

Aalto University  
School of Electrical Engineering  
Master's programme in ICT Innovation

De Paoli Alessio Juan

# **Automating Log Analysis and Management in Continuous Integration for Improved Efficiency**

Master's Thesis  
Espoo, July 30, 2024

Supervisor: Professor Jukka Manner  
Advisor: Pauli Maki-Pollari M.Sc.

<b>Author:</b>	De Paoli Alessio Juan	
<b>Title:</b>	Automating Log Analysis and Management in Continuous Integration for Improved Efficiency	
<b>Date:</b>	July 30, 2024	<b>Pages:</b> 59
<b>Major:</b>	Cloud and Network Infrastructure	<b>Code:</b> ELEC3059
<b>Supervisor:</b>	Professor Jukka Manner	
<b>Advisor:</b>	Pauli Maki-Pollari M.Sc.	
<b>Abstract</b>	<p>The work proposed in this Master's Thesis explores how automation technologies can address manual inefficiencies from managing logs in a Continuous Integration (CI) process.</p> <p>Specifically, the study aims to understand how automation can improve efficiency and time utilization for software development teams. By focusing on minimizing human work in processing logs and adding automation for Continuous Delivery, and providing insights into implementing automated solutions to streamline CI/CD pipelines.</p> <p>The contribution of the thesis is actually fully described as code and the entire release process is driven by company-internal repositories and a CI/CD pipeline.</p>	
<b>Keywords:</b>	Continuous Integration, Continuous Delivery, Automation, Log Analysis, Software Development, DevOps	
<b>Language:</b>	English	

# Acknowledgements

I would like to express my gratitude to all those who have supported and guided me throughout this journey.

First and foremost, I am profoundly grateful to my supervisor, Professor Jukka Manner, for his invaluable guidance, encouragement, and insightful feedback.

I also want to thank my team at Ericsson for their cooperation, assistance, and shared knowledge, which have greatly enriched this work.

To my friends, Davide and Giorgia, thank you for your continuous encouragement, for listening, and for being there through the ups and downs.

Lastly, to my girlfriend, Anna, your patience, understanding, and support during this period have meant the world to me. Your encouragement have been my anchor, and I am truly grateful for your presence in my life.

This thesis would not have been possible without each and every one of you. Thank you.

Espoo, July 30, 2024

De Paoli Alessio Juan

# Abbreviations and Acronyms

SDI	Software Defined Infrastructure
CI	Continuous Integration
CD	Continuous Delivery
DevOps	Combines software development with IT operations
IaC	Infrastructure as Code
UI	User Interface
XML	Extensible Markup Language
BDS	Big Data System
NLP	Natural Language Processing
LSTM	Long Short-Term Memory
DSL	Groovy based domain-specific language
YAML	YAML Ain't Markup Language
API	Application Programming Interface
HTTPErrors	Hypertext Transfer Protocol Errors
JIRA	Software based on agile methodology

# Contents

<b>Abbreviations and Acronyms</b>	<b>4</b>
<b>1 Introduction</b>	<b>9</b>
1.1 Problem statement . . . . .	10
1.2 Results . . . . .	11
1.3 Structure of the Thesis . . . . .	11
<b>2 DevOps Theoretical Foundation</b>	<b>13</b>
2.1 DevOps . . . . .	14
2.1.1 Overview and Key Concept . . . . .	14
2.1.2 Architecture . . . . .	16
2.1.3 Methologies . . . . .	16
2.1.3.1 Tools Landscape . . . . .	17
2.1.3.2 Metrics . . . . .	18
2.1.4 Benefit, Challenge and Limitation . . . . .	18
2.2 CI/CD . . . . .	20
2.2.1 Overview and Impact . . . . .	20
2.2.2 Methologies . . . . .	21
2.2.2.1 Tools . . . . .	21
2.2.3 Benefit, Challenge and Limitation . . . . .	23
2.3 Logging . . . . .	25
2.3.1 Overview . . . . .	25
2.3.2 Methologies . . . . .	26
2.3.2.1 Best practises in implementing automation for log processing . . . . .	26
2.3.2.2 Overview of serviced used in a software devel- opment environment . . . . .	27
2.3.3 Benefit, Challenge and Limitation . . . . .	28
2.4 Summary . . . . .	30

<b>3</b>	<b>Case Study</b>	<b>31</b>
3.1	Overview of environment . . . . .	31
3.1.1	Importance of CI in Ericsson's Operations . . . . .	31
3.1.2	CI Process at Ericsson . . . . .	32
3.1.3	Type of logs . . . . .	33
3.2	Manual inefficiency in log analysis . . . . .	33
3.3	Problem statement . . . . .	35
3.3.1	Methodology . . . . .	35
3.3.1.1	Interviews and Survey . . . . .	35
3.3.2	Data collection analysis . . . . .	36
3.3.2.1	Survey analysis . . . . .	37
3.3.2.2	Interviews analysis . . . . .	38
3.3.2.3	Goal . . . . .	41
3.4	Similar research on the topic . . . . .	41
3.5	Summary . . . . .	43
<b>4</b>	<b>Automation into existing CI/CD</b>	<b>44</b>
4.1	Machine Claiming overview . . . . .	44
4.1.1	Usage . . . . .	45
4.1.2	Rules . . . . .	46
4.2	Automation in error detection and analysis . . . . .	47
4.2.1	Tracking of the past issues . . . . .	47
4.2.2	Automatic JIRA ticket creation . . . . .	47
4.3	Summary . . . . .	49
<b>5</b>	<b>Result and Analysis</b>	<b>50</b>
5.1	Manual vs Automated . . . . .	51
5.2	Impact of automation into existing CI/CD . . . . .	52
5.2.1	Data comparison . . . . .	53
5.3	Summary . . . . .	54
<b>6</b>	<b>Conclusions</b>	<b>55</b>
6.1	Future Development . . . . .	56

# List of Tables

2.1	Lifecycle Overview . . . . .	15
2.2	DevOps Architecture [1] . . . . .	16
2.3	Benefit and Challenges . . . . .	19
2.4	Benefits Overview . . . . .	23
3.1	Automation tools for DevOps in Ericsson . . . . .	33
5.1	Time saved in CI analysis after implementation of automation . . . . .	53

# List of Figures

1.1	DevOps and CI/CD [2]	10
2.1	CI/CD: The relationship between continuous integration, delivery and deployment [3]	20
2.2	The overall process of software logging [4]	26
3.1	Estimation of time spent on CI analysis	37
3.2	Problems on our CI	38
5.1	Estimation of time spent on CI analysis after implementation	52

# Chapter 1

## Introduction

In the world of software development, things move quickly. Developers are constantly writing new code and updating existing applications. To stay competitive, they need to get these changes to users fast without sacrificing quality. That's where DevOps comes in a set of practices that combines software development (Dev) with IT operations (Ops), cooperating, increasing the speed, and improving the quality of software development. Development and operations are integrated with DevOps, which supports the quick and adaptable development and delivery of business processes. This integrated strategy delivers value more quickly and consistently, by streamlining communication, reducing errors, and expediting problem resolution.

DevOps has been widely adopted by industry heavyweights like Amazon and Google, who have achieved incredibly quick cycle times, demonstrating its practicality in the real world. DevOps must be customized to circumstances and product architectures rather than being limited to just one delivery methodology. It is possible to plan and deploy upgrades rapidly and reliably by using DevOps principles even in situations with constraints, such safety-critical systems. Furthermore, the integration of DevOps in software development and IT operations marks a significant advancement in how we approach software creation and maintenance. [5]

One of the key components of DevOps is CI/CD, which stands for Continuous Integration and Continuous Delivery. It provides high levels of automation that are at the core of DevOps. CI involves automatically integrating and testing code changes from multiple developers in a shared repository. This practice helps in detecting and resolving conflicts early, reducing the risk of major integration issues, and maintaining high software quality. Tools like Jenkins are commonly used to facilitate CI, allowing developers to automatically run tests and merge changes effectively. CD, on the other hand, extends CI by automating the software delivery process, ensuring that software can be reliably released at any time. Its main goal

is to automate the processes involved in deploying and testing software updates into production. Making deployments routine, predictable, and able to be carried out on demand is the aim of CD. When developers make changes, the latest version is automatically deployed to consumers after going through a set of tests to ensure everything functions as intended. In order to manage complicated software projects more effectively and dependably and to guarantee that software quality is maintained throughout the development lifecycle, CI/CD procedures are essential to modern software development. [6]

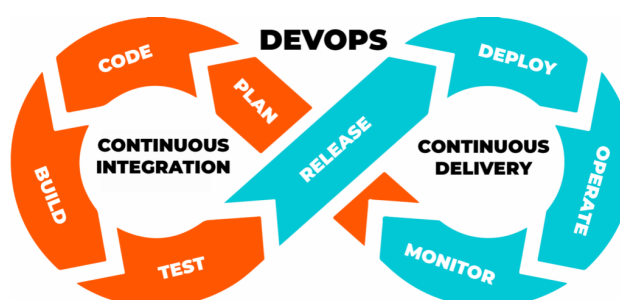


Fig. 1.1: DevOps and CI/CD [2]

## 1.1 Problem statement

Our focus is in the testing part, but something is wrong. Despite all this automation, a lot of data is still produced by the process, particularly logs. The result at the test stage of the CI/CD process are documented in these logs. They are crucial since they enable teams to ensure that everything is on track and assist in identifying the root cause of issues. The problem is that going through these logs by hand is not nearly as enjoyable as it may seem. The existing state of our CI process reveals a lack of automation on the analysis of the results, particularly in overseeing and managing the logs of daily and weekly loops. The manual actions involved in this process consume a significant amount of the team's time, leading to inefficiencies. Also could be that the same error occurs in different test cases or that there is a flaky test, a test that gives inconsistent results, in the system. This is the application of the research question of the thesis. Examining the logs from our test cycles, The goal of the thesis is to see how we might improve this manual slog. The thesis's main question is How to minimize human work in daily and weekly logs from CI process? If successful, it might allow the team to concentrate on more interesting tasks and expedite the release of software.

Why then is this relevant? Logs are essential in CI/CD pipelines for monitoring software development processes, quickly identifying and addressing issues, and maintaining overall software quality and efficiency. Manual intervention is a common part of the traditional log processing method, which can be laborious and prone to human error. This is especially difficult when handling a lot of log data because it might get difficult to recognize important problems and comprehend their context. We have around 60 lops that run daily, so the development cycle can be considerably slowed down by manual log processing, which might have an impact on the general efficacy and efficiency of software delivery. So, sustaining software quality and operational efficiency in software development requires efficient log management. Processes for software development and deployment can be speed up and made more dependable by automating log processing inside a CI/CD pipeline. [7]

We're not simply saving time by automating more of the CI/CD process, we are also improving our software and increasing the enjoyment of our team's work. We can summarize the research question, based on the above, as:

- How to minimize human work in the processing of daily and weekly logs from CI?

## 1.2 Results

In this thesis, we explore the implementation of automation within the CI log analysis process, which has led to significant outcomes. The primary achievement of our initiative is a reduction of 20 minutes per day in the time spent on log analysis, equating to approximately 2.2% of the total workload. While this time saving may appear modest, it accumulates to a substantial reduction over longer periods, freeing up valuable hours that can be redirected towards more critical and complex tasks within the software development lifecycle. This decrease in manual task duration has not only boosted team productivity but also qualitatively enhanced the working conditions and satisfaction of the development team by making their work less boring and repetitive.

## 1.3 Structure of the Thesis

This thesis is structured to provide a comprehensive overview of the implementation and impact of automation in the Continuous Integration (CI) log analysis process. Chapter 2 introduces the foundational concepts of Continuous Integration, Continuous Delivery, DevOps, and Logging, establishing the necessary context for understanding the subsequent discussions. Chapter 3 presents the case

study, offering an in-depth look at the operational environment and highlighting the problem of manual inefficiency that motivated this research. In Chapter 4, we delve into the implementation of automation within the existing Ericsson CI, detailing the methodologies and technologies employed. Chapter 5 then discusses the results and analyzes the impact of this automation, addressing the research question and evaluating the benefits achieved. Finally, Chapter 6 concludes the research by summarizing the findings and discussing potential avenues for future development, including the integration of AI-based systems and further automation capabilities.

## Chapter 2

# DevOps Theoretical Foundation

This chapter provides a summary of all the pertinent data required to understand the project's scope and the application's environment.

In particular, there are three sections that make up the background information provided in this chapter.

Section 2.1 provides background information, focusing on DevOps and its methodologies.

Section 2.2 presents the theoretical basis for the field's most crucial topic, CI/CD, beginning with an overview, and then presenting the modern methodologies.

Lastly, Section 2.3 presents the system of logging and their methodologies.

The information in this chapter helps to clarify the problem statement that was emphasized in the previous chapter. Specifically, the introduction of modern ways to automate software development processes aspects improves the comprehension of the issue and specifies the actions required to create a successful implementation, as detailed in the upcoming chapters.

## 2.1 DevOps

DevOps in modern software development, emphasising its role in the integration of development and operations, aims to achieve greater efficiency and quality in the software delivery process. It is described as a development methodology that tries to reduce the gap between development and operations. It emphasizes communication and collaboration, continuous integration, quality assurance, and delivery with automated deployment using a set of development practices. [8]

However, DevOps is more than just a set of rules or methods for developing software. It's a flexible and active way of working that helps organizations quickly adapt and succeed in today's rapidly changing environment of creating and managing software. The primary goal of DevOps is to expedite the creation, testing, and reliable release of software deliverables. It is characterized by a continuous and collaborative process, often symbolized by infinite loops, to represent the interconnectedness and repetitiveness required for ongoing improvement throughout the software development lifecycle. [9]

DevOps underlines its critical role in modern software development, focusing attention on collaboration, automation, and continuous improvement. It impacts the entire software and IT industry by building on lean and agile practices. The approach allows for more consistent and less painful software releases, with tests run early and often, reducing the overall end-to-end cycle time. [5]

### 2.1.1 Overview and Key Concept

This section provides a comprehensive overview of DevOps, emphasizing its significance in integrating and automating software development processes with IT operations. Key topics covered include the separation of development and operations and the challenges it presented and the key role of automation in managing infrastructure operations. It discusses the shift from manual to automated processes, such as deployment automation, infrastructure automation, and log management, which enhance operational efficiency. A key concept that is important to underline is Infrastructure as Code(IaC), the practice of managing infrastructure using code. It is vital in DevOps for enabling automation, ensuring consistency, enhancing scalability, reducing risks, managing costs, and fostering collaboration. It aligns with the DevOps goals of rapid service delivery and high operational efficiency, making it an essential practice for any organization aiming to adopt DevOps principles effectively. [10]

However, we can summarize the other key concepts in the DevOps lifecycle. It is showed in the Table 2.3 with 7 main parts: continuous development, integration, testing, deployment, feedback, monitoring, and operations.

Table 2.1: Lifecycle Overview

<b>Practice</b>	<b>Description</b>
Continuous Development	Facilitates an iterative process for the ongoing development and enhancement of software features, ensuring a consistent introduction of new functionalities and improvements.
Continuous Integration	Emphasizes the frequent consolidation of code contributions from various developers into a common repository, aiming to proactively identify and resolve integration challenges early in the development cycle.
Continuous Testing	Entails the persistent execution of automated tests throughout the development process to ensure that new code additions do not adversely affect the existing system or introduce defects.
Continuous Deployment	Involves the systematic and automated deployment of software updates to the production environment, striving for a process that is both seamless and efficient.
Continuous Feedback	Engages in the ongoing collection and analysis of feedback from all development stages, enhancing team communication and collaboration towards continual improvement.
Continuous Monitoring	Concentrates on the immediate and continuous surveillance of applications and systems in the operational environment to swiftly identify and address potential issues, thereby optimizing performance and maintaining system health.
Continuous Operations	Covers the perpetual management and upkeep of applications once deployed, ensuring the software remains stable, accessible, and reliable within the production setting.

This simplified overview encapsulates the essence of the DevOps, underscoring it as a dynamic and responsive approach that aligns software development with operational requirements to ensure efficient and innovative delivery to the market.

[9]

## 2.1.2 Architecture

DevOps Architecture is not a one-size-fits-all solution. Rather, it is a set of practices and tools designed to promote agility, flexibility, and innovation. This architecture leverages automation, CI, CD, microservices, and cloud computing to create a resilient, scalable, and easily manageable infrastructure. The core objective of DevOps Architecture is to streamline the development process, from coding and testing to deployment and monitoring, thereby enabling faster release cycles, reducing the time to market, and improving the overall quality of software products. This is the Table 2.2 we have the structure layer by layer:

Table 2.2: DevOps Architecture [1]

Layer	Technology	Description
Infrastructure	Virtualization technologies(e.g., VMs, containers)	generation, setup, and supervision of the foundational systems on which applications are executed.
Configuration Management	Tools like Ansible and Puppet	automated setup and governance of system components, ensures infrastructure remains in the desired state.
Continuous Integration	Jenkins and Travis CI	merging, testing and integrating code changes into a single shared repository to maintain code quality and readiness.
Continuous Delivery/Deployment	Spinnaker and Azure DevOp	automated progression of software modifications into live environments, enhancing the deployment frequency and reliability.
Monitoring and Logging	Prometheus and Grafana	systematic observation and analysis of application and infrastructure performance, providing insights for proactive management and optimization.

## 2.1.3 Methodologies

This section provides an in-depth examination of recent studies on DevOps methodologies. It presents an overview and analysis of the most important findings from the literature, also it looks in DevOps practices and how they impact software development, deployment, operations, and testing.

### 2.1.3.1 Tools Landscape

Utilizing tools is essential for automating DevOps. Deliveries of high quality with fast cycle times require a high level of automation. Therefore, as you transition to DevOps, it's critical to select the appropriate tools for your environment or project. Here's a detailed look at some of the essential tools used frequently in the DevOps ecosystem:

1. **Git:** A foundational tool for version control that allows developers to track and manage changes to code, facilitating collaboration by supporting features such as branching and merging.
2. **Lucidchart:** This tool aids in creating visual diagrams like flowcharts, making it easier to understand and communicate complex processes and system architectures among team members.
3. **Kubernetes:** An orchestration tool for managing containerized applications, it automates the deployment, scaling, and operation of applications across clusters of hosts.
4. **Ansible:** A powerful automation tool that manages configuration and deployment of applications, ensuring systems are set up and running as intended in a consistent manner.
5. **Jenkins:** A continuous integration tool that automates aspects of software development related to building, testing, and deploying, facilitating continuous delivery.

Each of these tools addresses different aspects of the DevOps workflow: Git and Jenkins streamline code versioning and integration, ensuring that code changes are automatically built, tested, and ready for deployment. Lucidchart enhances team communication and planning through visual documentation of systems and workflows. Kubernetes focus on container management, making applications easier to deploy, scale, and manage across different environments. Ansible automates the provisioning, configuration, and management of infrastructure, reducing manual effort and increasing consistency. These tools collectively support the DevOps goals of improving automation, collaboration, and efficiency across the software development life cycle. [9]

### 2.1.3.2 Metrics

After the tools overview we need to see how to measure the work of them, due to this we need some parameters from where we can get data and obtain results. DevOps metrics are used to track and evaluate the efficiency of DevOps practices inside a company. They are numerical evaluations that provide information on how successfully an IT department is improving the flow of work from development through operations and customer support in order to create value to the company. An example of the most common DevOps metrics [1] is:

- Lead time: The duration from code commit to code release.
- Deployment frequency: How often deployments occur, measured per time unit (e.g., daily, weekly).
- Mean time to recovery (MTTR): The average time required to recover from a production incident.
- Change failure rate (CFR): The proportion of deployments causing failures in the production environment.
- Availability: The percentage of time the system is operational and functional.
- Error rate: The frequency of errors occurring over a specified time period (e.g., hourly).
- Customer satisfaction: Measures how satisfied customers or end-users are with the product or service.
- Team productivity: The amount of work completed by the team over a specified period.

Tracking DevOps metrics gives several key benefits that means more efficiency, software quality and reliability, speeding up incident recovery, increasing system availability, and boosting customer satisfaction. These metrics allow us to avoid bottleneck, inefficiencies and also help us to identify area for fast team improvements.

### 2.1.4 Benefit, Challenge and Limitation

In this section, we going to see the benefits of implementing DevOps methodologies but also all the challenges and the issues we could have with this approach.

The main benefits of DevOps include faster software delivery, improved collaboration among teams, increased efficiency, higher quality software, and improved customer satisfaction. DevOps practices enable organizations to provide software more quickly, work together more effectively, streamline operations, and customer satisfaction by meeting their needs promptly. These advantages contribute to increased productivity, cost reduction, and a competitive edge in the market. On the other hand, the limitations of DevOps include resistance to changes, challenges in integrating legacy systems with modern DevOps tools, a lack of necessary skills and expertise among team members, and difficulties in implementing full automation due to either resource constraints or technical limitations. Furthermore, it can be very difficult to maintain security and compliance in increasingly automated and linked environments. These restrictions may make it more difficult to apply DevOps techniques successfully, necessitating the allocation of resources and strategic planning to get around them. In the Table we can see the summary of benefits and challenge on DevOps methodologies.

Table 2.3: Benefit and Challenges

<b>Benefits</b>	<b>Challenges</b>
Faster time-to-market	Lack of skills and expertise
Increased team collaboration	Resistance to change
Improved reliability of software	Legacy systems
More frequent releases	Lack of communication between teams
Greater flexibility	Integration and automation
Continuous feedback improvement	New ways of working

## 2.2 CI/CD

CI/CD is described as a DevOps methodology for modern software development and delivery. It emphasizes the importance of this process in achieving shorter compilation or build times and faster release cycles through more automation and testing. The methodology is gaining interest for its numerous benefits and efficiencies, including faster integration of code, which allows companies to deliver high-quality software releases more rapidly, obtaining a competitive advantage.[11]

### 2.2.1 Overview and Impact

This approach is particularly beneficial DevOps environments, where it supports continuous improvement and responsiveness to market changes. Fig. 2.1 provides an explanation of the process representation.

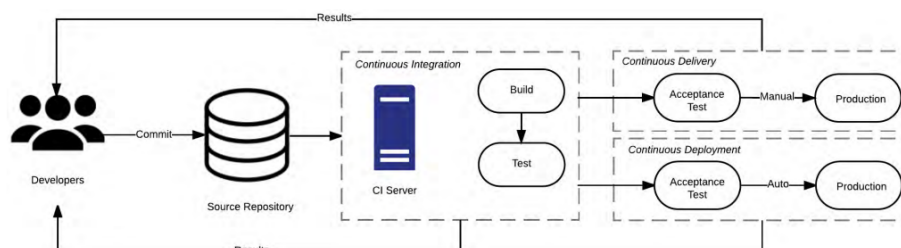


Fig. 2.1: CI/CD: The relationship between continuous integration, delivery and deployment [3]

We must underline the importance of CI/CD in the modern software development. The adoption of CI/CD practices has transformed the global IT industry by establishing new standards for software development and deployment. It has enabled organizations to accelerate digital transformation initiatives, adopt DevOps practices more widely, and improve collaboration between development and operations teams. As the demand for faster and more reliable software delivery continues to grow, the importance of CI/CD in the IT industry is expected to increase, further influencing development methodologies, corporate cultures, and market competitiveness. The way that CI/CD has changed the IT industry highlights how important it is to DevOps methods and how it will influence software development and deployment in the digital age.

## 2.2.2 Methodologies

As we have already seen, Continuous Integration (CI) and Continuous Deployment (CD) are key practices in modern software development that enable teams to increase efficiency, improve quality, and deliver applications more quickly and reliably. Here's an overview of methodologies commonly used for CI/CD:

- **Automated Testing:** Automated tests run during the CI phase to ensure that the new code does not break or degrade any existing functionality. Types of tests include unit tests, integration tests, and end-to-end tests..
- **IaC:** Manages and provisions infrastructure through code rather than manual processes, enabling consistent and repeatable environment setups.
- **Configuration management:** Tools like Ansible, Chef, and Puppet automate the configuration of software and systems, ensuring that environments are consistent and in the desired state.
- **Containers and orchestration:** Containers package software with all its dependencies, making it easy to deploy across different environments. Orchestration tools like Kubernetes manage these containers in production.
- **Monitoring and logging:** Continuous monitoring and logging of the application and infrastructure to quickly identify and resolve issues.
- **Feature flags:** Allow teams to toggle features on and off without redeploying, enabling safer testing in production environments and canary releases.

### 2.2.2.1 Tools

This subsection provides insights into main CI/CD tools. This qualitative analysis [12] lists a variety of modern tools used across different projects and organizations, highlighting the diversity and preferences in the CI/CD landscape. Here are the CI/CD tools mentioned: Hudson, Jenkins, Travis CI, GitLab CI/CD, CircleCI, Azure DevOps, Bitbucket Pipelines, AppVeyor, TeamCity, Cruise Control, Drone, Netlify, Bamboo.

Jenkins and GitLab CI as main and most common tools in the CI/CD landscape. Here's a brief description of each:

Jenkins is a Java-based open-source CI/CD tool renowned for its ease of installation and user-friendly GUI. With its wide range of plugins, it can be easily configured and is suitable even for novice users. Jenkins helps with automated deployment by building and pushing updates to the server in response to changes that

are committed to the GitLab repository. This is done via a webhook URL. Despite its user-friendly simplicity and adaptability, maintaining Jenkins becomes tough as the application and number of plugins grow, making it difficult to manage. It is suitable for large-scale enterprise projects due to its robust plugin support and flexibility in configuring automated deployment processes. However, its scalability can be a concern with the increasing complexity of projects. [12]

On the other hand, GitLab CI is an open-source version control system based on Git, offering CI/CD pipelines without the need for additional plugin support. All configurations are managed within a single YML file, simplifying the process of understanding and modifying pipelines regardless of project size. GitLab CI triggers a pipeline upon new changes committed to the repository, executing predefined jobs or stages for deployment. Its streamlined configuration process and documentation of pipeline changes as part of CI make it a maintainable option for continuous integration and deployment. It is recommended for simpler, microservice-based projects due to its efficient management of pipelines and ease of modification, making it a good option for sophisticated yet small projects. [13]

In summary, both Jenkins and GitLab CI are valuable tools in the CI/CD environment, each with its strengths and ideal use cases. Jenkins excels in large-scale enterprise projects with its extensive plugin support, whereas GitLab CI is preferred for its simplicity and scalability in smaller, microservice-based projects.

Each of these tools and methodologies focuses on automating steps in the software delivery process, improving developer productivity, and ensuring the delivery of high-quality software in a more efficient manner. The choice of tools and methodologies depends on the specific needs of the project, including the technology stack, team preferences, and existing workflows.

### 2.2.3 Benefit, Challenge and Limitation

CI/CD plays a main role in modern software development. This methodology brings a multitude of benefits that directly contribute to a more efficient, reliable, and agile software development lifecycle. The main benefits are:

Table 2.4: Benefits Overview

Benefits	Description
Faster release cycles	It enables more frequent code integrations, allowing teams to discover and fix bugs quicker, and speed up the time it takes to release new updates.
Improved quality and efficiency	By automating the build, test, and deployment processes, It reduces manual errors, enhances quality assurance, and increases operational efficiency.
Enhanced developer productivity	CI/CD pipelines automate repetitive tasks, freeing developers to focus on more strategic work.
Increased agility	It allows organizations to respond more quickly to market changes and customer feedback.
Cost and risk reduction	Frequent, incremental changes reduce the risk associated with large-scale software updates or releases.

The adoption of CI/CD, despite its numerous benefits, also presents several challenges and limitations that organizations must navigate:

- Complexity in existing systems and skill gaps: Integrating its pipelines into existing system and the implementation of CI/CD requires specific technical skills and expertise that may not be present within current teams.
- Quality assurance and testing challenges: Ensuring the quality of software with continuous integration and delivery necessitates a robust automated testing framework.
- Security concerns: Integrating security practices into the pipeline is critical but can be challenging.
- Toolchain complexity: Selecting and integrating the right set of tools for it can be overwhelming due to the vast array of available options.

- **Maintenance and infrastructure costs:** While CI/CD can reduce costs in the long term, setting up and maintaining the necessary infrastructure, especially in a scalable and resilient manner, can require significant upfront investment.
- **Risk of over-automation:** There's a risk of over-relying on automation in CI/CD processes, where human oversight becomes minimal.

Addressing these challenges requires a good strategy that includes investing in training, choosing the right tools, gradually integrating CI/CD practices into existing workflows, and fostering a culture of continuous improvement and collaboration across teams. Despite these, the benefits of adopting CI/CD often outweigh the limitations, making it a key driver of efficiency and innovation in software development. [11]

## 2.3 Logging

In transitioning from broader DevOps and CI/CD discussions to the focused area of logging and monitoring in this thesis, it's essential to articulate the critical role these practices play in enhancing and supporting DevOps methodologies. Logging is pivotal for the success of DevOps and CI/CD initiatives by ensuring continuous improvement and operational efficiency. The main insights it provides are quality assurance and performance optimization. By integrating logging into DevOps practices, organizations can achieve more reliable, secure, and efficient delivery pipelines, making this focus on a fundamental aspect of modern software development.

### 2.3.1 Overview

Logging in the context of software development refers to the process of recording information about the behavior and execution of a software system. This information, typically captured in log messages or files, provides a chronological record of events, errors, operations, and transactions that occur within the system. Logging serves as a fundamental component for various critical aspects of software maintenance and development. It aids in debugging by providing developers with a comprehensive insight into the application's behavior over time, proving to be indispensable for pointing the root causes of issues or bugs. Furthermore, it facilitates real-time monitoring of the system's operational health and performance, enabling the early detection and rectification of problems before they impact users. In term of security, logging assumes a pivotal role by documenting access and transactions, which can then be checked for any suspicious activities or potential security breaches. Lastly, logging is instrumental in performance analysis, allowing for the examination of system performance over time to identify any bottlenecks or to gain insights into user behavior, thereby guiding optimization efforts and influencing future development directions. [4]

As we have just seen, logs play a crucial role in software development and during the operational phase in numerous software systems. The process of software logging is divided into two main stages:

- Log instrumentation
- Log management

As shown in Fig. 2.2, log instrumentation involves the developers embedding logging code within the source code to capture information during runtime. On the other hand, log management is the process where operators gather these log messages and apply data analysis methods to extract insights into the system's runtime

behavior. While a range of open source and proprietary log management solutions exist, their success is significantly influenced by the quality of the logging code integrated into the system. Log messages that are the result of well-crafted logging code can substantially simplify various log analysis activities, such as system monitoring and fault diagnosis. [14]

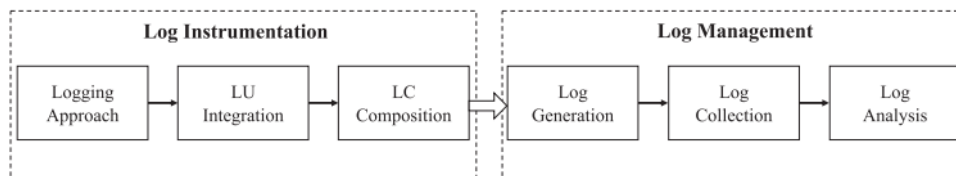


Fig. 2.2: The overall process of software logging [4]

We will specifically focus on the log collection and analysis aspect of software logging. This phase involves the aggregation of log messages generated by the software and the subsequent application of analytical techniques to these collected logs. The aim is to derive actionable insights into the software's runtime behavior, facilitating tasks such as system monitoring, error diagnosis, and compliance verification.

## 2.3.2 Methodologies

There are good practices and automated approaches to enhance logging in software development, particularly within the DevOps context. We are going to see in the following sections the best practises in implementing automation for log processing and a general overview of serviced used in a software development environment.

### 2.3.2.1 Best practises in implementing automation for log processing

Here, we delineate the best practices in logging processing with the implementation of automation.

A foundational best practice is the development and maintenance of high-quality logging code. The absence of standardized guidelines for logging has historically posed challenges for developers, necessitating the creation of benchmarks and tools that support high-quality logging practices. Developers are encouraged to utilize a consistent logging format across the entire codebase to simplify log analysis and include sufficient context in log messages to aid in diagnosing issues without needing to refer back to the source code.

Automated techniques to estimate code coverage through execution logs represent a significant advancement in logging practices. These methods allow for the assessment of which parts of the code are executed in real-world scenarios, facilitating a more targeted approach to testing and debugging. Automating this process helps in identifying untested paths and potential flaws in the application, ensuring a more robust software solution.

Another best practice involves enriching log analysis by automatically correlating logs with other telemetry data, such as traces and Application Performance Monitoring (APM) data. This approach enhances the contextual understanding of logs, enabling more effective troubleshooting and performance optimization. Automation in this area reduces manual effort and speeds up the diagnosis process, leading to quicker resolutions of issues. [15]

The utilization of software repositories including code, communication, runtime, and bug repositories is crucial for enhancing logging practices. These repositories offer a wealth of information that can be systematically analyzed to improve logging code and practices. By mining these repositories, developers can identify common logging issues, evaluate different logging approaches, and implement automated detection of logging code issues.

In a DevOps environment logging practices must evolve continuously. Automated feedback mechanisms should be established to inform developers of logging issues in real-time. This continuous feedback loop ensures that logging practices are consistently aligned with the evolving needs of the software development lifecycle.

In summary, there are systematic approaches to improve logging practices by developing guidelines for high-quality logging code, expanding log analysis techniques, and implementing automated methods for estimating code coverage for better problem diagnosis. These strategies aim to support both software development and IT operations sides, enhancing the overall quality and effectiveness of logging in DevOps environments. [16]

### **2.3.2.2 Overview of serviced used in a software development environment**

Setting up an effective logging environment is critical for the performance, reliability, and security of software applications. These systems provide insights into the application's operational status, help in debugging issues, and ensure that performance metrics meet the expected standards. Below is an overview of popular services used in software development environments for logging, along with their unique features:

- **Loggly:** A cloud-based log management and analytics service that helps in visualizing and analyzing logs in real time. It supports a wide range

of programming languages and platforms. It can dynamically scale with your application's needs, making it suitable for both small and large-scale applications. [17]

- Splunk: Offers powerful searching, visualization, and alerting capabilities. It can ingest data from different sources, making it versatile for analyzing machine-generated data. It is widely used for its comprehensive dashboarding capabilities. [18]
- ELK Stack (Elasticsearch, Logstash, Kibana): An open-source set of tools that work together to provide insights into your logs and operational data. Elasticsearch is a search and analytics engine, Logstash is for server-side data processing pipeline, and Kibana is for visualizing the data. This stack is highly customizable and scalable. [19]
- Datadog: A monitoring service for cloud-scale applications, providing monitoring of servers, databases, tools, and services, through a SaaS-based data analytics platform. It is known for its integration capabilities with various cloud services. [20]
- Graylog: An open-source log management platform that offers centralized log management, efficient searching, and alerting capabilities. It's designed to handle large streams of log data and provides a user-friendly interface for managing and analyzing logs. [21]

Each of these services has its strengths and is suited for different kinds of applications and organizational needs. Selection often depends on specific requirements such as scalability, cost, ease of use, integration capabilities, and the specific stack used in the development environment.

### 2.3.3 Benefit, Challenge and Limitation

Developers have various strategies to balance the benefits and limitations of logging. These strategies involve making informed decisions about log levels, dynamically configuring log levels to suit different scenarios, and refining logging practices to mitigate its negative impacts while maximizing its benefits. There are several benefits from logging, which can be categorized into:

- Troubleshooting: Logs play a central role in diagnosing runtime failures and serve as an indispensable tool for debugging. They allow developers to trace execution paths and identify root causes of software failures.

- **Tracking Execution Status:** Logging is instrumental in monitoring the progress and performance of software, offering real-time alerts for issues and anomalies, thereby facilitating performance tuning and resource allocation.
- **Assisting in Comprehension:** Beyond troubleshooting, logs enhance system comprehension, acting as a form of documentation that helps developers understand code functionality and runtime behaviors.
- **Serving Bookkeeping Purposes:** Logs act as a record of important transactions or operations, aiding in various analyses such as security, performance, and capacity planning.

However, logging also incurs several challenges and limitation, which include the management of large log data, impact on system behaviors, direct negative effects on users, and increased development efforts. The storage and processing of extensive log data can be costly and may produce noise that obscures important information, making log management challenging. Also logging can introduce performance overheads and, in some cases, may even alter the functional behaviors of a system. Inappropriate logging can confuse users, lead to misleading information, and expose sensitive data, thus undermining user trust and system security. Moreover, The development and maintenance of logging code require significant development effort, which can detract from code readability and overall software quality. Last but not least, manual inefficiencies can be a significant limitation for logging. When developers or system administrators have to manually go through extensive logs, configure logging levels, or manage log files, it can lead to several inefficiencies like time consuming, introducing human errors in the process and lack of consistency. [22]

## 2.4 Summary

This chapter provides a thorough overview necessary for understanding the project's scope and the application's specific environment, organizing this information into three clear sections. The first section introduces DevOps and its methods, laying the groundwork for the discussion that follows. The second section explores the important concepts of Continuous Integration and Continuous Deployment (CI/CD). The third section looks at system logging and different methods used for effective monitoring. The main message of this literature review is to establish a strong base of knowledge that supports and help to find the issues. It shows how DevOps, CI/CD, and logging practices are key to software development, helping readers understand the problems better and pointing out what needs to be done in future chapters for successful project implementation. This literature review is crucial because it connects theory with practice, highlighting how modern automation can improve software development and tackle specific challenges in the project.

## **Chapter 3**

### **Case Study**

This section is focus on outlining and detailing Ericsson’s involvement in the context of this research, specifically regarding automation technologies in Continuous Integration (CI) processes. This section should serve as a bridge connecting the theoretical aspects of your study with the practical application at Ericsson.

#### **3.1 Overview of environment**

This section aims to highlight the significance of Ericsson’s Continuous Integration (CI) system and its operational dynamics, offering a comprehensive overview of the environment. By exploring into Ericsson’s CI practices, we intend to illustrate not only the role these processes play in maintaining software quality and efficiency but also the complex ecosystem in which they operate. Ericsson presents a highly pertinent case study, to address manual inefficiencies in CI processes, for a thesis focused on the application of automation technologies. This is for several reasons, like its leading role in telecommunication, the commitment to software development quality, extense use of CI processes and innovator in automation technologies.

##### **3.1.1 Importance of CI in Ericsson’s Operations**

By integrating CI processes into its software development lifecycle, Ericsson leverages several key benefits that align with its corporate objectives:

- **Software Quality:** CI involves the frequent integration of code changes into a shared repository, followed by automated builds and tests. This practice allows Ericsson to detect and fix errors early in the development process, significantly improving the quality of the software.

- **Development Efficiency and Product Reliability:** By automating the build and test processes, CI minimizes manual tasks involved in code integration, allowing developers to focus on more critical aspects of software development.
- **Faster Time-to-Market:** CI enables Ericsson to more rapidly move from development to deployment, as the automated processes reduce the time required for manual testing and integration.
- **Collaboration and Transparency:** CI encourages collaboration among development teams by integrating their work frequently into the main repository. This practice enhances transparency, as the impact of changes made by individual developers on the overall project is visible and can be assessed promptly.

In summary, Continuous Integration is integral to Ericsson's strategy for maintaining high standards of software development, ensuring product quality, and achieving market responsiveness.

### 3.1.2 CI Process at Ericsson

As shown in Table 3.1, the Continuous Integration (CI) process at Ericsson is an extensive system that integrates a variety of development stages and tools to manage the software lifecycle. Starting with planning, JIRA is utilized for project management, allowing teams to track progress and issues with a Java-based interface. During the build phase, tools such as Maven and Gradle automate the compilation, testing, and packaging of the code. Maven leverages XML configurations, while Gradle uses Groovy along with a domain-specific language (DSL), catering to Java projects. The CI process ensures that each integration is verified by an automated build (using tools like Jenkins) to detect integration errors as quickly as possible.

Jenkins, a key component in the build phase, facilitates continuous integration and testing by scheduling and monitoring execution of predefined jobs with a user interface configuration in Java. Gerrit complements this by providing code review and repository management for Git, facilitating collaborative code changes and quality assurance before integration. TestNG is another integral part of the CI pipeline, providing a testing framework that uses XML for configuration, ensuring that the Java code is rigorously tested.

In the transition from build to deployment, JFrog Artifactory (implied by the mention of Jfrog) serves as a repository manager, which efficiently handles binary resources, ensuring they are retrievable and storable in a controlled way. The CI pipeline concludes with deployment, where Ansible is employed for its simple,

yet powerful, configuration management capabilities, using YAML for configurations, typically orchestrated with Python.

Table 3.1: Automation tools for DevOps in Ericsson

Tool	DevOps phase	Tool type	Configuration	Language
JIRA	Plan	Project management	UI	Java
Maven	Build	Build	XML	Java
Gradle	Build	Build	Groovy	DSL
Jenkins	Build-Test	Continuous Integration	UI	Java
Gerrit	Build-Test	Continuous Integration	UI	Git
TestNG	Test	Continuous Integration	XML	Java
Jfrog	Build-Deployment	Conf. management	UI	Java
Ansible	Deployment	Conf. management	YAML	Python

The CI process is a critical aspect of Ericsson’s ability to deliver high-quality software continuously. By leveraging these tools and processes, Ericsson can manage the scale and complexity of their operations while facing the typical challenges that come with large-scale CI environments.

### 3.1.3 Type of logs

Jenkins generates several logs to track its activities. These logs can reside in system files or within the Jenkins itself, depending on your operating system and configuration. There are logs for the overall Jenkins application, as well as individual build jobs. The most important logs and the one we will take into considerations are the console logs, which capture the detailed output displayed during the execution of a particular build. This console output provides valuable insight into the success or failure of each build step.

## 3.2 Manual inefficiency in log analysis

The existing state of our CI process reveals a lack of automation on the analysis of the results, particularly in overseeing and managing the logs of daily and weekly loops. These inefficiencies are critical to address, as they not only use significant amounts of time and resources but also obstruct the efficiency of the software development lifecycle, so the work of the team. Specifically, inefficiencies include:

- **Time-Consuming Manual Review:** Manually reviewing logs to identify errors, anomalies, or patterns consumes a significant amount of time. This is

time that could otherwise be spent on more value-adding activities, such as new feature development or optimizing existing code.

- **Error Risk with Manual Analysis:** When logs are reviewed manually, there's a high risk of missing subtle warnings that could turn into larger issues. Reliance on manual methods not only increases the chance of missing early signs of problems but also lead to human errors.
- **High Volume of Logs:** The CI/CD process generates a vast amount of log data. The huge volume of data makes it challenging for team members to sift through manually, leading to inefficiencies in identifying and addressing issues.
- **Complexity of Log Data:** Logs from test cycles can be highly complex, containing intricate details that require specialized knowledge to interpret. This complexity is compounded when logs from various stages of the CI/CD pipeline need to be correlated to diagnose issues effectively. The manual effort required to understand and act upon this information is substantial and prone to errors.
- **Inconsistent Test Results (Flaky Tests):** Identifying and diagnosing flaky tests manually through log analysis is difficult and time-consuming. These tests can lead to a lack of confidence in the testing process and may require repeated investigations to pinpoint the underlying issues.
- **Recurring Errors Across Different Test Cases:** Identifying patterns or recurring errors across different test cases its a manual process. Developers might not easily notice when the same error impacts multiple test cases, leading to duplicated effort in troubleshooting and fixing these issues.
- **Inefficiencies in Prioritizing Issues:** Manually reviewing logs makes it challenging to prioritize issues based on their impact or severity efficiently. Teams may spend time addressing minor issues while more critical problems that could have a significant impact on the product quality or release timelines are overlooked.
- **Lack of Automated Analysis for Log Data:** Automated tools can apply complex algorithms to quickly identify patterns, anomalies, and recurring issues within the log data.

To address the thesis's research question on minimizing human work in processing daily and weekly logs from CI, focusing on automating the analysis and management of log data could provide significant benefits. We will see the solution of the thesis in the chapter 4.

## 3.3 Problem statement

The approach of this thesis is a mixed research approach. Qualitative approach contrasts with quantitative research, which seeks to quantify problems and understand how prevalent they are by looking for statistical relationships. Instead of relying on numerical data, qualitative research, or the quality approach, is all about getting a closer, personal look at a topic. Instead of counting things or using statistics like in quantitative research, this method involves talking to people, and understanding their opinions. It's like choosing to have a deep, one-on-one conversation with someone to really understand what they think of the problem.

### 3.3.1 Methodology

The mixed approach combines both qualitative and quantitative methods to get a fuller understanding of a topic. In the context of estimating the hours spent on CI log analysis, this approach would mean using interviews to gather in-depth insights (the qualitative part) and surveys to collect more straightforward, numerical data (the quantitative part).

- **Interviews** offer a qualitative perspective. By talking directly to people involved in CI log analysis, you can understand their experiences, challenges, and perceptions in detail. This method allows you to dive deep into the "why" and "how" behind the time they spend on CI analysis, capturing the nuances and complexities of their work that numbers alone can't show.
- **Surveys**, on the other hand, provide quantitative data to estimate the hours the team spend on CI log analysis. This part aims to gather measurable, numeric information that can be analyzed statistically to understand how much time is typically spent on these tasks.

By targeting my team of three software developers, three master developers and a System Architect for both interviews and surveys, I was able to customize the questions to fit our work situation and roles, making the results more relevant and useful. Together, these methods provide a motivation for the problem of this thesis.

#### 3.3.1.1 Interviews and Survey

To explore the challenges for enhancing automation in the log analysis process of our CI/CD pipeline, I conducted interviews with key team members actively engaged in these processes. The aim was to gather in-depth insights into the practical implications of the current state of log analysis, its impact on team efficiency,

and to identify potential areas for introducing automation to reduce manual effort. The questions were designed to understand how these challenges affect their day-to-day operations and workflow efficiency. Interview questions:

1. **Problem Agreement:** Do you recognize the following issues in our CI process?
  - Time-Consuming Manual Review
  - Lack of Automated Analysis for Log Data
  - High Volume of Logs
  - Complexity of Log Data
  - Inconsistent Test Results (Flaky Tests)
  - Recurring Errors Across Different Test Cases
  - Inefficiencies in Prioritizing Issues (e.g. a bigger mistake than a slight one)
2. **Daily Workflow Impact:** How does the current state of log analysis affect your daily workflow? (e.g. Is it the first thing you do during the morning)
3. **Manual Review Process:** Can you describe the steps you take during a manual review of CI logs?
4. **Tracking and Addressing Recurring Errors:** (only if you agree on this problem) Do you track and address errors that recur across different test cases? How?
5. **Prioritization Process:** (only if you agree on this problem) Do you currently prioritize issues found during CI log analysis? How?
6. **Potential for Automation:** Where do you see potential for automating parts of our log analysis that are currently manual?
7. **Process Improvements:** What process improvements would you suggest for our CI log analysis?

Also, for this thesis, a survey was conducted to quantitatively measure the amount of time spent by the team on log analysis within our CI/CD processes over a one-month period.

### 3.3.2 Data collection analysis

The manual inefficiencies in managing logs within Ericsson's CI process, particularly around the testing stage, have a broad and significant impact on overall operations. Here's a concise analysis of these effects.

### 3.3.2.1 Survey analysis

A survey was performed for this thesis in order to quantify how much time the team spent over the course of a month on log analysis inside our CI/CD procedures. As we can see in Fig.3.1, it is evident that, on average, team members together spend approximately 180 minutes, so 3 hours per day to the analysis of logs. Typically, two developers work concurrently on CI. It is 7.5h a day per developer, so we can estimate the percentage of inefficiencies of people working on that and that is around 20%, equating to approximately 3 hours a day. This significant allocation of time underscores the considerable manual effort and highlights the pressing need for automation solutions. This finding quantifies the operational inefficiency, providing a case for the exploration and implementation of automated solutions to alleviate the manual work on the team and improve overall process efficiency.

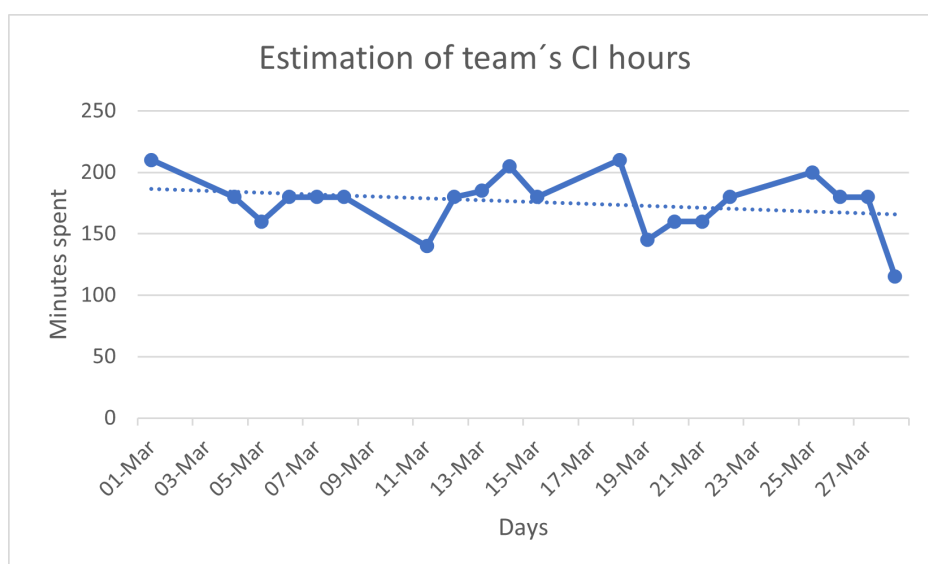


Fig. 3.1: Estimation of time spent on CI analysis

### 3.3.2.2 Interviews analysis

This part is all about the interviews analysis and what I got from my colleagues. By listening to their experiences, I wanted to find the important ideas and understand the different ways people think about my research topic. This will help us answer our main research question, how to minimizing human work in processing logs, even better and give a more complete picture.

#### 1. Problem Agreement

As we can see from Fig.3.2 the team confirms that these are recognized issues, especially considering:

- **Time-Consuming Manual Review:** Manual review is often extremely boring and error-prone. A high bar on the graph for this issue indicates it's a significant drain on your team's resources.
- **Lack of Automated Analysis for Log Data:** Limited automation for log data suggests this is an area for improvement, reflected in the graph.
- **Recurring Errors Across Different Test Cases:** These errors waste time and can mask more critical issues. If the graph shows a high value for recurring errors, it highlights a need for better root cause analysis.

The fact that my team identified these issues, particularly the manual review and lack of automation, suggests they understand the need for improvement.

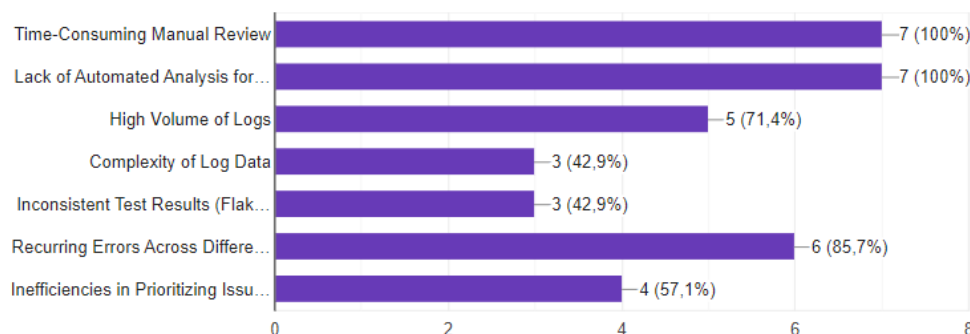


Fig. 3.2: Problems on our CI

## 2. Daily Workflow Impact

Team members generally aim for a quick log review before their daily meeting (around 9:30 AM) to identify any major issues. However, the amount of time spent on log analysis varies greatly. Most prioritize checking the bug tracker and number of failed builds first, diving deeper only for significant failures. Log analysis can take anywhere from a quick check to a lengthy 2-3 hour investigation depending on the complexity and number of issues encountered. This unpredictability can disrupt daily workflows, especially when complex issues arise.

## 3. Manual Review Process

When manually reviewing CI logs, people typically follow a process that involves several steps. First, they check the information logged in the JCAT log for the failed test case. This might reveal the issue directly, such as a failed assertion or exception. If more investigation is needed, they move on to more detailed logs depending on the type of failure. This could involve specific internal logs or `ssh_logs` for connection issues.

Once they have the relevant logs, they narrow down the root cause. This involves focusing on the first failed test case and then the specific steps within that case where the failure occurred. They then browse through the relevant logs based on timestamps to identify potential reasons for the failure.

Finally, depending on the identified issue, they might take further actions like checking the track history, analyzing individual loops for problems, or performing a detailed analysis of the identified problem. This highlights a multi-step process that requires identifying the initial failure point, selecting the appropriate logs, and then following a process to pinpoint the root cause and take corrective actions. Everyone emphasizes that manually sifting through these logs is quite boring and has a huge room for automation.

## 4. Tracking and Addressing Recurring Errors

Recurring errors are tracked, but with some limitations. JIRA is the primary tool used for tracking, especially for errors involving:

- Hardware or software issues requiring discussion with other teams.
- Issues fixed with a code commit.
- Product-related issues.

However, identifying recurring errors involves manually checking failed cases from different test loops/executions with similar symptoms. This is

seen as a time-consuming process. Additionally, some commonly occurring failures, like temporary test environment issues, aren't consistently tracked in JIRA.

Overall, JIRA is used, but the process seems manual and may not capture all recurring errors.

### 5. Prioritization Process

Issue prioritization from CI log analysis reveal a lack of a single, standardized method. Here's what emerged: Priority often aligns with the severity of the product impact, some prioritize issues they have expertise in for faster resolution. This can potentially delay addressing areas outside their skillset and a few prioritize based on the overall project impact. Issues affecting core features or the master branch receive higher attention.

Overall, while some level of prioritization exists, the current system seems subjective and might lack a clear, documented approach.

### 6. Potential for Automation

The responses suggest several areas for automation in the current manual log analysis process:

- **Grouping Similar Failures:** Automatically identify and group failures across different tests, loops, or executions based on similar symptoms (error messages, affected blocks). This would streamline analysis by grouping related issues.
- **Automatic Claiming:** Automate claiming recurring issues or failures that affect multiple loops in the same execution, reducing manual effort spent on repetitive tasks.
- **Visualizing Past Issues:** Implement a system to visualize previously identified issues within a specific loop. This would help analysts quickly determine if a current failure is a repeat issue.
- **Automating Known Faults:** Develop automated log analysis tools to detect known faults. This would eliminate the need for manual tasks like downloading logs, searching for relevant files and timestamps.
- **Pattern Recognition:** Implement automation to identify patterns in failures based on timestamps and error messages. This could potentially pinpoint the faulty component or root cause, saving analysis time.

The overall focus is on automating repetitive tasks like identifying similar failures, claiming recurring issues, and analyzing logs for known faults. This would free up valuable time for manual investigation of complex issues. Some responses even hinted at the potential for future advancements with AI-powered log analysis.

## 7. Process Improvements

The feedback highlights areas for improvement in CI log analysis, emphasizing reduced manual work and increased efficiency. Repetitive tasks like collecting similar failures and claiming recurring issues could be automated to free up time for analysis. A system to track past issues and facilitate collaboration would prevent duplicate work and leverage team expertise. Also, automatic ticket creation for high-priority issues could expedite addressing critical problems. Overall, the goal is to streamline the process by minimizing manual effort, fostering collaboration, and potentially automating aspects of issue tracking and prioritization. This would allow the team to shift focus from repetitive tasks to investigating and fixing the root causes of failures.

### 3.3.2.3 Goal

Following the implementation of automation in our log analysis process, as informed by our comprehensive interviews, our primary objectives are two:

- Significant time savings
- Increased team motivation

## 3.4 Similar research on the topic

Log analysis automation has emerged as a critical area of research and development due to its significant potential to streamline system monitoring, enhance security measures, and improve operational efficiency across various domains. This section reviews some studies and methodologies in the field, offering valuable insights into different approaches to automated log analysis.

Automated log parsing and anomaly detection play a crucial role in identifying and mitigating potential issues in real-time. A study introduces Drain, an online log parsing approach that utilizes a fixed-depth tree structure to systematically parse log files. The system is designed to detect anomalies by identifying patterns and outliers that deviate from normal behavior, which can indicate potential system failures or security breaches. Drain leverages machine learning techniques to

enhance its accuracy and efficiency in real-time scenarios, making it a valuable tool for continuous monitoring and rapid response in complex systems. [23]

Big data systems (BDSs) are intricate, consisting of various hardware and software components, such as distributed computing nodes, databases, and middleware. Analyzing logs generated by BDSs can expedite this process, improve testing, detect security breaches, customize operational profiles, and support other runtime-data analysis tasks. However, several practical challenges hinder the adoption of log analysis tools. This study shows that some practical solutions exist. [24]

Also, another research proposes a hybrid approach that combines natural language processing (NLP) feature extraction methods and deep learning for anomaly detection. The extracted features are then fed into a Long Short-Term Memory (LSTM) network for anomaly detection. The performance of LSTM is compared with other machine learning algorithms and the results. The paper concludes that LSTM is a promising tool for log analysis due to its superior performance in handling log data and detecting anomalies. [25]

In conclusion, these studies demonstrate the efficacy of machine learning techniques in real-time log parsing and anomaly detection, showcasing the potential for rapid response and continuous monitoring in complex systems. Research on Big Data Systems highlights the unique challenges and solutions related to log analysis in large-scale environments, emphasizing the need for scalable and accurate tools to manage and analyze extensive log data. Furthermore, hybrid approaches combining NLP feature extraction and deep learning, particularly LSTM networks, have proven effective in detecting anomalies, underscoring the superiority of advanced machine learning models in handling and analyzing log data. These diverse approaches collectively contribute to the ongoing improvement and innovation in automated log analysis, paving the way for more robust and efficient systems.

### **3.5 Summary**

The data clearly shows that team members currently use an average of three hours daily to manual log analysis that is not only time-consuming but also monotonous. By automating this process, we aim to save time, thereby enabling the team to focus on more engaging, and value-added activities. The insights gained from the interviews underscore the necessity of this transition, highlighting the anticipation of a work culture where time is optimized, and individuals feel more fulfilled and productive in their roles.

## Chapter 4

# Automation into existing CI/CD

As organizations aim to simplify their development processes, the integration of automation technologies into existing CI/CD pipelines has become the main focus. Next section is about integrating automation into established internal company CI/CD frameworks with a machine claiming. It introduces the concept of machine claiming and its role in optimizing CI/CD workflows for enhanced efficiency, reliability, and scalability. Through a examination of machine claiming usages. From understanding the fundamentals of machine claiming, this section serves as a definitive guide for navigating the complex terrain of automation in CI/CD.

### 4.1 Machine Claiming overview

The primary focus of this thesis is to explore an automated failure detection and claiming machine designed to streamline the process of identifying and addressing failures in software projects. This script operates by first getting user-defined rules specified in a JSON file, which describe the specific errors the system should detect. These rules are parsed from the JSON format into strings that the script can process. Following this setup, the machine executes a comprehensive loop that accounts for various dimensions such as project type, team involved, and the scope of work. During this phase, it processes the Jenkins console logs, searching for patterns that match the predefined error descriptions. By automating the detection of such failures, the script not only enhances the efficiency of the troubleshooting process but also ensures a systematic approach to error management across different projects and teams.

### 4.1.1 Usage

The script is designed to automate the process of failure detection in software projects by analyzing Jenkins console logs against predefined error patterns specified in a rules configuration file. The usage of this script outlines how to correctly input command line arguments to operate the script. Here is a breakdown of the command line parameters and their meanings:

- `-p <project_id>`: Specifies the project identifier.
- `-t <view_type>`: Indicates the view type, which can be flow or team.
- `-i <view_id>`: An optional custom value that can be used to specify a track ID or team ID.
- `-s <view_scope>`: Another optional parameter to define the scope of the view, like hourly, daily or weekly.
- `-l <loop_id>`: Also optional, used to specify a loop ID.
- `-c <rules>`: Path to the JSON file containing the rules configuration.
- `-n <count>`: Indicates how many past executions of a particular loop should be checked.

Several example commands are provided to demonstrate typical uses of the script:

- To process a specific project and view type without additional options:

```
python machine.py -p project_name -t flow
```

- To include a view ID or additional details like team and rules file:

```
python machine.py -p project_name -t team  
-i team_name -c ./rules.json
```

This script is a powerful tool for automated monitoring and analysis of software projects, capable of tailoring its operation to various project settings and requirements, thus facilitating a proactive approach to error management and operational efficiency.

### 4.1.2 Rules

Rule files, formatted in JSON, are crucial for the operation of automated failure detection systems. These files define the patterns and conditions under which specific errors or failures are identified within system logs. This section outlines how to create and structure these rule files effectively. Key Elements of the file:

- **Sub-type:** Categorizes the rule within a specific part of the system.
- **Message:** Describes the error scenario.
- **Patterns:** Contains the log patterns to be matched. Each pattern includes:
  1. **Mark:** The specific string or error to look for.
  2. **Operation:** The logical operator, like 'and' or 'or', defining how patterns are combined.
- **Reference:** A link to more detailed information or documentation.

The creation of effective rule files is an essential component in enhancing the robustness and responsiveness of automated failure detection systems. By systematically identifying common failures, defining precise patterns, and establishing logical operations for these patterns, organizations can develop highly effective monitoring tools that significantly improve system reliability. Additionally, organizing rules into clear categories and linking them to detailed documentation ensures that these tools are both manageable and maintainable. Finally, thorough testing of rule files in a controlled environment is crucial to confirm their accuracy and effectiveness before deployment. Implementing these steps will not only streamline the monitoring process but also minimize downtime and facilitate a proactive approach to system management, ultimately contributing to a more resilient infrastructure.

## 4.2 Automation in error detection and analysis

This section explains how the machine simplifies error detection and management. One of its key features is the ability to look back at past system activities to spot recurring issues. This helps in understanding and fixing errors that happen more than once. Another major feature is that the machine automatically creates JIRA tasks when it finds an error. They allow to save the effort of manually entering these issues or going through all the old logs to find that recurring error, making the process faster and more organized. By automating these tasks, the system helps teams save time and focus on more important work.

### 4.2.1 Tracking of the past issues

From our interviews, it's clear that identifying recurring errors has been a significant challenge, primarily due to the manual effort required to review failed cases across various test loops and executions. This process is not only time-consuming but also prone to oversight, particularly with intermittent issues like temporary test environment faults that are not consistently recorded in JIRA. The automation introduced by this machine addresses these challenges by systematically analyzing historical data to detect patterns and recurring errors. This capability allows the system to identify and log issues automatically, even those that are sporadically reported. By running this script, the machine can investigate into past executions much more efficiently than manual processes, facilitate collaboration would prevent duplicate work, ensuring that even less obvious, recurring problems are detected and documented.

### 4.2.2 Automatic JIRA ticket creation

Feedback from interviews highlighted a need for automatic ticket creation in JIRA for identified issues. This feature is essential for expediting the resolution of critical problems and significantly reducing manual efforts associated with logging and tracking these issues. By automating the JIRA tickets, the system ensures that every detected error can be promptly and systematically recorded, leaving no room for oversight or delay. This automation not only speeds up the process of addressing defects but also ensures a consistent documentation flow.

To automate the creation of JIRA tickets, the script uses the Atlassian Python module. This module allows direct interaction with the JIRA API, simplifying the process of managing tickets in response to problems detected in projects. Here is a description of how the code works: The script begins by importing essential modules: It imports the Jira class from the Atlassian Python library, enabling the

script to interact with JIRA and the `HTTPError` exception from the `requests` library to handle potential HTTP errors during API communication.

The main method is designed to create a JIRA ticket with the following parameters:

- `project_key` (`str`): The identifier key of the JIRA project where the ticket will be created.
- `summary` (`str`): A brief summary of the issue the ticket will address.
- `description` (`str`): A detailed description of the issue.
- `issue_type` (`str`, optional): Specifies the type of issue (e.g., Bug, Task, Story), with a default value of "Task".

Inside the function an instance of the `Jira` class is created, which requires the JIRA server URL and an authentication token (`MyPersonalAccessToken` in this case). then the function attempts to create a ticket by assembling issue data in the form of a dictionary, which includes the project key, summary, description, and issue type. The `issue_create` method from `atlassian` module of the `Jira` instance is called with the issue data to create the ticket.

In large and complex projects, precision is crucial to ensure efficiency and reduce overhead. Proper error handling becomes fundamental, especially in automated processes like creating JIRA tickets. The provided script incorporates robust error management to address potential issues during the API call. If complications arise, such as network disruptions or data formatting errors, the script is designed to catch an `HTTPError`. Upon encountering such errors, it prints the detailed response text to provide clarity on what went wrong and re-raises the exception. This mechanism alerts the calling process of the failure, enabling immediate corrective actions and preventing the creation of incorrect or empty tickets. This level of vigilance in error handling ensures that automated ticket creation does not introduce new problems into the project management workflow, thereby maintaining the integrity and continuity of the project's operations.

The method is then exemplified with a call, whenever we find an error and we want to report the issue, that tries to create a new JIRA ticket. If successful, it prints a success message along with the ticket details. If an error occurs, it catches the `HTTPError` and prints an error message. This script effectively automates the task of manual ticket entry in JIRA, ensuring that all detected issues can be promptly and consistently recorded, which aids in swift issue resolution.

### 4.3 Summary

In conclusion, the automation introduced in our system significantly enhances the management of software projects by addressing two key areas: the tracking of recurring past issues and the automation of JIRA ticket creation. By implementing a methodical approach to analyze historical data, our system effectively identifies patterns and recurring errors that might have been overlooked manually. This capability ensures that even intermittent or less obvious issues are caught and addressed. Furthermore, by removing the manual entry of error reports into JIRA, the system not only saves time but also reduces the potential for human error. This allows project teams to focus more on solving issues rather than documenting them, accelerating the resolution process and enhancing productivity.

Together, these automated processes ensure that every detected issue, no matter how small or infrequent, is promptly recorded and addressed, leading to a more robust and efficient project management cycle. This comprehensive approach to automation thus represents a significant step forward in how projects are monitored and managed, driving quicker resolutions, reducing downtime, and maintaining a high standard of software development.

## Chapter 5

# Result and Analysis

In modern software development, Continuous Integration and Continuous Deployment (CI/CD) pipelines are foundational elements that drive the quick evolution of applications. Within these systems, the generation and management of logs play a critical role. As such, the logs that are produced during various stages of CI/CD are indispensable for monitoring system operations, identifying errors, and ensuring software quality and efficiency.

However, the management of these logs, especially in a high-volume, fast-paced environment, presents significant challenges. The manual processing of logs through daily and weekly outputs has emerged as a particularly large and time-intensive task. This manual effort not only slows down the development process but also diverts valuable human resources away from more stimulating activities and the complexity of manual log analysis, often lead to inefficiencies.

Given the need to enhance the efficiency of log management in CI/CD pipelines, this thesis is based on the following research question:

- How to minimize human work in the processing of logs from CI?

Addressing this question is crucial not only to reduce the manual effort on the development team but also to improve the overall software quality and expedite the software release process.

This chapter presents the results of our investigations into automated log processing solutions. By exploring automation strategies and technologies, the thesis aims to provide answers to our research question, thereby offering solutions that can reduce human intervention in log analysis.

## 5.1 Manual vs Automated

The traditional approach to log analysis in CI/CD pipelines, as revealed through our comprehensive interviews, has been predominantly manual. This method involves significant human effort, where team members go through voluminous log files to detect anomalies, identify patterns, and address failures. The primary disadvantages of manual log analysis include the considerable time consumption and the repetitive nature of tasks such as tracking similar failures and addressing recurring issues. These tasks are not only labor-intensive but also prone to human error, which can lead to oversight of critical problems and duplication of effort.

In contrast, the automated log analysis system that we propose aims to address these inefficiencies by minimizing manual labor and enhancing process efficiency. Automation in this context serves several functions. Our system has a systematic approach to analyze historical data, effectively identifying patterns and recurring errors that could be missed through manual review. This method ensures that even intermittent or subtle issues are detected and resolved. Furthermore, the implementation of automatic ticket creation for issues is another facet of the proposed automation. This feature ensures that critical problems are escalated immediately and addressed promptly, thus reducing the turnaround time for problem resolution. By automating these aspects of the log analysis process, the proposed system not only frees up team members from manual effort but also allows them to focus on other tasks.

The implementation of automation in our log analysis process has successfully met our two primary objectives:

- time savings
- increased team motivation.

These achievements not only confirm the feasibility of reducing human labor in managing CI logs but also set the stage for discussing how our results address the research question about minimizing human effort in CI processes, which will be explored in the next section.

## 5.2 Impact of automation into existing CI/CD

For this thesis, we conducted a survey to quantify the time our team spent on log analysis within our CI/CD processes over a month. The data revealed that, on average, team members collectively dedicated about 180 minutes, or 3 hours per day, to log analysis. This substantial time investment underscores the significant manual effort involved and highlights the urgent need for automation solutions.

As a result of implementing our automation, we estimate that our team can now save 20 minutes per day. Fig.5.1 illustrates this reduction.

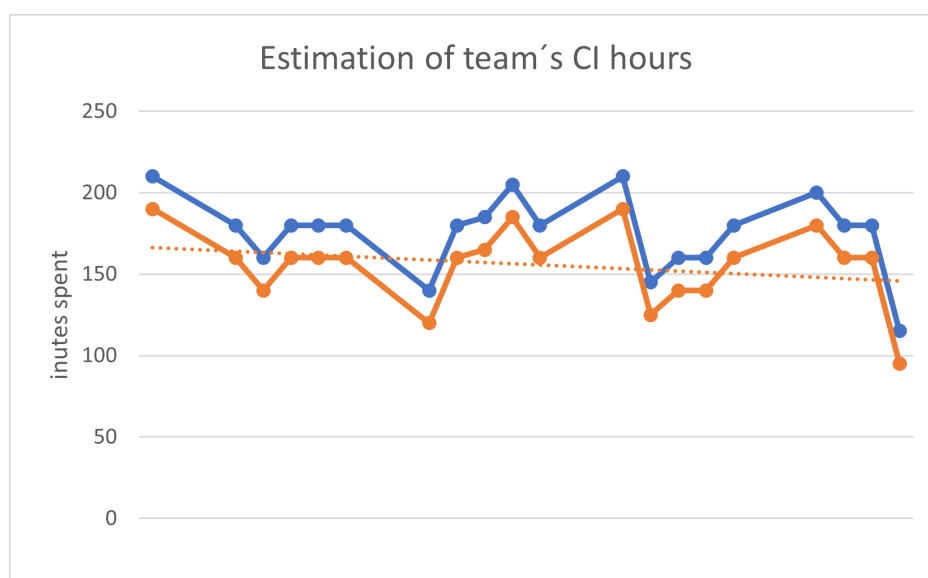


Fig. 5.1: Estimation of time spent on CI analysis after implementation

The figure presents a clear visual representation of the time savings achieved through the implementation of our automation processes in CI/CD log analysis. The orange line in the graph represents the new estimation of the total daily hours spent by the team on log analysis after automation. Previously, the team spent approximately 3 hours per day (as indicated by a higher blue line in the graph). With automation, this has been reduced, reflecting a daily saving of 20 minutes.

This reduction might seem modest at first glance, but it represents significant progress. The decrease in time spent each day accumulates to a substantial amount over weeks and months. This graph not only quantifies the impact of our automation efforts but also visually underscores the substantial efficiency gains, demonstrating that our automation initiative effectively streamlines operations and reduces manual labor in log analysis.

### 5.2.1 Data comparison

Two developers work concurrently on CI, each dedicating 7.5 hours per day. We previously estimated that inefficiencies in their tasks accounted for about 20% of their time, equating to approximately 3 hours per day combined. In Tab.5.1 we can see the adjustment in this percentage following the implementation of automation enhancements.

Table 5.1: Time saved in CI analysis after implementation of automation

	<b>Daily</b>	<b>Weekly</b>	<b>Monthly</b>	<b>%</b>
<b>Total Workload</b>	15h	75h	300h	100
<b>CI Analysis</b>	3h	15h	60h	20
<b>CI Analysis After Implementation</b>	2h 40min	13h 20min	53h 20min	~17.8
<b>Time Saved</b>	20min	1h 40min	6h 40min	2.2

The table illustrates the time allocation for CI log analysis before and after the implementation of an automation system, across daily, weekly, and monthly intervals, with an additional column for percentages.

- **Total Workload:** This row shows the total working hours at each time interval
- **CI Analysis:** This row indicates the time specifically spent on CI log analysis before automation
- **CI Analysis After Implementation:** This row shows the reduced time spent on CI analysis after the automation was implemented
- **Time Saved:** This row quantifies the time saved due to automation

The table effectively highlights the impact of automation on reducing the manual workload associated with CI log analysis. The reduction in time not only indicates an increase in efficiency but also suggests an improvement in the process's effectiveness, potentially allowing team members to reallocate their time towards more productive tasks. Overall, we managed to save 2.2%.

### 5.3 Summary

In this chapter, we have detailed the outcomes of our implementation of automation within the CI log analysis process. Our main achievement has been a reduction of 20 minutes per day in the time spent on log analysis, representing approximately 2.2% of the total workload.

This time saving, though seemingly modest, accumulates to a considerable reduction over longer periods, freeing up valuable hours that can be redirected towards more critical and complex tasks within the software development life-cycle. Moreover, this reduction in time spent on manual tasks has led to an increase in team productivity and a qualitative improvement in the working conditions and satisfaction of the development team. By diminishing the boring and repetitive aspects of log analysis, we have brought up a more engaging and productive work environment. Members can now focus more on areas that leverage their expertise and creativity, enhancing both individual and collective output.

## Chapter 6

# Conclusions

In wrapping up, the goal is to bring together the key idea of the thesis with the theory behind it. This means showing how logging in software development is not just about fixing problems but is linked to bigger ideas in creating and managing software. The implementation of automation within our software and CI log analysis processes has yielded significant benefits. By methodically analyzing historical data to detect recurring issues and automating JIRA ticket creation, we have streamlined issue tracking and resolution, reducing human error and saving valuable time. The 20-minute daily reduction in log analysis tasks, amounting to a 2.2% decrease in workload, underscores the efficiency gains achieved. Over time, this translates to substantial time savings, allowing the development team to redirect their efforts towards more complex and meaningful work. This shift not only boosts productivity but also enhances job satisfaction by alleviating the monotony of repetitive tasks.

Addressing the research question, "How to minimize human work in the processing of daily and weekly logs from CI?", our findings confirm that automation is not only possible but effective. By leveraging automated log analysis and integrating it with JIRA for issue tracking, we have significantly minimized the manual effort required for these tasks. The reduction in workload and the elimination of repetitive tasks demonstrate that such automation can substantially enhance efficiency and job satisfaction for development teams.

Future studies should keep looking for better ways to manage logging and to reduce the manual inefficiency, considering all the different advantages and disadvantages it brings. Continued exploration in this area is essential for developing even more sophisticated methods to handle logs, further reducing human intervention and enhancing the overall efficiency and effectiveness of software development processes.

## 6.1 Future Development

Looking ahead, there are several avenues for further enhancing our automated project management and CI log analysis system. One promising direction is the integration of AI-based systems to augment our current capabilities. By leveraging machine learning algorithms and natural language processing, we can develop an advanced AI that not only identifies recurring issues but also predicts potential future problems based on patterns in historical data. This predictive capability would allow teams to proactively address issues before they manifest, further reducing downtime and improving overall system reliability.

In addition to predictive analytics, an AI-based system could enhance the prioritization and categorization of detected issues. By understanding the context and impact of various errors, the AI could automatically assign priority levels and suggest optimal resolutions, thereby streamlining the triage process and ensuring that the most critical issues are addressed first. This intelligent prioritization would lead to more efficient use of resources and quicker resolution times.

Another transformative development could be the implementation of a fully automated system that not only logs and reports issues but also modifies the code directly to resolve certain types of errors. This system would involve sophisticated AI capable of understanding the code, identifying the root cause of issues, and applying fixes autonomously. For example, if a recurring bug is detected, the AI could analyze the affected code, determine the best corrective action, and implement the necessary changes. This self-healing capability would drastically reduce the need for manual intervention, allowing developers to focus on more complex and creative aspects of software development.

To ensure the robustness and safety of such an autonomous system, it would be essential to incorporate rigorous testing and validation mechanisms. Before deploying any code modifications, the AI system should run comprehensive tests to verify that the changes do not introduce new issues or regressions. Additionally, a review mechanism could be implemented where developers can approve or refine AI-suggested changes before they are merged into the main codebase.

Ultimately, these advancements in automation and AI integration represent the next frontier in project management and CI log analysis. By continuing to innovate and push the boundaries of what is possible, we can create even more efficient, reliable, and intelligent systems that significantly enhance the software development lifecycle. This future vision not only promises to improve productivity and reduce downtime but also sets the stage for a new era of intelligent, self-managing software development environments.



# Bibliography

- [1] D. Nyale, “Unravelling DevOps Agile Methodologies: A Comprehensive Review of Recent Research,” *International Journal for Research in Applied Science and Engineering Technology*, vol. 11, no. 11, pp. 2012–2021, 11 2023.
- [2] Megan Ferringer, “Here is the Difference Between CI/CD and DevOps and How They Work Together to Drive Innovation.” [Online]. Available: <https://www.navisite.com/blog/insights/ci-cd-vs-devops/>
- [3] MOJTABA SHAHIN, MUHAMMAD ALI BABAR, and LIMING ZHU, “Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices,” 2017.
- [4] B. Chen and Z. M. J. Jiang, “A survey of software log instrumentation,” 7 2021.
- [5] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” Tech. Rep., 2016.
- [6] L. T. Connelly, M. L. Hammel, B. T. Eger, and L. Lin, “Automated Unit Testing of Hydrologic Modeling Software with CI/CD and Jenkins,” in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering, SEKE*. Knowledge Systems Institute Graduate School, 2022, pp. 225–230.
- [7] A. Narendiran, D. Abhishek, P. Adithya, D. Ray, P. K. Auradkar, and H. L. Phalachandra, “Integrated log-aware CI/CD pipeline with custom bot for monitoring,” in *2023 8th International Conference on Cloud Computing and Big Data Analytics, ICCCBDA 2023*. Institute of Electrical and Electronics Engineers Inc., 2023, pp. 257–262.
- [8] F. Winkler and M. Westner, “A Systematic Literature Review of DevOps Success Factors and Adoption Models,” in *ACM International Conference*

- Proceeding Series.* Association for Computing Machinery, 12 2023, pp. 525–532.
- [9] S. Tatineni, “Article ID: IJITMIS\_12.01\_002 Cite this Article: Sumanth Tatineni, A Comprehensive Overview of DevOps and Its Operational Strategies,” Tech. Rep. 1, 2021.
- [10] Hashicorp, “Infrastructure as Code: What Is It? Why Is It Important?” 2023. [Online]. Available: <https://www.hashicorp.com/resources/what-is-infrastructure-as-code>
- [11] M. Rajasinghe and R. A. M. B. Rajasinghe, “CD methodology in software development teams Adoption challenges of CI/CD methodology in software development teams,” 2023. [Online]. Available: <https://doi.org/10.36227/tehrxiv.16681957.v1>
- [12] P. Rostami Mazrae, T. Mens, M. Golzadeh, and A. Decan, “On the usage, co-usage and migration of CI/CD tools: A qualitative analysis,” *Empirical Software Engineering*, vol. 28, no. 2, 3 2023.
- [13] Amity University and Institute of Electrical and Electronics Engineers, *Comparison of Different CI/CD Tools Integrated with Cloud Platform*, 2019.
- [14] S. He, J. Zhu, P. He, and M. R. Lyu, “Experience Report: System Log Analysis for Anomaly Detection,” in *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*. IEEE Computer Society, 12 2016, pp. 207–218.
- [15] J. Zhu, S. He, J. Liu, P. He, Q. Xie, Z. Zheng, and M. R. Lyu, “Tools and Benchmarks for Automated Log Parsing,” in *Proceedings - 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE-SEIP 2019*. Institute of Electrical and Electronics Engineers Inc., 5 2019, pp. 121–130.
- [16] B. Chen, “Improving the software logging practices in DevOps,” Tech. Rep.
- [17] D. A. Girei, A. Shah, and M. B. Shahid, “An Enhanced Botnet Detection Technique for Mobile Devices using Log Analysis,” Tech. Rep.
- [18] H. Aitiddir and N. Kerzazi, “Cloud Infrastructure Monitoring Using Splunk: Expectations and Challenges,” in *Proceedings - SITA 2023: 2023 14th International Conference on Intelligent Systems: Theories and Applications*. Institute of Electrical and Electronics Engineers Inc., 2023.

- [19] K. Brendel and N. Majmudar, "Survey of Automatic Online Log Parsing in Large-Scale Distributed Environments." [Online]. Available: <https://www.researchgate.net/publication/377531430>
- [20] "Datadog." [Online]. Available: <https://www.datadoghq.com/>
- [21] C. Yuan, W. Zhang, T. Ma, M. Yue, and P. P. Wang, "Design and implementation of accelerator control monitoring system," *Nuclear Science and Techniques*, vol. 34, no. 4, 4 2023.
- [22] H. Li, W. Shang, B. Adams, M. Sayagh, and A. E. Hassan, "A Qualitative Study of the Benefits and Costs of Logging from Developers' Perspectives," *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2858–2873, 12 2021.
- [23] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An Online Log Parsing Approach with Fixed Depth Tree," in *Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017*. Institute of Electrical and Electronics Engineers Inc., 9 2017, pp. 33–40.
- [24] A. Miranskyy, A. Hamou-Lhadj, and A. Larsson, "Operational-Log Analysis for Big Data Systems Challenges and Solutions," Tech. Rep.
- [25] *2018 4th International Conference on Frontiers of Signal Processing (ICFSP 2018) : September 24-27, 2018, Poitiers, France*. Institute of Electrical and Electronics Engineers, Inc., 2018.