this chapter, a series of case studies were exposed with the objective of evaluating the efficacy of various management strategies and operational changes within our software development process. Through the lens of key metrics, it was possible to assess how these initiatives impacted efficiency, productivity, and overall product quality.

Each case study provided valuable information on specific areas for improvement. From assessing management strategies to enhancing testing efficiency and addressing misclassification issues, uncovering opportunities to improve processes and drive continuous improvement.

By analyzing metrics such as the average time to resolve issues, the number of bugs per month, and the average age of bugs by priority, it was possible to quantify the impact of these initiatives and track progress over time. The data revealed tangible improvements, including but not only fewer reopens, and enhanced testing efficiency.

Furthermore, the integration of DORA metrics provides additional information on operational performance, highlighting the deployment frequency and the rate of change failure. The alignment of the company employees and the company current practices.

These case studies underscore the importance of using metrics as a guiding force in our decision-making process. By using data-driven insights to assess strategies and initiatives, it is also possible to identify areas for improvement, implement targeted solutions, and ultimately drive greater efficiency and product quality in software development efforts. As the company continues to iterate and refine their processes, they remain committed to embracing a culture of continuous improvement and excellence.

Chapter 7

Conclusion and Future Work

This final chapter presents a summary of the limitations encountered during this study, suggests directions for future work, and concludes the document. Contains three sections:

- **Conclusion:** The final section recaps all the key aspects and outcomes of the project.
- **Limitations:** This section presents the constraints encountered during the project.
- **Future Work:** This section identifies potential directions for further development and improvement.

7.1 Conclusion

This document addresses the common challenge companies face in gathering data from distributed development and operations tools. Centralizing this data is crucial, as it enables the creation of more insightful and valuable metrics. This, in turn, allows companies to benchmark their performance against competitors and identify areas for improvement, helping them maintain their competitiveness in the market.

During the state of the art analysis, information was gathered about the capabilities adopted by high-performing companies that implement DevOps. These capabilities, categorized into technical, process, and cultural aspects, totaled 18 capabilities, addressing **RQ1**. Additionally, research focusing on **RQ2** identified nine key isolated metrics aligning with the primary goal. Furthermore, DORA metrics were introduced and explained due to their significant impact in assessing companies, offering insights for improvement and benchmarking against the market. Two case studies using DORA metrics demonstrated valuable insights, particularly the Capital One case, which showed a 30 percent increase in deployment frequency to production without affecting the change failure rate. Finally, four tools were identified and briefly analyzed, revealing that almost every data gathering tool for metrics comes with a cost, addressing **RQ3**. These research results allowed to achieve **Objective 1**.

The first approach towards the solution was analyzing the organizational environment, presenting the four tools they mostly use, fulfilling **Objective 2**. With the necessary knowledge and understanding of the organizational environment, it became evident that the versatility and open-source nature of DevLake was the right choice. Regarding metrics, 30 were presented, most of which were tailor-made for the company's context. With the tool and metrics selected, **Objective 3** was partially achieved. The project involved installing and setting up the tool, considering security aspects due to the sensitive nature of the data. This setup allowed the automation of data gathering and the display of metrics organized into four different dashboards, finally achieving the remaining part of **Objective 3**. Finally,

to address **Objective 4** six case studies were conducted, with most of them showing that recent management changes led to a positive trend in planning and development efficiency.

Although the company has not yet fully adopted the tool during the project, it is clear that shifting from subjective evaluations to data-driven decision-making will significantly impact the company. Utilizing solid metrics reduces the risks tied to uncertainty and ambiguity, promoting a culture of responsibility and effectiveness.

As previously demonstrated, all research questions identified were answered and all objectives were fulfilled.

7.2 Limitations

Despite the project's success, there were some limitations. Certain metrics could not be gathered because they required the company to adapt its workflow, which was not feasible during the project's timeline. For instance, calculating some DORA metrics would necessitate introducing a new issue type called "Incident". When such an issue is created, DevLake interprets that an incident has occurred, and once the issue is closed it interprets that the incident is resolved. Furthermore, despite the considerable potential of DevLake there are still problems such as incorrect mapping of user emails and Bitbucket IDs and account replication in the database. While workarounds were conceivable, pursuing them it would also not be viable.

7.3 Future Work

As soon as the metrics stabilize, a new phase of the project can be initiated where specific metrics are made available to the developers, Grafana is already configured, so this phase can initiate faster. The metrics should be carefully selected to avoid misleading the developers. As metrics become accessible to developers, they will enhance their sense of ownership and responsibility, improve communication, and align team goals with broader organizational objectives, thus contributing to collective achievement.

Additionally, as the tool gets updated, it is expected to be possible to create more metrics. By continually integrating potentially useful metrics, the company can have even better support for decision-making processes. This iterative approach will help maintain the tool's relevance and effectiveness in meeting the dynamic needs of developers and the organization as a whole.

Bibliography

[1] S. Bobrovskis and A. Jurenoks. "A Survey of Continuous Integration, Continuous Delivery and Continuous Deployment". In: *BIR Workshops* (2018). Accessed: 2024-06-23. url: https://www.researchgate.net/publication/354720705_A_STUDY_AND_ANALYSIS_OF_CONTINUOUS_DELIVERY_CONTINUOUS_INTEGRATION_IN_SOFTWARE_DEVELOPMENT_ENVIRONMENT.

[2] Nasreen Azad and Sami Hyrynsalmi. "DevOps critical success factors — A systematic literature review". In: *Information and Software Technology* 157 (May 2023), p. 107150. issn: 0950-5849. doi: 10.1016/J.INFSOF.2023.107150. (Visited on 06/23/2024).

[3] Barbara Kitchenham and Stuart Charters. "Guidelines for performing Systematic Literature Reviews in Software Engineering". In: 2 (Jan. 2007). (Visited on 06/23/2024).

[4] Leonardo Leite et al. "A Survey of DevOps Concepts and Challenges". In: *ACM Computing Surveys (CSUR)* 52 (6 Nov. 2019). issn: 15577341. doi: 10.1145/3359981. url: https://dl.acm.org/doi/10.1145/3359981 (visited on 06/23/2024).

[5] Welder Pinheiro Luz, Gustavo Pinto, and Rodrigo Bonifácio. "Adopting DevOps in the real world: A theory, a model, and a case study". In: *Journal of Systems and Software* 157 (Nov. 2019), p. 110384. issn: 0164-1212. doi: 10.1016/J.JSS.2019.07.083. (Visited on 06/23/2024).

[6] Ramtin Jabbari et al. "What is DevOps? A systematic mapping study on definitions and practices". In: *ACM International Conference Proceeding Series* 24-May-2016 (May 2016). doi: 10.1145/2962695.2962707. url: https://dl.acm.org/doi/10.1145/2962695.2962707 (visited on 06/23/2024).

[7] Dulani Meedeniya, Iresha Rubasinghe, and Indika Perera. "Traceability Establishment and Visualization of Software Artefacts in DevOps Practice: A Survey". In: *International Journal of Advanced Computer Science and Applications* 10 (July 2019), pp. 66–76. doi: 10.14569/IJACSA.2019.0100711. (Visited on 06/23/2024).

[8] Mayank Gokarna. "DevOps phases across Software Development Lifecycle". In: (Jan. 2021). doi: 10.36227/techrxiv.13207796.v2. (Visited on 06/23/2024).

[9] Mali Senapathi, Jim Buchan, and Hady Osman. "DevOps Capabilities, Practices, and Challenges: Insights from a Case Study". In: *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. EASE '18. Christchurch, New Zealand: Association for Computing Machinery, 2018, pp. 57–67. isbn: 9781450364034. doi: 10.1145/3210459.3210465. url: https://doi.org/10.1145/3210459.3210465 (visited on 06/23/2024).

[10] Nicole Forsgren, Jez Humble, and Gene Kim. *Accelerate: The Science of Lean Software and DevOps Building and Scaling High Performing Technology Organizations*. 1st. IT Revolution Press, 2018. isbn: 1942788339. (Visited on 06/23/2024).

[11] Ricardo Amaro, Ruben Pereira, and Miguel Mira Da Silva. "Capabilities and Practices in DevOps: A Multivocal Literature Review". In: *IEEE Transactions on Software Engineering* 49 (2 Feb. 2023), pp. 883–901. issn: 19393520. doi: 10.1109/TSE.2022.3166626. (Visited on 06/23/2024).

[12] Nazatul Nurlisa Zolkifli, Amir Ngah, and Aziz Deraman. "Version Control System: A Review". In: *Procedia Computer Science* 135 (Jan. 2018), pp. 408–415. issn: 1877-0509. doi: 10.1016/J.PROCS.2018.08.191. (Visited on 06/23/2024).

[13] Mark Stillwell and Jose G.F. Coutinho. "A DevOps approach to integration of software components in an EU research project". In: *1st International Workshop on Quality-Aware DevOps, QUDOS 2015 - Proceedings* (Sept. 2015), pp. 1–6. doi: 10.1145/2804371.2804372. url: https://dl.acm.org/doi/10.1145/2804371.2804372 (visited on 06/23/2024).

[14] Google Cloud. *DORA | DevOps Capabilities: Code Maintainability*. url: https://dora.dev/devops-capabilities/technical/code-maintainability/ (visited on 06/23/2024).

[15] Victoria Holt et al. "The usage of best practices and procedures in the database community". In: *Information Systems* 49 (2015), pp. 163–181. issn: 0306-4379. doi: https://doi.org/10.1016/j.is.2014.12.004. url: https://www.sciencedirect.com/science/article/pii/S0306437914001914 (visited on 06/23/2024).

[16] DORA. *DORA | DevOps Capabilities: Database Change Management*. url: https://dora.dev/devops-capabilities/technical/database-change-management/ (visited on 06/23/2024).

[17] Paul Hammant. *Trunk Based Development*. https://trunkbaseddevelopment.com/. visited on 2024-06-23. 2020.

[18] Nicole Forsgren et al. *State of DevOps Report 2017 | Puppet by Perforce*. 2017. url: https://www.puppet.com/resources/history-of-devops-reports#2017 (visited on 06/23/2024).

[19] Google Cloud. *DORA | DevOps Capabilities: Trunk-based Development*. url: https://dora.dev/devops-capabilities/technical/trunk-based-development/ (visited on 06/23/2024).

[20] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices". In: *IEEE Access* PP (Mar. 2017). doi: 10.1109/ACCESS.2017.2685629. (Visited on 06/23/2024).

[21] Fiorella Zampetti et al. "An empirical characterization of bad practices in continuous integration". In: *Empirical Software Engineering* 25 (2 Mar. 2020), pp. 1095–1135. issn: 15737616. doi: 10.1007/S10664-019-09785-8/TABLES/13. url: https://link.springer.com/article/10.1007/s10664-019-09785-8 (visited on 06/23/2024).

[22] S. Rossel. *Continuous Integration, Delivery, and Deployment: Reliable and faster software releases with automating builds, tests, and deployment*. Packt Publishing, 2017. isbn: 9781787284180. url: https://books.google.pt/books?id=1xhKDwAAQBAJ (visited on 06/23/2024).

[23] AWS. *What is CI? - Continuous Integration Explained - AWS*. url: https://aws.amazon.com/devops/continuous-integration/ (visited on 06/23/2024).

[24] Google Cloud. *DORA | DevOps Capabilities: Monitoring and Observability*. 2023. url: https://dora.dev/devops-capabilities/technical/monitoring-and-observability/ (visited on 06/23/2024).

[25] Lucy Ellen Lwakatare et al. "DevOps in practice: A multiple case study of five companies". In: *Information and Software Technology* 114 (Oct. 2019), pp. 217–230. issn: 0950-5849. doi: 10.1016/J.INFSOF.2019.06.010. (Visited on 06/23/2024).

[26] Huan Zhou et al. "CloudsStorm: A framework for seamlessly programming and controlling virtual infrastructure functions during the DevOps lifecycle of cloud applications".

In: *Software: Practice and Experience* 49 (10 Oct. 2019), pp. 1421–1447. issn: 1097-024X. doi: 10.1002/SPE.2741. url: https://onlinelibrary.wiley.com/doi/full/10.1002/spe.2741 (visited on 06/23/2024).

[27] Peter Mell and Timothy Grance. "The NIST Definition of Cloud Computing". In: (Sept. 2011). doi: 10.6028/NIST.SP.800-145. url: https://csrc.nist.gov/pubs/sp/800/145/final (visited on 06/23/2024).

[28] DORA. *DORA | DevOps Capabilities: Documentation quality*. https://dora.dev/devops-capabilities/process/documentation-quality/. Visited on 2024-06-23.

[29] Steve Mansfield-Devine. "DevOps: finding room for security". In: *Network Security* 2018 (7 July 2018), pp. 15–20. issn: 1353-4858. doi: 10.1016/S1353-4858(18)30070-9. (Visited on 06/23/2024).

[30] Mojtaba Shahin, Ali Rezaei Nasab, and Muhammad Ali Babar. "A qualitative study of architectural design issues in DevOps". In: *Journal of Software: Evolution and Process* 35 (5 May 2023), e2379. issn: 2047-7481. doi: 10.1002/SMR.2379. url: https://onlinelibrary.wiley.com/doi/full/10.1002/smr.2379%20https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2379%20https://onlinelibrary.wiley.com/doi/10.1002/smr.2379 (visited on 06/23/2024).

[31] Sikender Mohsienuddin Mohammad. "An Exploratory Study of Devops and It's Future in the United States". In: (Nov. 2016). issn: 2320-2882. url: https://papers.ssrn.com/abstract=3668439 (visited on 06/23/2024).

[32] Mojtaba Shahin and M. Ali Babar. "On the role of software architecture in DevOps transformation: An industrial case study". In: *Proceedings - 2020 IEEE/ACM International Conference on Software and System Processes, ICSSP 2020* 10 (June 2020), pp. 175–184. doi: 10.1145/3379177.3388891. url: https://dl.acm.org/doi/10.1145/3379177.3388891 (visited on 06/23/2024).

[33] Ron Westrum. "A Typology of Organisational Cultures". In: *Quality safety in health care* 13 Suppl 2 (Jan. 2005), pp. ii22–7. doi: 10.1136/qhc.13.suppl_2.ii22. (Visited on 06/23/2024).

[34] DORA. *DORA | Well-being*. url: https://dora.dev/devops-capabilities/cultural/well-being/ (visited on 06/23/2024).

[35] Kent State University. "Software Metrics Software Engineering". In: (). (Visited on 06/23/2024).

[36] Mirco Hering, Dominica Degrandis, and Nicole Forsgren. "Measure Efficiency, Effectiveness, and Culture to Optimize DevOps Transformation". Oct. 2015. url: https://itrevolution.com/product/measure-efficiency-effectiveness-and-culture-to-optimize-devops-transformation/ (visited on 06/23/2024).

[37] Ricardo Amaro, Rúben Pereira, and Miguel Mira Da Silva. "Capabilities and metrics in DevOps: A design science study". In: *Information Management* 60 (2023), p. 103809. doi: 10.1016/j.im.2023.103809. url: https://doi.org/10.1016/j.im.2023.103809 (visited on 06/23/2024).

[38] Plandek. *Build Failure Rate: Definition, Related Metrics Use Cases*. url: https://plandek.com/blog/build-failure-rate/ (visited on 06/23/2024).

[39] Qentelli. *Average Build Duration* -. url: https://knowledgebase.qentelli.com/knowledge-base/average-build-duration/ (visited on 06/23/2024).

[40] Qentelli. *Lines of Code* -. url: https://knowledgebase.qentelli.com/knowledge-base/lines-of-code/ (visited on 06/23/2024).

[41] Mario Fernandez. *What Is Code Duplication? A Definition and Overview | Dev Interrupted Powered by LinearB*. Feb. 2021. url: https://linearb.io/blog/code-duplication (visited on 06/23/2024).

[42] Apache DevLake. *Code Quality Duplicated Blocks | Apache DevLake - Open-Source Dev Data Platform for Productivity*. url: https://devlake.apache.org/docs/Metrics/CQDuplicatedBlocks (visited on 06/23/2024).

[43] Software. *Pull Request Frequency | DevOps Metrics*. url: https://www.software.com/devops-guides/pull-request-frequency (visited on 06/23/2024).

[44] Qentelli. *Code Coverage -*. url: https://knowledgebase.qentelli.com/knowledge-base/code-coverage/ (visited on 06/23/2024).

[45] seyedhamidreza$_m$*etrics*$_t$*hesis*. *DevOps Metrics Objectives and Key Results*. 2022. url: https://hdl.handle.net/10292/15688 (visited on 06/23/2024).

[46] DevLake. *Code Quality Maintainability-Debt | Apache DevLake - Open-Source Dev Data Platform for Productivity*. url: https://devlake.apache.org/docs/Metrics/CQMaintainability-Debt (visited on 06/23/2024).

[47] LogicMonitor. *Uptime vs. Availability | LogicMonitor*. url: https://www.logicmonitor.com/blog/uptime-vs-availability (visited on 06/23/2024).

[48] Google Cloud. *DORA | DevOps Research and Assessment*. 2023. url: https://dora.dev/ (visited on 06/23/2024).

[49] Roberto Pietrantuono et al. "Towards continuous software reliability testing in DevOPs". In: *Proceedings - 2019 IEEE/ACM 14th International Workshop on Automation of Software Test, AST 2019* (May 2019), pp. 21–27. doi: 10.1109/AST.2019.00009. url: https://dl.acm.org/doi/10.1109/AST.2019.00009 (visited on 06/23/2024).

[50] Atlassian. *Atlassian Survey 2020 - DevOps Trends | Atlassian*. 2020. url: https://www.atlassian.com/whitepapers/devops-survey-2020 (visited on 06/23/2024).

[51] Gene Kim and Jez Humble. *2013 State of DevOps Report*. 2013. url: https://www.puppet.com/resources/history-of-devops-reports#2013 (visited on 06/23/2024).

[52] Kevin Storer et al. "State of DevOps Report 2023". In: (2023). (Visited on 06/23/2024).

[53] Nigel Kersten. *History of DevOps Reports | Puppet by Perforce*. en. https://www.puppet.com/resources/history-of-devops-reports. Visited on 2024-06-23. 2021.

[54] Jez Humble. *DORA's Journey: An Exploration. DevOps Research and Assessment (DORA)*. https://medium.com/@jezhumble/doras-journey-an-exploration-4c6bfc41e667. Visited on 2024-06-23. Feb. 2019.

[55] DORA Team. *DORA | Dora Joins Google Cloud*. Dec. 2018. url: https://dora.dev/news/dora-joins-google-cloud/ (visited on 06/23/2024).

[56] Nicole Dr. Forsgren, Jez Humble, and Gene Kim. *State of DevOps 2018*. 2018. url: https://services.google.com/fh/files/misc/state-of-devops-2018.pdf (visited on 06/23/2024).

[57] A. Srividya, Durga Rao Karanki, and Gopika Vinod. *Advances in RAMS Engineering In Honor of Professor Ajit Kumar Verma on His 60th Birthday*. Ed. by A. Srividya, Durga Rao Karanki, and Gopika Vinod. 1st ed. 2020. Springer International Publishing, 2020. isbn: 9783030365189. doi: 10.1007/978-3-030-36518-9. (Visited on 06/23/2024).

[58] Michelle Irvine, Dave Stanke, and Nathen Harvey. "State of DevOps 2021 Accelerate". In: (2021), pp. 10–10. (Visited on 06/23/2024).

[59] DORA. "Capital One Drives Continuous Delivery Improvement with Insights from DORA Executive Summary Increase in Number of Releases 20x No Increase in Incidents". In: (Apr. 2017). (Visited on 06/23/2024).

[60] Cassidy Horton and Lauren Graves. *Largest Banks In The U.S. 2023 – Forbes Advisor*. Aug. 2023. url: https://www.forbes.com/advisor/banking/largest-banks-in-the-us/ (visited on 06/23/2024).

[61] Capital One. *Capital One Credit Cards, Bank, and Loans - Personal and Business*. url: https://www.capitalone.com/ (visited on 06/23/2024).

[62] Marc Sallin et al. "Measuring Software Delivery Performance Using the Four Key Metrics of DevOps". In: *Lecture Notes in Business Information Processing* 419 LNBIP (2021), pp. 103–119. issn: 18651356. doi: 10.1007/978-3-030-78098-2_7. (Visited on 06/23/2024).

[63] Apache DevLake. *Apache DevLake - Open-Source Dev Data Platform for Productivity | Apache DevLake - Open-Source Dev Data Platform for Productivity*. url: https://devlake.apache.org/ (visited on 06/23/2024).

[64] Code Climate. *Software Engineering Intelligence | Code Climate*. url: https://codeclimate.com/ (visited on 06/23/2024).

[65] *LinearB: Software Delivery Management*. url: https://linearb.io/ (visited on 06/23/2024).

[66] Swarmia. *Code Climate Velocity vs. Swarmia | Swarmia*. url: https://www.swarmia.com (visited on 06/23/2024).

[67] Google Cloud. *DORA | DORA Quick Check*. url: https://dora.dev/quickcheck/ (visited on 06/23/2024).

[68] Atlassian. *Welcome to Jira | Atlassian*. url: https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software (visited on 06/23/2024).

[69] Atlassian. *Bitbucket Overview | Bitbucket*. url: https://bitbucket.org/product/guides/getting-started/overview#a-brief-overview-of-bitbucket (visited on 06/23/2024).

[70] klesh. *Release v0.2.0 · apache/incubator-devlake · GitHub*. url: https://github.com/apache/incubator-devlake/releases/tag/v0.2.0 (visited on 06/23/2024).

[71] Sonar. *What is a Code Smell? Definition Guide Examples | Sonar*. url: https://www.sonarsource.com/learn/code-smells/ (visited on 06/23/2024).

[72] Atlassian. *API request limits | Bitbucket Cloud | Atlassian Support*. Accessed at 2024-06-23. url: https://support.atlassian.com/bitbucket-cloud/docs/api-request-limits/.

Appendix A

Grafana Dockerfile

```
FROM grafana/grafana-oss:10.3.3

## Change icon
COPY fav32.png /usr/share/grafana/public/img
COPY fav32.png /usr/share/grafana/public/img/apple-touch-icon.png

## Replace Logo
COPY logo.svg /usr/share/grafana/public/img/grafana_icon.svg

## Update Background
COPY background.svg /usr/share/grafana/public/img/g8_login_dark.svg
COPY background.svg /usr/share/grafana/public/img/g8_login_light.svg
COPY background.svg /usr/share/grafana/public/img/login_background_dark.
    svg
COPY background.svg /usr/share/grafana/public/img/login_background_light
    .svg

USER 0

## Update Title
RUN sed -i 's|<title>\[\[.AppTitle\]\]</title>|<title>EMVENCI Dashboard
    </title>|g' /usr/share/grafana/public/views/index.html

## Update Title
RUN find /usr/share/grafana/public/build/ -name *.js -exec sed -i 's|
    AppTitle="Grafana"|AppTitle="EMVENCI Dashboards"|g' {} \;

## Update Login Title
RUN find /usr/share/grafana/public/build/ -name *.js -exec sed -i 's|
    LoginTitle="Welcome to Grafana"|LoginTitle="Welcome to EMVENCI
    Dashboard"|g' {} \;

## Remove Documentation, Support, Community in the Footer
RUN find /usr/share/grafana/public/build/ -name *.js -exec sed -i 's|\[{
    target:"_blank",id:"documentation".*grafana_footer"}\]|\[\]|g' {} \;

## Remove Edition in the Footer
RUN find /usr/share/grafana/public/build/ -name *.js -exec sed -i 's|({
    target:"_blank",id:"license",.*licenseUrl})|()|g' {} \;

## Remove Version in the Footer
RUN find /usr/share/grafana/public/build/ -name *.js -exec sed -i 's|({
    target:"_blank",id:"version",.*CHANGELOG.md":void 0})|()|g' {} \;

## Remove News icon
```

```
37 RUN find /usr/share/grafana/public/build/ -name *.js -exec sed -i 's|..
       createElement(....,{className:.,onClick:.,iconOnly:!0,icon:"rss","
       aria-label":"News"})|null|g' {} \;
38
39 ## Remove Open Source icon
40 RUN find /usr/share/grafana/public/build/ -name *.js -exec sed -i 's|.
       push({target:"_blank",id:"version",text:`${..edition}${.}`,url:..
       licenseUrl,icon:"external-link-alt"})||g' {} \;
41
42 USER grafana
43
44 EXPOSE 3000
```

Listing A.1: Grafana customized Dockerfile

Appendix B

Grafana HTTPS Configuration

```
[server]
;http_addr = #the ip addres to bind to, leaving this commented or empty
    will bind to all interfaces
http_port = 3000
domain = mysite.com #The public facing domain name used to access
    grafana from a browser
protocol = https
root_url = %(protocol)s://%(domain)s:%(http_port)s/ #dynamically created
     root_url
cert_key = /etc/grafana/grafana.key
cert_file = /etc/grafana/grafana.crt
enforce_domain = false # Redirect to correct domain (Prevents DNS
    rebinding attacks)
```

Listing B.1: Grafana HTTPS parameters

Appendix C

Grafana enable LDAP

```
[auth.ldap]
# Set to `true` to enable LDAP integration (default: `false`)
enabled = true

# Path to the LDAP specific configuration file (default: `/etc/grafana/ldap.toml`)
config_file = /etc/grafana/ldap.toml

# Allow sign-up should be `true` (default) to allow Grafana to create users on successful LDAP authentication.
# If set to `false` only already existing Grafana users will be able to login.
allow_sign_up = true
```

Listing C.1: Grafana enable LDAP parameters

Appendix D

Grafana LDAP TOML

```
# To troubleshoot and get more log info enable ldap debug logging in grafana.ini
# [log]
# filters = ldap:debug

[[servers]]
# Ldap server host (specify multiple hosts space separated)
host = "dns-or-ip"
# Default port is 389 or 636 if use_ssl = true
port = 636
# Set to true if LDAP server should use an encrypted TLS connection (either with STARTTLS or LDAPS)
use_ssl = true
# If set to true, use LDAP with STARTTLS instead of LDAPS
start_tls = false
# The value of an accepted TLS cipher. By default, this value is empty. Example value: ["TLS_AES_256_GCM_SHA384"])
tls_ciphers = []
# This is the minimum TLS version allowed. By default, this value is empty. Accepted values are: TLS1.1, TLS1.2, TLS1.3.
min_tls_version = ""
# set to true if you want to skip ssl cert validation
ssl_skip_verify = false
# set to the path to your root CA certificate or leave unset to use system defaults
root_ca_cert = "path-to-root-ca"
# Authentication against LDAP servers requiring client certificates
client_cert = "path-to-client-public-certificate"

# search user bind_dn
bind_dn = "cn=admin,dc=grafana,dc=org"
# search user bind password
bind_password = '$__env{LDAP_BIND_PASSWORD}'

# Timeout in seconds (applies to each host specified in the 'host' entry (space separated))
timeout = 10

# User search filter, for example "(cn=%s)" or "(sAMAccountName=%s)" or "(uid=%s)"
search_filter = "(UserPrincipalName=%s)"

# An array of base dns to search through
search_base_dns = ["dc=grafana,dc=org"]

# The group search filter
```

```
group_search_filter = "(member:1.2.840.113556.1.4.1941:=CN=%s,CN=Users,
    DC=grafana,DC=org)"
group_search_base_dns = ["dc=grafana,dc=org"]
group_search_filter_user_attribute = "cn"

# Specify names of the ldap attributes your ldap uses
[servers.attributes]
name = "givenName"
surname = "sn"
username = "cn"
member_of = "memberOf"
email =  "userPrincipalName"
```

Listing D.1: Grafana LDAP configuration

Appendix E

Grafana Group Mapping

```
[group_mapping]
# Map ldap groups to grafana org roles
[[servers.group_mappings]]
group_dn = "cn=grafana_admin,ou=security groups,dc=grafana,dc=org"
org_role = "Admin"
grafana_admin = true

[[servers.group_mappings]]
group_dn = "cn=grafana_manager,ou=security groups,dc=grafana,dc=org"
org_role = "Viewer"
org_id = 2

[[servers.group_mappings]]
group_dn = "cn=grafana_manager,ou=security groups,dc=grafana,dc=org"
org_role = "Viewer"
org_id = 3

[[servers.group_mappings]]
group_dn = "cn=grafana_backend,ou=security groups,dc=grafana,dc=org"
org_role = "Viewer"
org_id = 3

[[servers.group_mappings]]
group_dn = "cn=grafana_frontend,ou=security groups,dc=grafana,dc=org"
org_role = "Viewer"
org_id = 2
```

Listing E.1: Grafana Group Mapping

Appendix F

Query Average Time to Test

```
SELECT
    DATE_ADD(DATE(change_test_date), INTERVAL -DAYOFMONTH(DATE(
    change_test_date)) + 1 DAY) AS month,
    AVG(TIMESTAMPDIFF(HOUR, change_test_date, change_done_date)) AS "
    Average Time to Test Issues"
FROM
    lake.issues
-- Last time the issue went to testing (Ignoring reopens)
INNER JOIN (
    SELECT
        issue_id,
        MAX(issue_changelogs.created_date) AS change_test_date
    FROM
        lake.issue_changelogs
    WHERE
        field_name = 'status'
        AND original_to_value = 'Testing'
    GROUP BY
        issue_id,
        original_to_value
) AS last_testing_occurrence ON issues.id = last_testing_occurrence.
    issue_id
-- Last time issue went to done (Ignoring possible reopens)
INNER JOIN (
    SELECT
        issue_id,
        MAX(issue_changelogs.created_date) AS change_done_date
    FROM
        lake.issue_changelogs
    WHERE
        field_name = 'status'
        AND original_to_value = 'Done'
    GROUP BY
        issue_id,
        original_to_value
) AS done_occurence ON issues.id = done_occurence.issue_id
WHERE issues.priority IN ($priority_name)
GROUP BY month;
```

Listing F.1: Query Average Time to Test