Analysis and Monitoring of Task Allocation in DevOps Processes

by

Damien DESVENT

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS IN SOFTWARE ENGINEERING
M.A.Sc.

MONTREAL, APRIL 28, 2025

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my supervisor, Dr. Francis Bordeleau, for his invaluable support and guidance throughout my research. His expertise and thoughtful advice were instrumental in shaping this work. I am particularly grateful for his leadership of the ÉTS DevOps IRC, which made this thesis possible by providing both financial support and an incredible environment of mutual collaboration and encouragement. I would also like to thank all the researchers in the chair for their kindness, advice, and contributions that helped my research grow and evolve. Specifically, Samah Kansab gave me invaluable advice related to the writing of this thesis.

I extend my deepest thanks to Dr. Ali Tizghadam, the co-supervisor of my thesis, whose knowledge and expertise greatly advanced my work. His trust in my abilities, along with the support of TELUS and the ÉTS DevOps IRC, is fundamental to this thesis's success. This partnership not only offered financial backing but also enabled access to relevant real-world data and crucial interactions with TELUS professionals. I am deeply grateful to Dr. Tizghadam and all the TELUS collaborators who engaged with me during this journey.

I would also like to thank the École de Technologie Supérieure (ÉTS) for fostering a collaborative environment and providing high-quality resources that supported my work. A special thank you goes to the library staff for their assistance, which proved indispensable during the writing process.

Lastly, I want to express my deepest gratitude to my partner for her unwavering support. Her encouragement, patience, and understanding helped me overcome the challenges of research and thesis writing, particularly as an international student. Her presence is truly a source of inspiration and motivation throughout this journey.

Analyse et suivi de l’allocation des tâches dans les processus DevOps

Damien DESVENT

RÉSUMÉ

Dans le développement logiciel, le besoin croissant d’adaptation rapide a conduit à l’adoption des méthodologies Agile, tandis que la demande de déploiements continus et de haute qualité a favorisé l’émergence des pratiques DevOps. Ces approches mettent l’accent sur la division du travail en tâches plus petites, ce qui améliore la visibilité de l’équipe et l’organisation. Cependant, l’allocation des tâches, le processus d’attribution des tâches spécifiques à des individus, reste un défi persistant. Une allocation efficace est essentielle pour maintenir l’efficacité et optimiser les flux de travail. Bien qu’il existe des cadres théoriques et des algorithmes, ils ne s’alignent souvent pas sur les réalités de l’industrie, ce qui prive les organisations de stratégies fiables.

Cette thèse étudie l’allocation des tâches dans un contexte industriel. Au lieu de se concentrer sur des modèles mathématiques ou d’apprentissage automatique avancés, elle met l’accent sur l’obtention d’informations exploitables étroitement liées à des scénarios du monde réel. L’étude introduit des mesures et des outils pour analyser et améliorer l’attribution des tâches.

Menée en collaboration avec TELUS, un fournisseur canadien de services de télécommunications, cette étude s’appuie sur des données réelles issues de leurs projets GitLab. Une communication régulière avec les employés de TELUS a permis de fournir un contexte essentiel, d’identifier les domaines clés à améliorer et de valider les hypothèses, garantissant ainsi la pertinence des opérations de TELUS et des pratiques de l’industrie en général. Cette recherche repose sur l’hypothèse que l’analyse du *Lead Time* des tâches et l’identification des facteurs qui l’influencent peuvent améliorer l’attribution des tâches. La méthodologie comprend l’extraction et le traitement des données, la définition de métriques pour caractériser les tâches à travers de multiples dimensions, et la visualisation des résultats. Une contribution clé est la définition d’un critère de caractérisation des tâches basé sur leur capacité à respecter les estimations, ce qui sert de base pour identifier les échecs des tâches et mener des analyses de leurs causes profondes.

L’analyse de 3107 tâches provenant de 137 projets chez TELUS, a révélé un taux d’échec des tâches constant de 34% toutes tailles estimées confondues et stable dans le temps, soulignant la nécessité d’analyses et d’un suivi accru. Les conclusions montrent également que l’utilisation des colonnes d’attente fait monter le taux d’échec de 34% à 56%, une tendance plus marquée pour les tâches avec de petites estimations, soulignant la nécessité d’un suivi et d’une analyse plus approfondis de leur impact. Par ailleurs, le multitâche a été analysé en tant que facteur potentiel d’échec des tâches, mais les résultats n’ont pas été concluants, ce qui suggère la nécessité d’une étude plus approfondie. En outre, 35% des tâches échouées ont été sous-estimées, une tendance particulièrement évidente dans les petites estimations, ce qui souligne l’importance d’améliorer les processus d’estimation des tâches. Ces analyses fournissent aux gestionnaires de projet chez TELUS et dans d’autres entreprises des outils précieux pour mieux planifier et exécuter les tâches.

Cette recherche présente un cadre open-source entièrement fonctionnel pour l'analyse de l'attribution des tâches, dotant les organisations d'un outil pratique pour identifier les inefficacités. Plutôt que de préconiser des changements radicaux, elle donne la priorité à des informations exploitables et à des solutions pratiques pour l'attribution des tâches. Ces résultats constituent une base solide pour les recherches futures, en particulier pour le développement de systèmes automatisés permettant de détecter et d'atténuer les comportements inefficaces en matière d'attribution des tâches, offrant ainsi des améliorations précieuses pour les applications industrielles.

Mots-clés: allocation des tâches, DevOps, développement logiciel, analyse de tableau Kanban

Analysis and Monitoring of Task Allocation in DevOps Processes

Damien DESVENT

ABSTRACT

In software development, the increasing need for rapid adaptation has led to the adoption of Agile methodologies, while the demand for continuous, high-quality deployment has driven the rise of DevOps practices. These approaches emphasize breaking work into smaller tasks, which improves team visibility and organization. However, task allocation—the process of assigning individuals to specific tasks—remains a persistent challenge. Effective allocation is crucial for maintaining efficiency and optimizing workflows. Although theoretical frameworks and algorithms exist, they often fail to align with industry realities, leaving organizations without reliable strategies.

This thesis investigates task allocation in an industry-based context. Instead of focusing on advanced mathematical or machine learning models, it emphasizes deriving actionable insights closely aligned with real-world scenarios. The study introduces metrics and tools to analyze and improve task allocation.

Conducted in collaboration with TELUS, a Canadian telecommunications provider, this study leverages real-world data from their GitLab projects. Regular communication with TELUS employees provided essential context, identified key areas for improvement, and validated hypotheses, ensuring relevance to both TELUS operations and broader industry practices. This research is based on the hypothesis that analyzing task lead time and identifying its influencing factors can improve task allocation. The methodology involves data extraction and processing, defining metrics to characterize tasks across multiple dimensions, and visualizing results. A key contribution is the definition of a criterion for characterizing tasks based on their ability to meet estimates, which serves as a foundation for identifying task failures and conducting related analyses.

Analysis 3148 issues from 137 projects in TELUS revealed a consistent 34% task failure rate across estimated sizes and over time, underscoring the need for further tracking and improvement. The findings also indicate that the use of waiting columns raises the failure rate from 34% to 56%, a trend more pronounced in tasks with small estimates, highlighting the need for closer monitoring and analysis of their impact. Additionally, multitasking was analyzed as a potential contributor to task failure, but the findings were inconclusive, suggesting the need for further investigation. Moreover, 35% of failed tasks were underestimated, a trend particularly evident in smaller estimates—emphasizing the importance of improving task estimation processes. These insights provide project managers at TELUS and other companies with valuable tools to improve task planning and execution.

This research presents a fully functional open-source framework for task allocation analysis, equipping organizations with a practical tool to identify inefficiencies. Rather than advocating for radical change, it prioritizes actionable insights and practical solutions for task assignment. These findings lay a strong foundation for future research, particularly in developing automated systems to detect and mitigate inefficient task allocation behaviors—offering valuable improvements for industry applications.

Keywords: task allocation, DevOps, software development, Kanban board analysis

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

LIST OF ABBREVIATIONS

API	Application Programming Interface
CPU	Central Processing Unit
ETS	École de Technologie Supérieure
IaC	Infrastructure as Code
IRC	Industrial Research Chair
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ML	Machine Learning
MV	Multitasking Value
LT	Lead Time
ORM	Object-Relational Mapping
PQ	Preliminary Question
PR	Pull Request
RLT	Reference Lead Time
RQ	Research Question
SDN	Software-Defined Networking
SQL	Structured Query Language
TINAA	TELUS Intelligent Network Analytics and Automation ecosystem
TWC	Time spent in Waiting Columns

USP	User Story Point
VPN	Virtual Private Network
WIP	Work in progress

INTRODUCTION

Context

The modern software development industry requires diverse expertise and adaptable professionals to meet the evolving needs of businesses (Dima & Maassen, 2018). Companies strive for fast, high-quality development, requiring every team member to be clear on their tasks. This is where task allocation plays a crucial role. While task allocation is conceptually simple, its effective execution remains challenging due to numerous influencing factors, particularly in software development(Duggan, Byrne & Lyons, 2004).

Task allocation can take various forms. It may be assigned unilaterally by a team manager, at the moment a task is defined, or whenever a person becomes available to take on the work. Some companies adopt algorithms to introduce a more structured and scientific approach to the allocation process. Regardless of the method, task allocation is crucial for optimizing work distribution, ensuring that the right people are assigned to the right tasks.

Effective task allocation begins with understanding the work involved. The goal is not to determine an exact duration, but rather to provide an estimate of the workload and complexity associated with each task. Companies adopt different approaches to estimate the workload. Some may omit explicit estimation altogether and simply assign tasks directly, while others employ various techniques to gauge task size. These methods are often tied to Agile principles, relying heavily on human experience. Once the workload associated with a task has been estimated, the task allocation process can proceed effectively. Understanding the effort required enables the allocation of tasks to the most suitable individuals, ensuring that the distribution of work aligns with team members' skills and capacities.

In this project, we worked in collaboration with TELUS, one of Canada's main telecommunications providers, to develop methods and tools to support the analysis of task allocation in software projects. This research is conducted in the context of the Kaloom-TELUS ÉTS Industrial

Research Chair in DevOps. With a global workforce numbering in the thousands, TELUS employs several hundred people in IT roles. While the majority of these individuals are focused on operations rather than development, the company's adoption of Infrastructure as Code (IaC), Software-Defined Networking (SDN) for network infrastructure, and DevOps principles have resulted in a significant number of employees actively engaged in software development. This partnership allowed us to leverage real data and gain valuable insights into the underlying challenges through ongoing discussions with its authors.

Problem

The problem addressed by this thesis is the lack of systematic support for improving task allocation in DevOps processes. While existing research primarily focuses on distributed software development or theoretical approaches, industries like TELUS, operating in standard environments, have limited guidance for analyzing and optimizing task allocation.

Objectives

This thesis is part of a more global project that aims at developing methods and tools to enhance task allocation in DevOps processes. The objectives of this thesis are to:

- Analyze tasks and Kanban workflows to identify specific patterns and behaviors that influence task allocation;
- Define a structured approach to categorize tasks based on their characteristics to better align them with the required skills and capacity for meeting estimated completion times;
- Identify key factors that lead to effective or inefficient task allocation, providing actionable recommendations for optimizing DevOps processes.

These objectives are addressed through two dimensions: implementing relevant telemetry and analyzing self-defined metrics to extract key insights.

To achieve our objectives, we followed a rigorous methodology based on the analysis of over 3,000 issues collected from 137 projects in TELUS. This approach enabled us to address one Preliminary Question (PQ) and four Research Questions (RQs), each targeting a specific aspect of the workflow and task execution:

- **PQ1:** ***Are tasks in the TELUS workflow sufficiently estimated and completed to support reliable lead time analysis?*** As a starting point, we conducted a preliminary analysis to assess whether there is a sufficient number of tasks with valid estimates, which is a prerequisite for conducting meaningful analysis for the next RQs. Our results indicate that 72% of all tasks are estimated. This proportion rises to 88% when considering only tasks that have been moved to the *In progress* column (which indicate that work associated to tasks is started), confirming that the dataset is adequate for supporting the subsequent analyses.
- **RQ1:** ***How do task failure rates evolve over time and across estimated sizes?*** The first analysis enables us to establish a metric to categorize tasks based on their ability to meet estimates, allowing for an examination of its evolution and characteristics. The results show that 34% of tasks failed, confirming the relevance of this analysis. The failure rate remains consistent across estimated sizes and generally stable over time.
- **RQ2:** ***How does the time tasks spend in waiting columns of a Kanban board affect the overall task failure rate?*** Understanding the time spends in waiting columns, which at TELUS correspond to *Blocked*, *On Hold*, and *Waiting for External*, can reveal potential delays and inefficiencies in the workflow. Investigating this relationship helps identify whether prolonged waiting times contribute to higher task failure rates and missed delivery expectations. Findings show that 11% of failed tasks involve waiting columns, accounting for 7% of all tasks. Additionally, the use of waiting columns directly increases the failure rate from 34% to 56%, with a stronger impact on tasks with smaller estimates.
- **RQ3:** ***What is the impact of multitasking and workload distribution on the failure rate of tasks?*** Multitasking can lead to context switching, delays, and reduced focus, potentially

increasing the likelihood of task failure. Analyzing their impact helps uncover whether balancing assignments across team members contributes to more reliable task completion. However, the findings do not establish a clear relationship between multitasking and task failure, highlighting the need for further investigation.

- **RQ4:** ***How does the underestimation of task influence the likelihood of task failure?*** Underestimating tasks can lead to unrealistic expectations, time pressure, and incomplete execution. Exploring this effect helps determine whether inaccurate estimations are a significant factor contributing to task failure. The findings show that 35% of failed tasks are underestimated, with a higher tendency for underestimation in tasks with smaller estimates.

Contributions

The main contributions of this thesis are as follows:

1. **Analysis of task allocation practices** by identifying gaps in TELUS's DevOps processes, defining key metrics for task allocation, and evaluating their impact on task distribution.
2. **Identification of key factors influencing task allocation** in the TELUS context, providing a foundation for future process improvements.
3. **Development of a ready-to-use framework** that integrates a data collection tool and a pre-configured visualization dashboard, enabling efficient extraction, storage, and analysis of relevant data with provided metrics.

Throughout the study, TELUS collaborators have validated progress and conclusions by providing regular feedback. Their expertise in the business domain guided the research, ensuring informed decision-making on the approach and proper interpretation of the analyses. Additionally, their involvement allowed data validation in a specific project scope.

Thesis Outline

This thesis is structured as follows. Chapter 1 provides an overview of the thesis context. It introduces key concepts essential for understanding the thesis, covering both publicly available knowledge and specific aspects of the TELUS context. Chapter 2 presents a literature review, offering insights into the state of the art on task allocation in software development. Chapter 3 details the methodology used in the thesis, including the RQs, the data extraction, the definition of metrics, and the visualization. It also describes the technical infrastructure used in the conducted work. Chapter 4 highlights the results and findings derived from the data analysis, addressing the RQs. Chapter 5 underlines the development of a ready-to-use framework for extracting, analyzing, and monitoring tasks.

CHAPTER 1

BACKGROUND AND RELATED WORK

1.1 Introduction

This chapter establishes the essential context for understanding the rest of the thesis, introducing key concepts related to the research. It provides an overview of software development management principles, including team organization, task estimation, and task allocation, as well as major methodologies such as Kanban and DevOps. Additionally, a section is dedicated to the industrial partner, TELUS, outlining the main considerations of this collaboration.

1.2 Team Organization

Work organization is a cornerstone of software engineering, ensuring efficiency, workflow smoothness, and productivity in project execution (Nagappan, Murphy & Basili, 2008). The prevailing method in the field is grounded in Lean principles, which emphasize reducing inefficiencies. Instead of working preemptively and identifying the purpose later, this approach prioritizes responding to specific needs as they arise. This philosophy aims to maximize added value while reducing time spent on activities that do not contribute to the project's goals (Poppendieck & Cusumano, 2012).

A practical application of this principle is the division of work into manageable tasks. By analyzing needs and expectations, tasks are defined to address specific requirements. The use of finely granular tasks is essential for effective and transparent work management. This approach involves breaking the workload into numerous smaller tasks instead of a few large ones. Such granularity ensures shorter duration for individual tasks, facilitating better tracking of progress and preventing extended focus on a single activity.

These tasks enable the distribution of work among team members. The process begins with estimating the workload associated with each task. This step evaluates factors such as the effort required, the specific knowledge needed, and the task's priority. Accurately defining a task ensures it reflects the actual work involved, promoting clarity and alignment. This information, combined with contextual factors such as team members' availability, expertise, and preferences, guides the allocation of tasks to individuals (Trudel & Boisvert, 2011).

In summary, task management encompasses two fundamental processes. First, task estimation involves analyzing various factors to approximate the effort required for completion. Second, task allocation implies assigns tasks based on estimations, priorities, prerequisites, and the individual circumstances of team members. A detailed exploration of these processes is provided in the subsequent sections.

1.3 Task Estimation

1.3.1 Context

At first glance, task estimation may seem unnecessary, as it appears secondary to the primary goal of completing the work itself. However, estimation plays a pivotal role in enabling us to predict the effort required for individual tasks and, by extension, the overall effort needed for projects, milestones, or specific objectives. These predictions enable effective resources allocation while adhering to established priorities.

The complexity of estimation varies significantly depending on the nature of the task. For straightforward, repetitive tasks, such as those on a production line where processes are well established and thoroughly understood, estimations can be made with a high degree of accuracy, ensuring optimal resource allocation. Conversely, software development tasks—such as implementing a new feature or, even more unpredictably, resolving a bug—present a much greater challenge for accurate estimation. In these cases, the difficulty lies in anticipating the precise actions required and, consequently, the time and effort needed to complete the task (Albrecht & Gaffney, 1983).

Although it is possible to analyze tasks in advance to gain a clearer understanding of the associated effort, this process often demands significant time and resources, which are themselves part of the task. The challenge, therefore, lies in striking a balance between the time invested in producing an estimate and the accuracy of the resulting prediction. Achieving this balance is essential to ensuring that estimations contribute effectively to resource allocation without detracting from the overall efficiency of the work.

1.3.2 Measurement and Metrics

This complexity has given rise to a significant observation: temporal estimation, commonly employed in the industrial sector with metrics such as *man-days* or *man-hours*, proves effective when precise estimates are feasible. However, its limitations become evident when estimations are imprecise. In such cases, a flexible estimate is mistakenly converted into a rigid measurement. This can lead to various challenges, particularly for teams. When estimates are perceived as overly optimistic, teams may feel ahead of schedule and experience a subsequent drop in productivity. Conversely, when tasks take longer than estimated, teams may feel they are falling behind, fostering unnecessary stress and pressure (Goswami & Urminsky, 2020).

This situation is exacerbated by management reliance on these estimates to organize workflows and enforce deadlines. Teams may be encouraged to prioritize speed over quality in an effort to meet estimated timelines. Under such conditions, critical but indirect value-adding activities—such as testing, documentation, and code reviews—are often neglected in favor of delivering the core development work (Kim *et al.*, 2021). This prioritization not only affects the quality of the final product but also places a significant constraint on workers, leading to increased stress and a decline in opportunities for continuous learning. In such an environment, teams lack the time to acquire new skills or share knowledge, which can result in hyper-specialization and reduced adaptability among workers (Arifin, Daengdej & Khanh, 2017).

In light of these challenges, the industry has explored alternative approaches to estimation that avoid the use of explicit time measurements. The goal is to combine flexible estimation methods with equally flexible evaluation metrics, allowing for a more adaptable and less rigid approach to task and workload management.

T-shirt Sizes

One early approach to non-temporal estimation is using T-shirt sizes (e.g., S, M, L, XL). This method, rooted in Agile principles, offers several advantages: it is simple to understand and implement, and it shifts the focus from quantitative to qualitative measurement. By doing so, it minimizes the implicit association with time, encouraging teams to think in terms of relative effort rather than specific duration (Mallidi & Sharma, 2021).

However, the use of qualitative attributes is not without its limitations. One significant drawback is the increased complexity of analyzing non-quantified data. Unlike numerical estimates, qualitative measurements lack precision, which complicates data interpretation and comparison. Additionally, this approach poses challenges in establishing key performance indicators (KPIs), as these metrics often rely on quantifiable data to track and evaluate progress effectively.

User Story Points

Another approach to qualifying estimates in a non-temporal manner is the use of User Story Points (USPs). This method, also derived from Agile principles, differs from T-shirt sizes by introducing quantification. A predefined sequence of numbers is used as a reference to represent the workload associated with a task. While some teams opt for a simple sequence of integers (e.g., 1, 2, 3, 4, ...), the Fibonacci sequence is more commonly employed. The Fibonacci sequence, where each number is the sum of the two preceding numbers (e.g., 1, 2, 3, 5, 8, 13, 21, ...), introduces a non-linear scale, which reflects the increasing uncertainty associated with estimating larger tasks.

This method offers several advantages, the most significant being its ability to facilitate effective monitoring of team performance. By tracking the number of USPs a team can handle during a *sprint*[1] it becomes possible to gauge the team's capacity. This knowledge allows for optimal work planning by ensuring the assigned workload aligns with the team's capabilities (Trudel & Boisvert, 2011).

[1]A sprint is a fixed time period typically spanning a few weeks, as defined in the Scrum methodology.

However, one notable drawback of using USPs lies in the potential misuse of this method, which can inadvertently reintroduce the very issues it seeks to resolve: time-based estimation and the resulting pressure on teams. To avoid this pitfall, it is essential to recognize that methods like USPs are designed to complement the principles and philosophies underpinning frameworks such as Lean, Agile, and Scrum. These tools should never overshadow or replace these guiding philosophies but must instead be applied in harmony with them to achieve their intended benefits (Coelho & Basu, 2012).

1.4 Task Allocation

Task allocation is the process of assigning a specific unit of work to an individual. The primary goal of task allocation is to make explicit the allocation of tasks to team members and ensure that each team member clearly understands their responsibilities. While the principle itself is straightforward, the challenge lies in optimizing task allocation due to the numerous constraints that influence the process.

We organize these constraints into distinct categories to enhance clarity and understanding.

- **Material constraint**. Minimal in software development compared with other fields, they still exist with for example the need to have the right licenses or hardware.
- **Organizational constraint**. This includes task characteristics such as prioritization and size. Larger tasks, for instance, might need to be assigned earlier or allocated to more experienced developers (Tsai, Moskowitz & Lee, 2003).
- **Dependency constraint**. It exists in all projects, as certain tasks must be completed to allow work on others. For example, with the need to provide the infrastructure for a software before deploying it.
- **Human constraint**. Perhaps the most important in software development. This constraint is linked to many components such as availability, experience, knowledge, access rights, and even preferences. Indeed, some tasks require specific skills that only certain people have and people who enjoy their work are likely to perform better (Lin, 2013).

This multifaceted problem can be likened to the Knapsack Problem, a well-known algorithmic challenge (Vanderster, Dimopoulos, Parra-Hernandez & Sobie, 2009). In this analogy, the backpack represents the team's capacity, the boxes symbolize tasks of varying sizes, and the objective is to maximize the value of tasks completed without exceeding the team's capacity.

In summary, while task allocation may appear simple in theory, its implementation and optimization are anything but trivial. This complexity has made it a subject of extensive research for decades and a cornerstone in efforts to improve work processes.

1.5 DevOps and Kanban

DevOps processes typically use Kanban boards to manage the allocation and evolution of tasks during the development process. In this section, we introduce the concepts of DevOps and Kanban relevant to this thesis.

1.5.1 Kanban

Context

Kanban is a workflow management method designed to optimize value delivery throughout the process. The term "Kanban" is derived from the Japanese word for "label," reflecting the method's origins in Japan and its roots in Toyotism[2]. Originally, the principle involved attaching labels to parts on the production line to monitor progress in real time and facilitate a just-in-time approach. In its modern iteration, Kanban no longer relies on physical labels but rather on tasks displayed in a digital board. However, the core principle remains unchanged: to make work visible. Whether physically displayed on a wall or digitally on each team member's screen, the Kanban board must be accessible to all, ensuring continuous monitoring and optimal decision-making (Vacanti & Coleman, 2020).

[2]Toyotism refers to a manufacturing philosophy developed by Toyota, emphasizing efficiency, just-in-time production, and continuous improvement to minimize waste and maximize value.

Kanban boards differ depending on the company. While the presence and names of columns vary, the core principles remain consistent in most cases. The column names presented below are used by TELUS and other companies (Corona & Pani, 2013). Yet, explanations apply beyond the specific names. In this context, column names are provided throughout this section for clarity, even if their exact names may vary.

Figure 1.1 illustrates a Kanban board example. The main columns, in bold, are used in almost all Kanban boards across the industry (sometimes with a different name) while others are only often used.

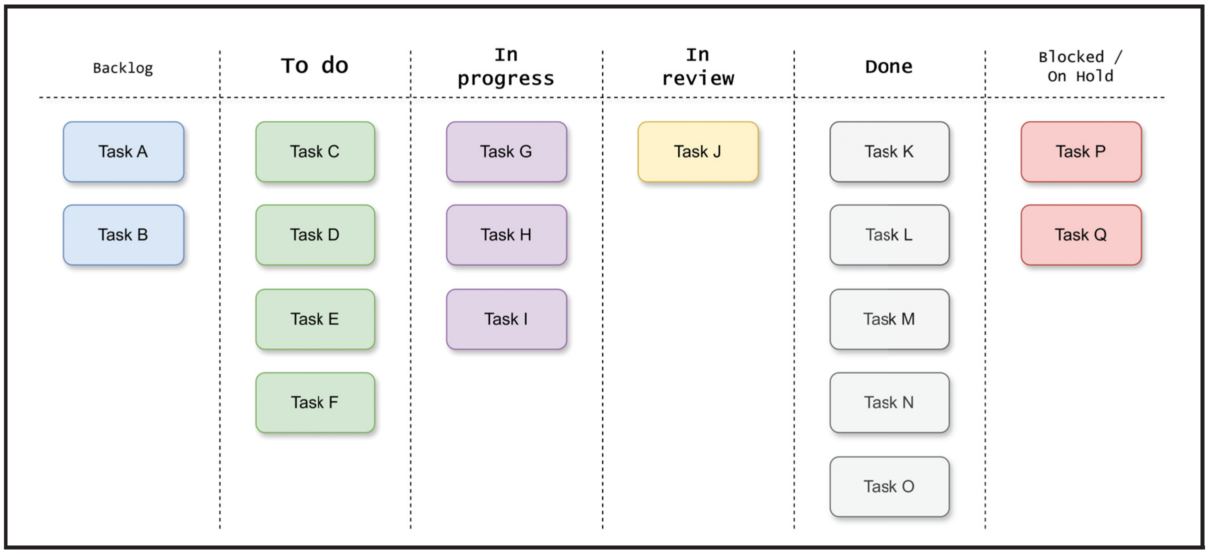


Figure 1.1 Kanban board example

Backlog Column

The first column, often labeled *Backlog*, serves as the starting point for tasks when they are created. It acts like the default column where tasks go after creation. At this stage, no work has yet been performed on the task, and there is no guarantee it will proceed; it may be deemed irrelevant or a duplicate of an ongoing task.

To do Column

Then comes the *To do* column, where tasks from the *Backlog* are moved after analysis and a decision to proceed with them in the scope of the current development cycle. Practices regarding this step vary depending on the team's workflow. In some cases, tasks remain in the *Backlog* until a triggering event, such as the start of a new project phase or a new sprint in sprint-based workflows. Regardless of the approach, the *To do* column serves as a holding space for tasks that have been confirmed but have not yet been started. Unlike the *Backlog*, this column is tightly managed to avoid the accumulation of tasks. It typically contains only a limited number of items, as the remaining tasks are left pending in the *Backlog*.

***In progress* Column**

Next is the *In progress* column, also referred to as *In development* or similar names, representing tasks that are actively being worked on. This column is crucial for tracking team activity, as it shows exactly who is working on what. It is essential to manage this column carefully and minimize task accumulation. Like the *To do* column, the *In progress* column is subject to limitations, also known as work-in-progress (WIP) limits, a key principle of the Kanban methodology. WIP limits restrict the number of tasks that can occupy a column at any given time, promoting focus and efficiency. For the *In progress* column, a common WIP limit corresponds to the number of team members plus some additional tasks to account for contingencies (Wang, Conboy & Cawley, 2012).

***In review* Column**

Once the work of a task is complete, it is moved to the review and verification stage, represented by the *In review* column (or a similarly named column). This stage encompasses both the waiting period before the review begins and the review process itself. When a team member finishes a task and submits it for validation, they move the task from the *In progress* column to the *In review* column. During this time, the task awaits reviews, and the goal is to minimize this waiting period as it represents inefficiency. Once the review begins, the task remains in the *In review* column until the process is complete. If the task is accepted after a review, it moves to the *Done* column, which houses completed tasks. Unlike the other columns, the *Done* column has no WIP limit, as no additional work is performed on tasks at this stage.

Specificities With the Review Process

The dual nature of tasks in the *In review* column—where they are both waiting for review and actively under review—can create challenges in maintaining visibility. One potential solution is to split the column into two distinct ones: *Waiting for review* and *In review*. However, teams often prefer to keep a single column for two main reasons.

The first reason is clarity and simplicity. Striking the right balance between granularity and usability is crucial. While adding more columns for precise tracking might improve detail, it can also make the Kanban board overly complex and difficult to read. Furthermore, users may be reluctant to update such a detailed board regularly, reducing its overall effectiveness. A Kanban board that is not consistently updated loses its value as a reliable tool. On the other hand, having fewer columns simplifies the board and encourages regular updates, but at the cost of reduced visibility into specific stages of progress within a column. Ultimately, the primary goal of the Kanban is to make work visible, and this balance must always be considered.

The second reason for not splitting the *In review* column is the widespread use of a separate mechanism for code-related tasks: Pull Requests (PRs). PRs provide an organized way to track, document, and support the review process when merging code changes into a shared repository. This mechanism allows developers to assign reviewers, add descriptions, and outline the work performed. Reviewers can leave comments, suggest modifications, or request clarifications directly in the PR. This system often supplements the Kanban board and ensures that the review process is well documented. If changes are being made after a reviewer comment, the task should ideally move back to the *In progress* column until the necessary adjustments are made, ensuring that the Kanban board remains an accurate reflection of the work. The term *Merge Request* is also used in some tools and represents the same mechanism. The difference only comes from a naming convention.

Done Column

The last column in the Kanban, *Done*, indicates that the tasks doesn't require any additional work and that everything needed has been done.

Additional Columns

Finally, additional columns are sometimes included to accommodate reports or processes specific to the company's operations. These often include waiting columns, such as *On hold* for tasks temporarily paused, *Waiting for external* for tasks dependent on another team, or *Waiting to deploy* for tasks ready for deployment but requiring manual intervention. These columns help capture nuances in workflows specific to the organization's context (Trudel & Boisvert, 2011).

1.5.2 DevOps

Context

DevOps is a movement that emerged two decades ago. As its name suggests, it aims to unify the work of developers and system administrators. However, DevOps goes beyond mere unification—it is a comprehensive philosophy underpinned by a set of principles. The movement originated within companies transitioning to Agile methodologies. Agile is a collection of approaches that promote iterative work processes, close collaboration among stakeholders, prioritization of individuals and interactions, and an emphasis on delivering functional software over exhaustive documentation. Adopting Agile often requires a complete rethinking of workflows, transforming outdated, rigid processes into flexible, efficient systems that embrace change.

While Agile significantly improved productivity and quality during the development phase, companies noticed a bottleneck when it came to deployment and operations. Software components could now be developed in a matter of weeks or days, but deployments often occurred only once a year, stifling the benefits of Agile. This discrepancy highlighted the need to extend Agile principles to operations, giving rise to DevOps (Kim *et al.*, 2021).

Principles and Capabilities

DevOps strives to ensure that the entire development and operations pipeline is seamless, fast, and resilient. It is founded on five key principles:

1. quality,
2. working in small batches,

3. automation,
4. continuous improvement,
5. collaboration.

These principles manifest in 36 distinct capabilities organized into six categories:

- technical,
- lean management,
- lean product development,
- transformational leadership,
- culture,
- other.

Without examining each capability in detail, DevOps encompasses both technical aspects, such as test automation, continuous integration, deployment, version control, and monitoring, as well as methodological considerations, including Lean principles, work visibility, experimentation, and feedback loops. Many DevOps principles closely align with those of the Kanban methodology, making Kanban board a natural choice for teams adopting DevOps practices (Forsgren, Humble & Kim, 2018).

Summary

The advent of DevOps has revolutionized how companies deliver software. Delivery times, which refer to the interval between writing code and deploying it into production, have been reduced from several months to mere minutes in some cases. Quality has also seen significant improvements because production considerations are now integrated into the development process from the outset. In addition, the automation of testing, integration, and deployment have greatly enhanced reliability, speed, and security. This shift allows workers to dedicate their efforts to value-creating activities rather than focusing on repetitive, time-consuming tasks that can be automated.

1.6 TELUS Context

1.6.1 Context

This thesis is conducted as part of the Kaloom-TELUS Industrial Research Chair (IRC) in DevOps, established in 2021 by Dr. Francis Bordeleau in partnership with two industry leaders: Kaloom and TELUS.

The purpose of such a chair is to leverage the expertise and practical knowledge of the industrial sector, ensuring that research is not only well founded but also highly relevant and applicable in a professional context. Specifically, this chair focuses on developing methods and tools to support the systematic improvement of DevOps processes.

This thesis adopts this applied approach and benefits from a context deeply rooted in the realities of the software development industry. It draws on real-world data from TELUS and includes regular meetings with TELUS collaborators to maintain a focus on research that directly supports and advances the professional domain, rather than diverging from it. Given TELUS's role as a key industrial partner in this research, it is essential to outline the TELUS context, as it provides a critical foundation for fully understanding the thesis.

1.6.2 TELUS Company

TELUS is a Canadian national telecommunications company established over 30 years ago. While renowned for its expertise in telecommunications, TELUS has since diversified into various industries, including health services and products, IT and customer service, and even agriculture.

The partnership with the IRC focuses on a specific department: TELUS Intelligent Network Analytics and Automation (TINAA). This department aims to optimize operations and reduce costs by automating repetitive tasks and managing network performance functionalities. Dr. Ali Tizghadam, Network Softwarisation Lead at TINAA and co-author of this thesis, describes the mission of TINAA as follows: "As the name suggests, TINAA is a kind of operating system for automation that we build on top of our current cloudified network to enable creation and control of services. TINAA also helps to create network applications that can automatically control the performance of the network under different unexpected changes. And this gets us closer to realising the dream of having a self-driving network" (Brice, 2021, para.6).

This thesis is conducted specifically in collaboration with the TINAA department, enabling a detailed study of their work processes and associated practices. The following section will outline the work tracking and management tool used in the department.

1.6.3 GitLab Tool

Our study is specifically based on data retrieved from TELUS GitLab platform. GitLab is a source code hosting and management platform built on Git[3]. It facilitates collaborative work among large teams while ensuring no information is lost. Projects serve as the organizational units, and alongside Git's version management capabilities, GitLab provides features designed to streamline workflows. This section provides an overview of the features offered by GitLab and used in the thesis context.

Issue

One such feature is *issues*, which represent tasks created in a project. These tasks include a title, description, and comments to track progress or highlight problems encountered. Issues are open by default when created, with the objective of closing them upon task completion (i.e., when the problem they address has been resolved).

To maintain a general perspective and avoid focusing solely on GitLab, the term "tasks" will be used more frequently throughout this thesis. However, the term "issues" may occasionally appear, primarily in technical contexts or when greater precision is required.

[3]Git is a distributed version control system used to track changes in files, enabling collaboration and managing code versions in software development.

Issues also allow for assigning one or more team members to specific tasks, representing task allocation. This assignment forms a crucial data point for our study.

Weight

Another important action that can be performed on issues is assigning a weight—associated with the task estimation phase. This weight represents an estimation of the task's workload, quantifying the effort required to complete it. While GitLab uses numerical values for weights, TELUS employs a T-shirt size system for task estimation, following a standardized procedure to convert T-shirt sizes into weights. This conversion is based on the Fibonacci sequence and is detailed in Table 1.1.

Table 1.1 Previous and current TELUS estimation guidelines

T-shirt size	GitLab weight
XS	1
S	2
M	3
L	5
XL	8
XXL	13

Before 2023

T-shirt size	GitLab weight
XS	1
S	3
M	5
L	8
XL	13
XXL	21

Since 2023

As observed, TELUS previously used two different guidelines. The first, which included the weight 2, is no longer in use as of 2023. The current guideline eliminates weight 2 and introduces weight 21. These changes are significant, as they may introduce bias into the data, depending on how rigorously teams have adopted the new guideline.

For consistency and clarity, the terms "estimated size," "estimated category," and "estimate" will be used interchangeably throughout the remainder of this thesis to refer to weights.

Iteration

Issues can also be linked to iterations, which are the implementation of sprints in GitLab. At TELUS, iterations typically last two weeks and help organize work by specifying the sprint during which a task is planned to be completed. Like other task details, this information is editable and often adjusted when a task is postponed—typically due to changes in priority, increased complexity, or dependencies that require additional time.

Label

Labels are commonly used in GitLab to characterize issues. These labels can be attached and detached from them and are used to indicate different kinds of information such as the priority, the scope, the related team, the severity, ... Labels are also used to group tasks into columns, thus labels are used for Kanban boards with one label for each column.

Kanban Board

Lastly, Kanban boards are used by most of the teams in TELUS. Their Kanban boards include the main columns (*To do*, *In progress*, *In Review*, and *Closed* which refers to the *Done*) and more specific columns with *Blocked*, *On Hold* and *Waiting for External*. While *Waiting for External* explicitly indicates external dependencies, such as awaiting access permissions or technical information from another team, *Blocked* and *On Hold* are generally used interchangeably to denote pauses in development due to various reasons. From a technical standpoint, Kanban columns in GitLab are implemented as labels attached to issues. This information is also included in our analysis.

There is no *Backlog* named column in TELUS Kanban boards since they do have a column provided by GitLab for open tasks that are outside of the other columns. This column is used the same as a *Backlog* column. In the same logic, they do have a column provided by GitLab for closed tasks outside of other columns. On this point the process differs across teams with some ones who use the explicitly named *Closed* column of the Kanban while others directly use the column provided by GitLab, leading to disparities we have to be aware of.

Figure 1.2 illustrates a Kanban board example specific to TELUS. The example highlights the differences between columns provided by GitLab and columns defined by TELUS using labels.

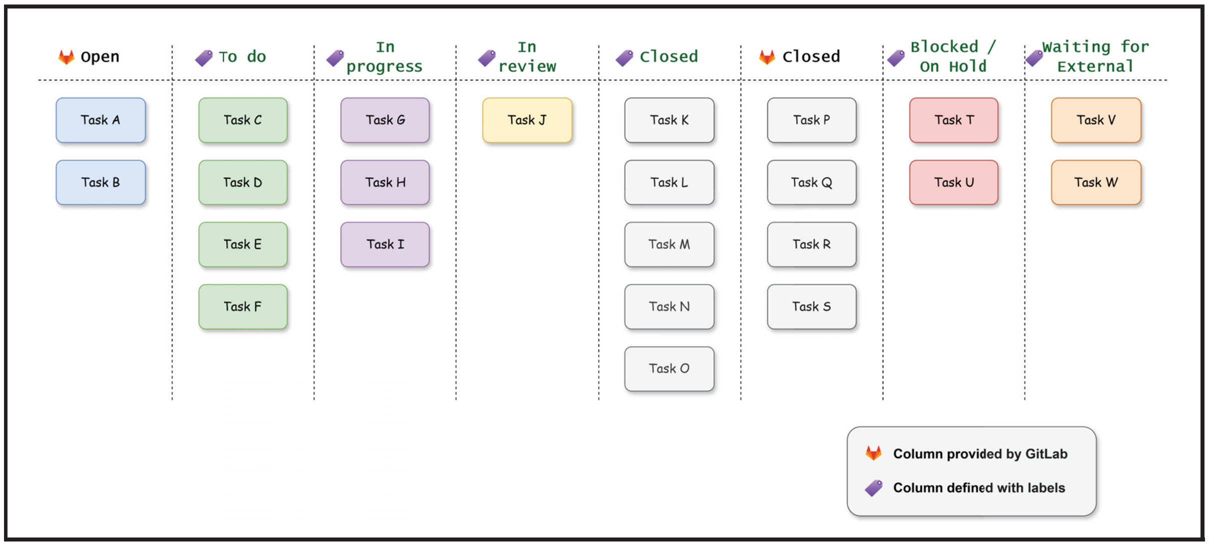


Figure 1.2 TELUS specific Kanban board example

1.6.4 Additional Consideration

The analysis is conducted on all projects in TINAA group, which consists of multiple teams and hundreds of projects. This data has been cross-verified and supplemented by discussions with team members to ensure accurate understanding and interpretation.

Another important consideration is how TELUS teams use GitLab. In the TINAA group, GitLab serves as the sole project management and task-tracking tool. Unlike companies that use generic task-tracking tools and integrate them with GitLab, in TELUS all task management—whether related to code or not—is conducted directly within GitLab. Consequently, in addition to code-related projects, the platform also hosts non-programming projects, where no files are present, and the sole purpose is to create tasks and a Kanban board. This approach standardizes the company's workflow, but it is important to note that data from these "code-free" projects is also included in the analysis, even though certain behaviors may differ in some aspects.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

Task allocation has long been a topic of interest in software development (Kraut & Streeter, 1995). Therefore, it is essential to establish context and position this thesis within the scope of related work. This chapter presents an overview of the literature on the task allocation problem and its potential solutions. It is organized into three main sections: the first one investigates related work in the context of task allocation, the second one explores general solutions proposed to tackle the task allocation problem, and the third one focus specifically on machine learning (ML)-based approaches to task allocation. Additionally, an introductory section on task estimation is included at the beginning of the chapter to provide background on one of the key components of task allocation.

This research intentionally covers topics that, while not directly aligned with the thesis, are closely related to the subject matter, offering a broader perspective on the current directions pursued in the literature. However, it does not explore methodology-related fields, such as Kanban analysis or data retrieval techniques, as the primary focus remains on the ultimate goal: improving task allocation in DevOps processes. By narrowing the scope, the thesis emphasizes actionable insights and advancements directly contributing to more effective task allocation strategies.

A summary of the articles covered in this literature review is provided in Table 2.1

2.2 Task Estimation

Task estimation plays a crucial role in software development (Albrecht & Gaffney, 1983). Although the primary objective of this thesis is task allocation, task estimation cannot be overlooked, as it serves as a prerequisite for effective allocation. For this reason, a brief overview of task estimation is necessary to provide context for the thesis.

Table 2.1 Summary of articles covered in this literature review

Topic	Article
Task estimation	Albrecht & Gaffney (1983)
	Gupta, Sikka & Verma (2011)
	Arifin *et al.* (2017)
	Mallidi & Sharma (2021)
Task allocation - context	Fabriek, Brand, Brinkkemper, Harmsen & Helms (2008)
	Amrit (2005)
	Lamersdorf, Munch & Rombach (2009)
	Imtiaz & Ikram (2017)
	Mahmood, Anwer, Niazi, Alshayeb & Richardson (2017)
	Nundlall & Nagowah (2021)
Task allocation - general solutions	Lin (2013)
	Aslam & Ijaz (2018)
	Roque, Araújo, Dantas, Saraiva & Souza (2016)
	Mayez, Nagaty & Hamdy (2022)
Task allocation - ML solutions	William, Kumar, Chhabra & Vengatesan (2021)
	Aslam & Ijaz (2018)
	Al-Fraihat *et al.* (2024)
	Nakra, Dave, Chenchala & Agarwal (2023)
	Gong, Yang, Lyu & Li (2024)

An interesting classification in software effort estimation lies in the distinction between algorithm-based and analogy-based techniques. In their article, Gupta *et al.* (2011, p. 5) compare these two approaches and explicitly state a preference: "Algorithmic effort estimation models have been proved to be deficient in many aspects." However, the article has its limitations in application for our context, as it focuses on effort estimation in software development at a broader level rather than on individual tasks. While there are similarities, these findings cannot be directly applied without careful consideration.

The comparison between time-based and effort-based estimation is also particularly relevant in the context of this thesis and TELUS processes. In their work, Arifin *et al.* (2017) compare these two methods using linear regression and scalar variability, with actual task completion times as the reference. They report favorable results for both approaches, with a slight tendency for time-based estimation to perform better. While the data is compelling, the results must be interpreted with two considerations. The first arises from the chosen reference—actual task completion time—which inherently favors time-based methods. The second one is tied to the nature of time-based estimation itself: developers can adapt their work to meet deadlines as they approach. Although this behavior may improve the accuracy of time estimates, it can lead to unintended drawbacks, such as reduced quality or increased stress levels (Mallidi & Sharma, 2021).

2.3 Task Allocation

In this section, the focus now shifts to the core subject of this thesis: task allocation. While numerous articles explore this subject and provide valuable insights for this study, research specifically aimed at improving task allocation in an industrial context is considerably more limited, with existing studies often presenting certain limitations. For the purpose of this analysis, task allocation has been categorized into three distinct areas:

1. **Context**: Articles that explore the challenges and characteristics of task allocation in software development.
2. **General Solutions**: Articles that propose frameworks, algorithms, or methodologies to tackle task allocation challenges.
3. **Machine Learning Solutions**: Articles that leverage ML algorithms to address task allocation challenges.

2.3.1 Context

Although task allocation is crucial in software development (Fabriek *et al.*, 2008), it has only recently gained significant attention in the literature. For example, the work of Amrit (2005), published just 20 years ago, is one of the earliest articles addressing this topic. This pioneering study analyzes the impact of communication between team members on task allocation. Over the years, additional studies have emerged, such as the survey conducted by Lamersdorf *et al.* (2009), that explores the state of practices used for task allocation in distributed software development. This study highlights the importance of task allocation in the industry, stating: "Task allocation is still one of the major challenges in global software development due to an insufficient understanding of the criteria that influence task allocation decisions" (2009, p. 1). However, while it provides valuable insights into current practices, it focuses on practices currently used, rather than on identifying best practices.

The studies by Imtiaz & Ikram (2017), Mahmood *et al.* (2017), and Nundlall & Nagowah (2021) seek to characterize the factors influencing task allocation in global software development. They consistently highlight the critical roles of individual expertise, organizational structure, and sourcing models. These findings align with Conway's law[1], demonstrating its relevance to task allocation. However, these studies have some limitations in the context of this thesis. While they focus on global or distributed development environments, TELUS operates in a somewhat different setting. Although TELUS has multisite teams and numerous developers working remotely—partially or fully—their environment does not fully align with the contexts described in these studies, making it difficult to apply the findings directly.

2.3.2 General Solution

Understanding the factors that influence task allocation is important, but the ultimate goal is to enhance task allocation. The predominant approach in the literature to achieve this goal is by making the task allocation process more systematic. To this end, some studies have attempted to propose frameworks and methodologies to systematize and optimize task allocation. For instance, Lin (2013) explored the concept of task allocation in distributed Agile teams, laying the groundwork for the work of Aslam & Ijaz (2018). The latter defined an algorithm for task allocation based on the principle of role-based assignment rather than user-based assignment. This approach breaks the process into two steps: developers are assigned predefined roles based on their skills and expertise, and tasks are subsequently allocated to these roles using the algorithm. While promising, this approach has a limitation—it remains theoretical and has

[1]Conway's law posits that organizations design systems in a way that mirrors their communication structures.

not yet been applied or validated in a practical industrial context. This gap between theoretical proposals and real-world implementation is a recurring issue in this field.

Other studies, such as the one by Roque *et al.* (2016), incorporate a multi-objective approach (e.g., balancing time and cost) and leverage the principle of task similarity to prioritize assigning similar tasks to the same developer, thereby improving lead time. Although the theories presented are compelling, this study has only been validated through a preliminary empirical analysis and does not provide insights into its practical application in a company.

Lastly, while not the same as task allocation, bug triage offers relevant approaches worth examining. For example, Mayez *et al.* (2022) propose a bug triage algorithm that stands out by focusing on developer workload, aiming for continuous workload normalization. This approach highlights the importance of considering developer capacity in allocation, which could inform task allocation strategies more broadly.

2.3.3 Machine Learning Solution

Other approaches to improving task allocation through systematization leverage ML algorithms with the aim of creating fully automated processes.William *et al.* (2021), for instance, propose a method for automatic task allocation and compare several techniques. They use an open source employee record dataset from Kaggle Repository and set up a machine learning model that uses some factors (service year, training year, education, ...) to predict the best person for task allocation. Their results show a good accuracy (more than 95%) but can be subject to bias. For instance, in single-person projects, the model could achieve 100% accuracy by relying solely on the project ID. As a result, the study is promising but a lack of external validation makes it difficult to use this solution in an industrial context.

An intriguing advancement is presented by Shafiq, Mashkoor, Mayr-Dorn & Egyed (2021), who implement the framework proposed by Aslam & Ijaz (2018) and even develop a Jira extension for task allocation. While this article delivers a valuable and tested solution, its practical adoption in a company poses challenges. It requires the definition of roles——possibly on a per-project basis—and the use of these roles to allocate tasks to team members. Although the framework is technically sound, its industrial relevance is questionable. Implementing such a system for the sake of knowing that, for example, tasks involving *HTML*[2] should be assigned to a front-end[3] developer may not justify the effort. Similarly, Al-Fraihat *et al.* (2024) propose enhancements by comparing ML techniques to improve the performance of the previously proposed algorithm. Yet again, while the technical aspects are promising, the approach struggles with practical relevance in an industrial setting.

A particularly noteworthy contribution comes from Amazon researchers, as discussed by Nakra *et al.* (2023). They propose a fully automated task allocation framework that includes task creation from project requirements, assignment, and even automatic verification of the process. While the technical achievement is impressive, the solution is in its infancy, having been tested only on auto-generated data. Once more, the primary challenge lies not in the technical feasibility but in its real-world applicability in an industrial context.

Other domains offer additional perspectives. For instance, Gong *et al.* (2024) explore a two-stage reinforcement learning approach for task allocation. While this technique is technically relevant to our context, the algorithm is applied to assign tasks to robots in manufacturing plants, which is not directly suitable for our purposes.

2.4 Summary

To conclude, this literature review highlights that the studies offer various resources for task allocation, ranging from general concepts to abstract models and even implemented solutions. However, most of these studies focus on distributed development, which is highly specific and does not align with our context. Moreover, while many solutions provide valuable and technically sound results, they are often overly theoretical, making them a challenge to implement or less

[2]HTML is a language used to create web pages.

[3]The development tasks are often split into two categories: front-end for the visible part like the user interface and back-end for the invisible part like the processing and the storage.

relevant in an industrial setting. Crucially, none of the studies succeed in demonstrating an improvement in task allocation in a real-world context based on their findings.

This thesis situates itself within the literature more as a contextual exploration rather than a solution-oriented approach. The goal is not to develop a solution but to provide actionable insights that enhance task allocation. As such, technically complex solutions are not the focus. Instead, the research remains closely tied to the industry, leveraging the TELUS context to ensure practical relevance. The priority is to deliver meaningful, industry-applicable findings rather than pursuing an idealized theoretical solution.

It is important to note that this literature review is not exhaustive. Further analysis, particularly in non-distributed development contexts, could be beneficial to establish a more comprehensive state of the art in our specific setting. Additionally, exploring the state of the practice in major tech companies could provide valuable insights and industry-relevant findings.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter describes the methodology followed in this thesis to address the research problem. As shown in Figure 3.1, we adopt a rigorous and structured approach to achieve our objectives. The first step in analyzing task allocation involves extracting the appropriate data required for testing hypotheses and conducting meaningful analysis. For this, the data must be accurate, up-to-date, and relevant to the context of the study.

Since the Kanban system at TELUS contains a wide range of information, we carefully filter and select only the data that is directly related to task allocation and its relevant research dimensions. To support this, we implement a reliable data extraction mechanism that integrates smoothly with TELUS's infrastructure and ensures continuous and consistent data extraction. After collection, the data is processed into meaningful objects and stored in a persistent and accessible format to support long-term analysis. The final step involves visualizing the data to transform it into actionable insights. This is achieved through the design and implementation of interactive dashboards and telemetry systems, which support exploratory data analysis and interpretation.

3.2 Research Questions

In line with the project's overall objective of improving task allocation, the conducted work in the thesis aims to provide an approach to analyze task allocation. However, the study of related work in this field and preliminary researches on task allocation reveal a lack of support for task allocation analysis.

Therefore, the thesis is not intended directly to improve task allocation, but rather to provide methods, metrics, tools, and results as a basis to enable the improvement of task allocation.

This objective leads to the following Research Questions (RQ):

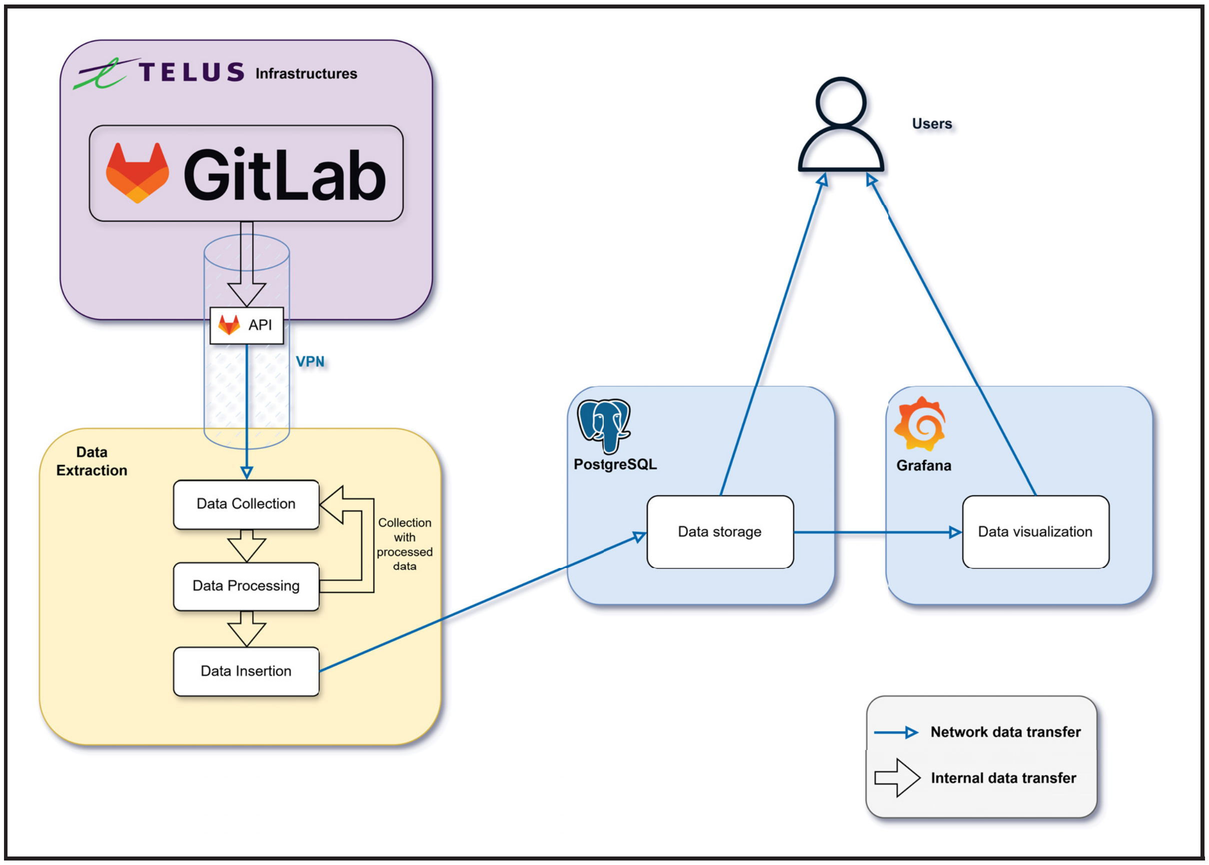


Figure 3.1 Proposed approach for data analysis

- RQ1: How do task failure rates evolve over time and across estimated sizes?
- RQ2: How does the time tasks spend in waiting columns of a Kanban board affect the overall task failure rate?
- RQ3: What is the impact of multitasking and workload distribution on the failure rate of tasks?
- RQ4: How does the underestimation of task influence the likelihood of task failure?

3.3 Data Extraction

3.3.1 Data Collection

As previously mentioned, TELUS teams use the GitLab tool, which is hosted on their internal servers. The first step in extracting the required data is identifying a method to connect to GitLab. For this purpose, GitLab provides an Application Programming Interface (API)[1], which exposes a significant amount of data for external use. To ensure security, the entire TELUS GitLab platform, including the API, is accessible exclusively through a Virtual Private Network (VPN)[2]. This VPN enables access to TELUS servers via a secure entry point that incorporates multiple layers of encryption and authentication.

In addition to VPN protection, the GitLab API is inherently secure, as it only provides access to information that the requesting user is authorized to view. Specifically, a request without a designated user token will only return publicly available information from the GitLab instance, meaning access is limited to non-restricted content. To retrieve more comprehensive data, a token must be created and associated with the user's GitLab account. This token specifies the permissions it holds (e.g., API read access) and is included as a header in the API request. Since the token is tied to the account of the user who created it, it acts on their behalf. For example, if the user has access rights only to projects in a specific group, the token's requests will only be able to return data from that group.

To effectively interact with the GitLab API, it is also essential to identify the appropriate API endpoints for the required data. For this study, we collected information related to projects, issues, Kanban boards, and users. Additionally, annotations made on issues were retrieved. While some of this information is directly accessible through specific API endpoints, other data required cross-referencing or preprocessing. The complete list of API endpoints used is detailed in Table 3.1.

[1]An API is a software component designed to enable communication between different software components.

[2]A VPN establishes a secure digital connection between a user and a remote server, ensuring encryption and access control.

Table 3.1 API endpoints for data collection from TELUS GitLab

Kind of data	**GitLab API endpoint**
Projects information	*/projects*
General issues information	*/projects/:project_id/issues*
Cloned issues information	*../issues/:issue_id/discussions*
Assignees information	*../issues/:issue_id/discussions*
Issues state information	*../issues/:issue_id/resource_state_events*
Issues weight information	*../issues/:issue_id/resource_weight_events*
Issues labels information	*../issues/:issue_id/resource_label_events*
Kanban information	*../issues/:issue_id/resource_label_events*
Users information	*/users*

To collect assignment information for tasks, the *issues* endpoint provides details about the individuals currently assigned to an issue. However, this approach only captures the present state and does not provide insights into the history of assignments. To address this limitation, we opted to extract data from the discussion section of each issue. When an assignee is added or removed, GitLab automatically generates a system message in the discussion thread (e.g., "assigned to John Doe" or "unassigned Jane Smith"). By parsing these system-generated messages, we can reconstruct the full timeline of assignment changes. This approach allows us to track when individuals were assigned or unassigned from an issue, providing a comprehensive view of how the task assignment evolves over time.

Similarly, for the status and weight associated with each issue, the *issues* endpoint offers only the current state without capturing historical changes or identifying who made those changes. To overcome this, we also used dedicated endpoints designed to provide the required historical data, including details of the users who performed these actions.

As explained earlier, in GitLab, Kanban columns correspond to labels. At TELUS, these column-related labels follow a consistent naming convention with the prefix *Status* and indicate the current position of a task within the Kanban workflow (e.g., *Status::To Do*, *Status::In progress*, *Status::Closed*). To track task movement across columns, we use the *resource_label_event* endpoint, which provides a complete history of label modifications. This includes timestamps, the specific status labels added or removed, and the user responsible for each change. For instance, if a task moves from *Status::To Do* to *Status::In progress*, the event history records both the removal of the former label and the addition of the latter.

Once all data is collected through API requests, the next step consists in processing and transforming it into actionable and meaningful information.

Dataset
The dataset resulting from the data collection comprises 3148 issues created between May 13 and November 10, 2024 from 137 projects in the TINAA group.

3.3.2 Data Processing

After collecting all data from GitLab, a significant gap remained between the raw JSON responses from the API and the structured format needed for effective analysis. While it is technically possible to store the raw data directly, doing so would necessitate extensive processing during data requests and analysis, resulting in inefficiencies, increased resource consumption, and repeated computations. To overcome this, we aim to convert the raw data into clean, structured, and analysis-ready formats.

This process involved defining meaningful data structures (objects[3]) that capture key elements of the GitLab data collected earlier. These structures are designed to match the raw data format while also making it easier to analyze and interpret. Once defined, the raw JSON data is parsed and organized into these structured representations.

Using this structured approach offers two main benefits: it simplifies working with the data and makes the information easier to understand, even when the original API responses are complex or nested. By organizing the data into a clear and consistent data format, we reduce duplication, improve efficiency, and enable more straightforward analysis. A class diagram describing the

[3]An object is a structured data entity that groups related attributes together, representing a real-world concept or entity. It is an instance of a class, which defines its attributes and structure.

main data classes and their relationships is provided in Figure 3.2. All classes and its attributes are described in details in Table 3.2.

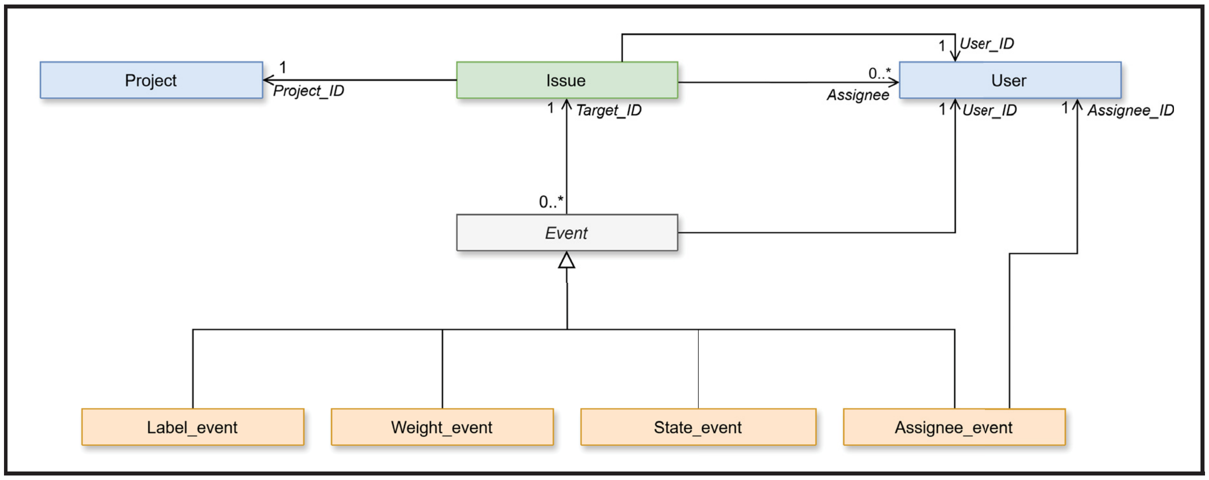


Figure 3.2 Class diagram of extracted data

Table 3.2 Objects used to represent collected data

Object	Description	Attributes	Attribute description
Project	Global information about GitLab projects	ID	ID of the project
		Name	Name of the project
		Name with namespace	Name of the project with its namespace
		Namespace ID	ID of the namespace
		Created at	Date of creation of the project
		Last activity at	Date of the last activity on the project
Issue	Specific information for each issue	ID	ID of the issue
		Project_ID	Project ID of the issue
		Title	Title of the issue
		Description	Description of the issue
		Weight	Weight of the issue
		State	State of the issue
		Created_at	Date of creation of the issue
		Closed_at	Date of the end of the issue
		User_ID	ID of the user who created the issue
		Assignee	List of assignees of the issue
		Cloned_from	ID of the issue from which this issue is cloned
User	Information about the users	ID	ID of the user
		Username	Username of the user
		Name	Name of the user
		State	State showing whether the user is active or not

Continued on next page

Object	Description	Associated attributes	Attribute description
		Is_bot	Whether the user is a bot or not
Label _event	Additions and removals of labels (including Kanban columns)	ID	ID of the event
		User_ID	ID of the user associated with the event
		Target_ID	ID of the target resource.
		Target_Type	Type of the target resource
		Created_at	Date of event
		Action	Action performed in the event
		Label_ID	ID of the label associated with the event
		Label_Name	Name of the label associated with the event
		Label_color	Color of the label associated with the event
		Label_description	Description of the label associated with the event
Weight _event	Evolution of the defined weight for an issue	ID	ID of the event
		User_ID	ID of the user associated with the event
		Target_ID	ID of the target resource
		Created_at	Date of the event
		Weight	Weight set during this event
State_event	Evolution of the state (open or closed) for an issue	ID	ID of the event
		User_ID	ID of the user associated with the event
		Target_ID	ID of the target resource
		Target_type	type of target resource
		Created_at	Date of the event
		State	State associated with the event
Assignee _event	Evolution of allocation made to issues	ID	ID of the event
		Target_ID	ID of target resource
		Author_ID	User ID of the event author
		Assignee_ID	User ID related to the assignment
		Action	Action of the event
		Created_at	Date of the event

For some objects, such as **Project**, **Label_event**, **Weight_event**, and **State_event**, the conversion from API responses to objects primarily involves basic processing (date parsing, data cleaning, ...) to ensure proper formatting. This also includes tasks like converting empty strings to null values and filtering strings to remove special characters that could cause errors.

For **User** object, in addition to standard formatting, an extra step is required to determine whether a user is a human or a *bot*. Indeed, in TELUS context, some issues are created automatically through tools like request portals (feature requests, e.g., "Add dark mode to the user dashboard") or bug reporters (error tracking, e.g., "Button unresponsive"). These issues are created in GitLab by a dedicated system user, commonly referred to as a bot, which is not a human contributor. The GitLab API does not explicitly label users as bots in the JSON response. However, it provides a filter option, *without_project_bots*, which can be set to *true* to exclude bot accounts from the results. By making two API requests—one with the filter enabled and one without—and comparing the returned user lists, we can infer which accounts are bots. This information is then stored as a boolean attribute, *is_bot*, within the user object.

For **Issue** object, in addition to the standard formatting, further processing involves analyzing the discussions associated with each issue. Specifically, the first event in a discussion is checked to determine whether it indicates issue cloning. If cloning is detected, the attribute *cloned_from* is populated with the source issue ID; otherwise, it is set to null. This step is necessary because the GitLab API does not directly provide information about the issue cloning.

For **Assignee_event** object, the lack of a direct API endpoint necessitates a different approach (as noted in Table 3.1). The solution is to extract assignment information by analyzing the discussions associated with tasks. Discussions have the advantage of containing a complete history of changes to an issue, including assignment and unassignment actions. Furthermore, the API includes a dedicated endpoint for discussions, enabling the retrieval of all assignment events for all tasks.

However, this method has two significant limitations. The first is the inability to retrieve assignment information for users who are no longer present in GitLab. This limitation arises from the nature of discussions, which are primarily designed for human readability rather than machine processing. Thus, discussions display the usernames of workers instead of their user IDs (the unique identifier used in other objects). To obtain the corresponding user ID from a username, a cross-referencing process is conducted using the user list. Since both the username and user ID fields are unique, this process is straightforward but cannot be executed if the worker is no longer listed as a GitLab user. One potential workaround could involve using usernames instead of user IDs throughout the system. However, as all other API endpoints provide user IDs, using the username would cause the same drawback. Therefore, the decision is made to proceed with cross-referencing and to exclude deleted users from consideration. This decision is justified by the fact that deleted users represent a small proportion of the overall data, particularly in recent tasks, which are the primary focus of this study.

The second drawback is the intensive processing required by this approach. As mentioned earlier, assignment events must be identified by cross-referencing discussion data with user data. Additionally, because discussion events are not formatted for software processing, their structure includes verbose sentences that describe which users were assigned or unassigned to tasks. Parsing these sentences to extract meaningful information is challenging. The sentences vary in structure depending on the events they describe (e.g., "assigned to John, Mickael, and Kelly and unassigned to Georges."). Some refer to a single user, others describe multiple users for a single type of event (assignment or unassignment), and yet others combine multiple users and multiple event types (some users being assigned while others are unassigned). This variability significantly increases the complexity of extracting information.

To address this challenge, a rule-based solution is implemented. While the variety of sentence patterns adds complexity, all patterns are predefined and known, enabling the creation of rules to detect and separate each pattern for accurate information extraction.

Once all raw data has been processed into meaningful information, it can be stored in a persistent format for long-term use.

3.3.3 Data Insertion

After collecting and processing the data, the next step is to store this data in a manner that ensures easy and long-term accessibility.

Since the data is organized into objects with well-defined attributes, a relational database is an appropriate choice for storage[4], as it supports structured querying and preserves relationships between entities.

[4]A relational database is a structured database that organizes data into tables with rows and columns, using relationships to link related data efficiently.

To align with the object-oriented design of the application and follow best practices in software development (Torres, Galante, Pimenta & Martins, 2017), an Object-Relational Mapping (ORM)[5] framework is used to serve as the interface between the application and the database. Using an ORM eliminates the overhead of manually converting objects to SQL queries, making the codebase easier to write, maintain, and extend. It also helps minimize object-relational impedance mismatch (Colley, Stanier & Asaduzzaman, 2018).

3.3.4 Speed Improvement Techniques for Data Extraction

The proposed approach enables us to meet all our requirements, successfully collecting, processing, and inserting data in the database without any problem. However, the sheer volume of data (nearly one thousand projects and thousands of issues) combined with the need to send multiple API requests to GitLab for each issue makes the process highly time-consuming. The entire execution can take up to six hours, which introduces practical limitations such as VPN session timeouts. To address these challenges, several solutions were investigated and implemented to improve the approach's overall speed and efficiency.

Filtering Using a Starting Time

The first improvement stems from the fact that not all data need to be retrieved during every execution. Some data, particularly older information, may be less relevant. Additionally, once the tool has retrieved data from prior runs, most of the information is already available, and only new or updated data needs to be fetched to stay current. To address this, a filter based on a starting date is implemented. This filter ensures that only issues updated after a specified date are retrieved. The GitLab API supports this functionality through the *updated_after* parameter in the *issues* endpoint, which allows fetching only issues that have been updated since the provided date.

This enhancement ensures that only relevant issues are retrieved, significantly reducing the volume of unnecessary API calls. Furthermore, the benefits are enhanced by reducing the

[5]Object-Relational Mapping (ORM) links database tables to class objects in code, enabling database interaction through object-oriented programming.

number of all issue-dependent API calls (state, weight, label and assignee). Unfortunately, the *projects* API endpoint does not benefit from this improvement. Although the *projects* API endpoint does include a similar *updated_after* parameter, it only accounts for updates made directly to projects, excluding changes to issues in those projects. Filtering projects based on this parameter could lead to unintended exclusions of updated issues. However, this limitation is mitigated by the efficiency of the project's API, which retrieves information for multiple projects simultaneously, ensuring a relatively quick process even when all projects are included.

Parallelization

Retrieving only desired data strongly improves speed, but sometimes there is still a large amount of data that has to be retrieved, and the limitation of some API endpoints, which send only one object per request, still implies thousands of API calls. By analyzing the time consumed by the tool, we observe that most of it is wasted waiting for API response. In contrast, all processing performed is comparatively very fast. This observation leads to the decision to implement parallelization for handling requests.

For our needs, which involves significant input and output (I/O) operations, a relevant solution is the use of shared memory and *threads*[6] within the same *process*[7] (Lin & Snyder, 2008). This approach is preferred over separate memories in multiple processes, which are better suited for computational tasks. Threads are part of a process and enable the execution of multiple tasks simultaneously. For instance, when you download something using an internet browser, a thread manages the download while another thread handles navigation, ensuring the browser remains responsive. Threads allow for pseudo-parallelization, also known as concurrency, achieved by rapidly switching the CPU's focus among threads (Birrell, 1989). Consequently, there is no computational improvement with multithreading, but it is well suited for tasks that involve waiting times, such as API calls. An illustration of the difference between concurrency and parallelism is available in Figure 3.3 to help better understand the concept.

[6]A thread is a sequence of instructions given to the CPU by a program.

[7]A process, in this context, is the instance of a computer program in execution.

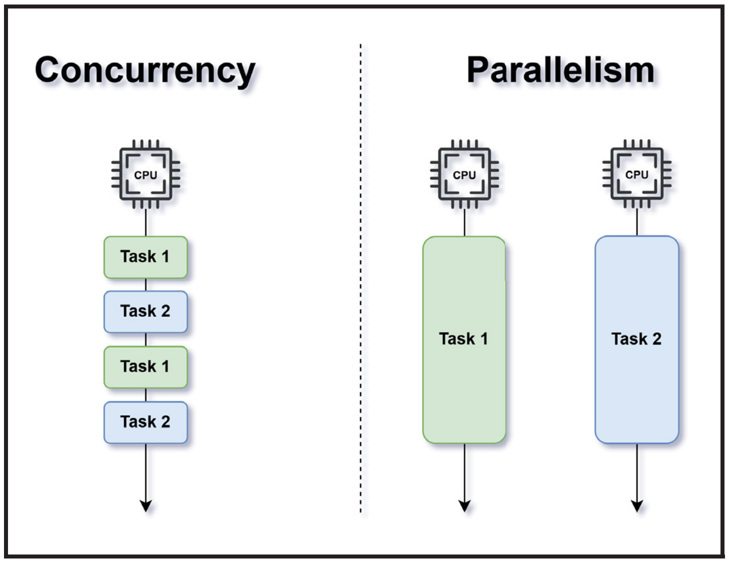


Figure 3.3 Concurrency over parallelism schema

In the proposed approach, as previously mentioned, certain API calls must occur in a specific sequence due to dependencies. For example, user information must be retrieved before fetching assignment data, as the process requires comparing usernames with user IDs. These dependencies are illustrated in Figure 3.4, which outlines the relationships between API calls and emphasizes three main stages that must be executed in the correct order.

The addition of these constraints, combined with the high volume of API calls required for each endpoint, led to the decision to parallelize requests only in the same endpoint. In other words, for a given API endpoint (represented by a box in Figure 3.4), multiple API calls are executed concurrently, but no calls for other endpoints are initiated simultaneously. This approach has the advantage of being less complex to implement while being sufficiently fast for the application's requirements. Implementing multithreading with a large number of threads among API endpoints could demand more resources and potentially cause errors, such as exceeding API quotas[8]. However, it is theoretically feasible to parallelize all requests across endpoints while respecting the dependencies outlined in the three colored steps of Figure 3.4.

[8]APIs typically enforce user quotas to limit the number of requests per second or hour. These restrictions are designed to ensure security and maintain performance by preventing a single user from overwhelming the system with excessive API calls.

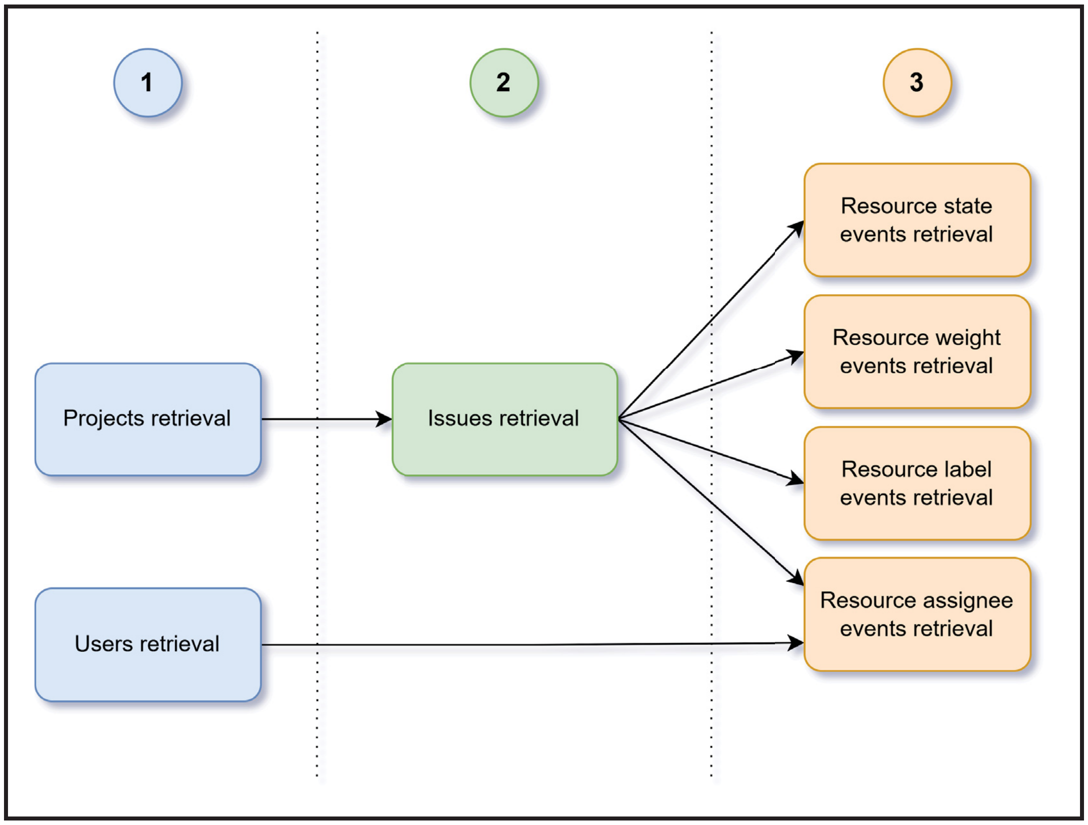

1
2
3
Projects retrieval
Users retrieval
Issues retrieval
Resource state events retrieval
Resource weight events retrieval
Resource label events retrieval
Resource assignee events retrieval

Figure 3.4 Dependency diagram for API calls

Continuous Extraction

Another possible improvement is to enable continuous data extraction.

Yet, for the purposes of this study, continuous extraction is not essential, and therefore this improvement is not used in our methodology.

The next section presents all defined metrics in the thesis based on the collected data.

3.4 Metrics Definition

3.4.1 Lead Time

Lead Time (LT) is defined as the duration required to complete a task. It is measured from the moment work begins on the task until its completion. In the context of Kanban, LT begins when the task is first moved to the *In progress* column and ends when it reaches either the *Done* or *Closed* column.

To evaluate task performance, we focus on identifying whether a task has been completed within an acceptable timeframe, using LT as a key indicator of execution efficiency. In the literature, LT is commonly used as a proxy for the effort spent on a task (Duc Anh, Cruzes, Conradi & Ayala, 2011). Since tasks can vary in size, we assume that different task sizes require different amounts of effort. Analyzing all tasks together, regardless of size, may lead to misleading conclusions, as longer LTs for smaller tasks may be misinterpreted as inefficiency.

3.4.2 Reference Lead Time and Success/Failure

To address this, we introduce the concept of *Reference Lead Time (RLT)*, which represents the median LT of all tasks in a given estimated category. A task is considered successfully completed if its LT is below the RLT for its size. We use the median as the threshold because it is robust to outliers, provides a reliable central tendency measure, and is widely adopted in similar studies (de Almeida, Lounis & Melo, 1998; El-Emam, Goldenson, McCurley & Herbsleb, 2001).

However, discretizing success and failure based on a strict threshold can introduce noise (Rajbahadur, Wang, Kamei & Hassan, 2019), particularly for data points near the threshold boundary. To mitigate this, we introduce a tolerance margin of 40% around the RLT. Tasks falling within this margin are excluded from the binary classification to reduce the impact of borderline cases and improve the reliability of the analysis. For example, if the median RLT for tasks of size 1 is one day, we apply a tolerance margin of 40%, resulting in an upper threshold of 1.4 days. Tasks completed in less than 1.4 day are classified as successful, those taking more than 1.4 days as failures.

An illustration of the success/failure notion is represented in Figure 3.5.

Exclusion of not estimated tasks

The application of success/failure criteria, as defined in this study, necessitates the exclusion of

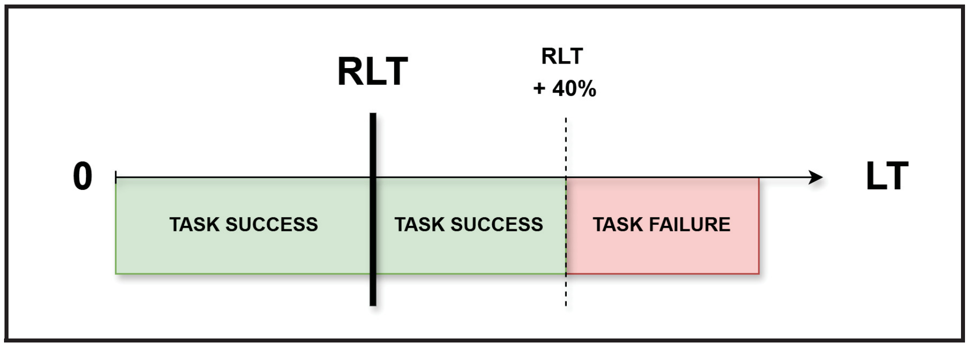


Figure 3.5 Success and failure depending on the LT

all not estimated tasks from the analysis. This exclusion is due to the inability to calculate a RLT for such tasks. Even if it were possible, the results would lack accuracy, as task allocation analysis requires a reliable reference to ensure meaningful comparisons. One potential solution could involve leveraging mechanisms like clustering to generate artificial references. However, given the risk of introducing bias into the analysis and the relatively low proportion of not estimated tasks (12%), this limitation will not be addressed at this stage.

Additionally, this decision is supported by a preliminary analysis of not estimated tasks, which reveals that they are largely specific to certain projects and types of work. This specificity further emphasizes the need to exclude these tasks, leaving them for potential exploration in a future, dedicated analysis.

3.4.3 Time Spent in Waiting Columns

To address RQ2, we need to calculate the time that a task spends in *Waiting columns*, which at TELUS include *Blocked*, *On Hold*, and *Waiting for External*. For this purpose, we define a metric called Time spent in Waiting Columns (TWC), which is the total time spent across all of the waiting columns.

3.4.4 Workload and Multitasking Value

To investigate RQ3, there is a need to define metrics associated with multitasking. Therefore, two metrics are defined: the *workload* and the *Multitasking Value* (MV).

To effectively analyze multitasking, the first step is to establish a reliable method for calculating the number of tasks that an individual handles at any given time. This metric is referred to as the *workload*. It includes all tasks that are assigned to a person that are between the Kanban columns *In progress* (included) and *Done* (excluded).

However, the workload is not related to specific tasks, making its analysis in relation to success/failure difficult. Therefore, another metric is used, this time to describe, for a task, the number of tasks already assigned to the person allocated to it when the allocation is performed. This is referred to as the MV. For example, if John works currently on two tasks (meaning that he is currently assigned to two tasks for which work is in progress), he currently has a workload of 2. Now if a third task is assigned to him: this third task would have a MV of 2. It is important to get the difference between these two concepts. While workload refers to the total number of tasks a person is handling at a specific moment, MV is task-specific. It represents the workload of the individual assigned to a particular task, providing insight into the context in which that task is being managed.

3.4.5 Underestimation

RQ4 focuses on the notion of task underestimation. For this purpose, we define a metric, simply called "underestimation", as a boolean. This metric takes the value "true" if a task has failed, but would have been a success if it had been estimated one-size larger.

3.5 Visualization

Once having all the data retrieved, processed and stored into meaningful data structures, it is possible to use it. Data visualization allows its analysis and the finding of meaningful insights. Therefore, the visualization is done in two steps:

1. The first one by visualizing global statistics related to the dataset. This is useful to understand the context of the data (from which projects, which groups, when the issues have been created, ...). This first step aims not to directly bring value added but to give support for exploration and analyses. Making it an essential prerequisite for this thesis.
2. The second one by use defined metrics to explore, investigate, and answer our RQs. These metrics are the basis for value-added visualizations. These visualizations are the sources for the set of results presented in the next chapter.

3.6 Technical Infrastructure and Tools

This section outlines the key technologies used for data extraction, processing, storage, and visualization.

3.6.1 Data Extraction and Processing

The data extraction process is handled by a custom tool developed in *Python*[9], which communicates with the TELUS GitLab API to retrieve relevant data. The collected data is then parsed, structured, and stored in a relational database for analysis.

To implement parallelization (multithreading) in Python, the *concurrent.futures* module is used. This module provides a high-level interface for asynchronous executions. Asynchronous execution allows tasks to be performed out of sequence, rather than in a blocking or synchronous manner. This provides the basis for enabling concurrency, and consequently, pseudo-parallelization. The module supports both multithreading and multiprocessing; however, as previously explained,

[9]Python is a high-level programming language known for its readability and versatility.

multithreading is more suitable in our case. Implementation is achieved using *ThreadPoolExecutor*, an object capable of creating and managing threads for asynchronous operations.

3.6.2 Data Storage and Management

The selected database system is *PostgreSQL*[10], chosen for its strong integration with Python-based tools and support for structured queries. To link the data extraction tool and the database, *SQLAlchemy* is chosen as the ORM framework due to its extensive community support, robust compatibility with Python, and seamless integration with PostgreSQL.

No dedicated database management interface is used in this project, as there is no need for manual editing of tables. Instead, database schema management is handled automatically through the *SQLAlchemy* ORM (Object-Relational Mapping) framework. If the structure of the data changes—such as when adding a new attribute to an object—the existing data is considered outdated. In such cases, the entire dataset is re-extracted to maintain consistency. SQLAlchemy ensures that database tables are created automatically based on the Python object definitions, allowing for a seamless mapping of object attributes to database columns.

3.6.3 Deployment and Persistence

The PostgreSQL database is deployed locally using a *Docker*[11] container. This setup facilitates fast deployment, version control, and avoids dependency issues. While the system currently uses a local deployment, the architecture is flexible and can be easily adapted to connect with an external database server if needed, without requiring structural changes.

[10]PostgreSQL is a powerful open-source relational database known for its stability, performance, and compatibility.

[11]Docker is a platform that uses containerization to package and deploy software in isolated environments, ensuring consistency and portability.

3.6.4 Visualization

For visualization, the system integrates with *Grafana*[12], which provides interactive dashboards for exploring the stored data. Grafana queries the PostgreSQL database and displays relevant metrics and trends, supporting deeper analysis and interpretation.

Beyond offering tools for creating graphs and charts, Grafana leverages dynamic dashboarding. This dynamic capability allows for continuous updates, the use of variables to filter data efficiently, and the ability to focus quickly on specific subsets of information. This dynamism is a significant advantage compared to static tools, where manual intervention is required to filter or modify data from different perspectives.

Limitations

Using Grafana to connect to the database and visualize data from SQL queries works effectively for a wide range of purposes but comes with inherent limitations. The most significant challenge stems from the reliance on direct SQL queries in Grafana. This approach introduces two major drawbacks:

1. **Encapsulation of added value**: Since all the data transformations and visualizations are confined to Grafana, the resulting insights and knowledge are not easily transferable to other tools or systems.
2. **Dependence on SQL for complex analyses**: While SQL is powerful for querying relational databases, it is not optimized for advanced data processing tasks. Complex queries often grow to exceed 500 lines, making them difficult to understand, adapt, and debug.

To address these limitations, one practical solution is to use *SQL views*[13]. By implementing views for commonly used transformations, repetitive tasks can be minimized, and a shared base layer for multiple queries can help reduce overall complexity. This approach has the added advantage of keeping the processing logic in the database, which is already a familiar and managed environment.

[12]Grafana is an open-source platform for monitoring and visualizing time-series and structured data through customizable dashboards.

[13]SQL views are virtual tables created from predefined queries.

Despite these improvements, SQL remains ill suited for handling extensive data processing or advanced analytics. A potential next step would involve developing a dedicated processing layer capable of retrieving data from the database, performing complex operations, and exposing the results through an API. Such a solution would offer several benefits, including support for advanced processing techniques (e.g., using specialized frameworks or machine learning), portability, and enhanced security (Gamma, Vlissides, Helm, Johnson & Ralph, 1995). However, the development and maintenance of such a system would require significant effort and resources, making it impractical for simpler use cases.

In conclusion, while using SQL queries in Grafana provides an accessible and straightforward way to analyze data, it has limitations, particularly for advanced or large-scale processing. SQL views offer a manageable workaround to reduce query complexity, but a dedicated processing system accessible via an API would be the most optimal solution for complex analyses. In a research context, where simplicity and resource efficiency are often prioritized, the development of such a tool may not be justified and should only be considered for specific, high-complexity requirements. Thus, further investigation regarding the implementation of such solution is beyond the scope of this thesis.

3.6.5 Continuous Extraction

As mentioned, having continuous extraction could be useful in some contexts. This could be achieved without significant changes to the code by, for instance, wrapping the main function in a loop. This would result in a program that automatically restarts data extraction as soon as the previous run is completed. Alternatively, a cron job[14] could be employed to schedule data extraction at regular intervals, such as nightly.

A more advanced approach to continuous extraction could involve the use of GitLab webhooks[15] to capture updates on projects and issues directly. This architecture offers the advantage

[14]A cron job is a Linux command used to schedule tasks for automatic execution at specified times.

[15]A webhook is an event-driven communication mechanism that automatically sends data between applications via HTTP.

of immediate synchronization with data updates, as changes are caught and without delays. Furthermore, it improves the efficiency of data retrieval by identifying precisely which data has been updated, eliminating unnecessary requests. However, this method would require significant modifications to the existing codebase and administrative privileges to configure GitLab webhooks, making the setup more complex.

However, since the study does not require continuous data extraction, the tool has been designed to run on demand, providing new data only when necessary.

CHAPTER 4

RESULTS

4.1 Introduction

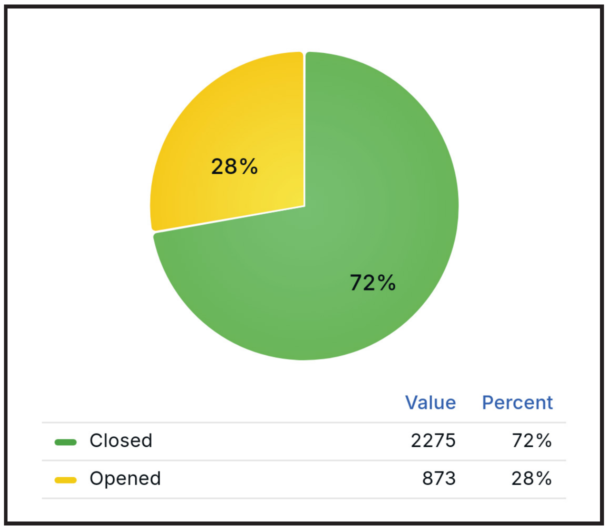
In accordance with the project's overall objective of improving task allocation, the conducted work in the thesis aims to provide an approach to analyze task allocation. Consequently, the results presented in the following sections do not aim to directly improve task allocation, but rather to provide methods, results, and insights as a basis to support the analysis of task allocation, enabling its improvement.

In this chapter, we present the results obtained by applying the methodology outlined in the previous chapter. We report the analyses performed and the corresponding findings, emphasizing their relevance and practical implications for our industrial partner.

To address our RQs, we first need to ensure the reliability of our analyses. This is done in the first Preliminary Question (PQ).

4.2 PQ1: *Are tasks in the TELUS workflow sufficiently estimated and completed to support reliable lead time analysis?*

As shown in Figure 4.2, 72% of tasks have associated estimates, which represents a sufficient proportion for meaningful analysis. To initiate our analysis and gain a better understanding of TELUS's task management process, we first examine the state of all tasks. Each task can be either open (still in progress or pending) or closed (completed or resolved). As shown in Figure 4.1, among the 3,148 tasks analyzed, 2,275 (72%) are closed, while 873 (28%) remain open. Focusing on the closed tasks, we assess the extent to which task sizes have been estimated, since estimation is a prerequisite for categorizing task performance (i.e., success or failure). While 28% of tasks lack estimates and are excluded from the analysis, the remaining 72% provide a sufficiently large dataset for conducting reliable lead time analysis.



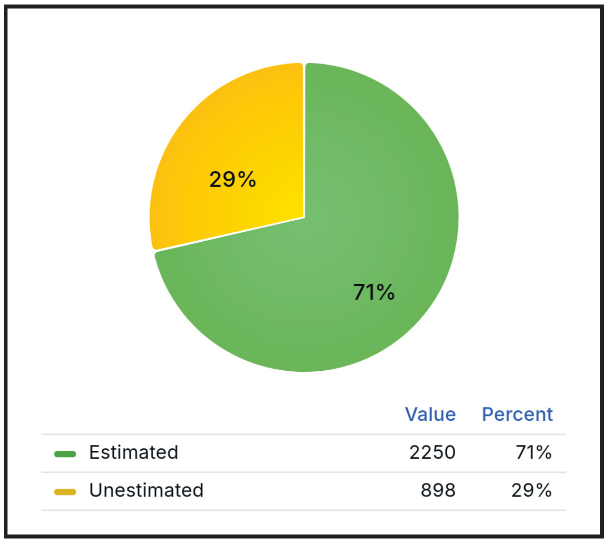
	Value	Percent
🟩 Closed	2275	72%
🟨 Opened	873	28%

Figure 4.1 Proportion of open tasks

The proportion of missing estimates drops significantly to just 12% when considering only tasks that have been moved to the *In progress* column, as shown in Figure 4.3. This observation highlights that estimation practices are more consistently applied once work on a task has officially started. In contrast, tasks that remain in earlier stages of the workflow may not be prioritized for estimation, which is expected in practical settings where some tasks are unprioritized or still under discussion. By refining our dataset to include only tasks that reached the *In progress* stage, we reduce noise introduced by tasks that are not yet actionable. This filtering ensures that the remaining subset reflects active work and provides a more reliable basis for analyzing lead time and task performance.

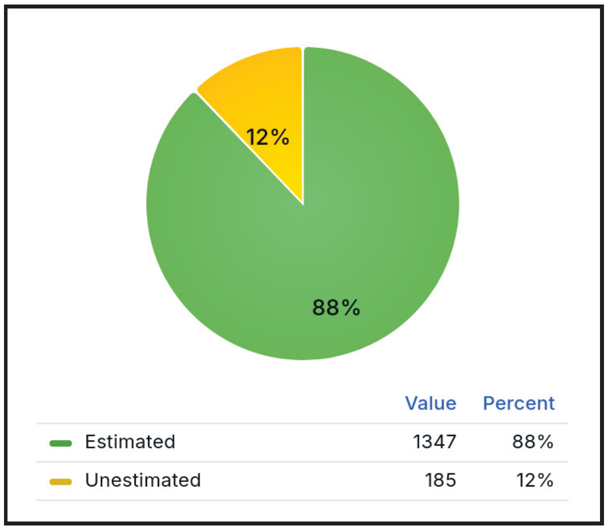
Our findings show that tasks with an estimated size of 1 account for 41.6% of all tasks, which we interpret as a strong preference for small, manageable task sizes within TELUS teams. As shown in Figure 4.4, tasks with size 3 (18.9%) are also relatively frequent, reinforcing the trend toward breaking down work into smaller increments. Tasks without estimation, **represented with size 0 in the thesis**, are frequent as well (28.5%).

A notable observation is the significant presence of tasks with an estimated size of 2 despite the discontinuation of this estimate for new tasks across all teams since 2023 (as detailed in



	Value	Percent
🟩 Estimated	2250	71%
🟨 Unestimated	898	29%

Figure 4.2 Proportion of estimated tasks



	Value	Percent
🟩 Estimated	1347	88%
🟨 Unestimated	185	12%

Figure 4.3 Proportion of estimated tasks among tasks that went to the *In progress* column

Chapter 1). This suggests that the new official guideline might not be applied uniformly across TELUS teams. Leading to potential bias related to this estimate.

Additionally, certain estimated sizes have been excluded from this distribution due to minimal usage and non-compliance with official TELUS guidelines. The excluded sizes include 4, 7, 9, 10, 11, 12, 20, 22, 25, and 45, collectively accounting for 1% of the dataset (35 tasks). By filtering out these rare cases, the analysis remains focused on the primary estimation categories used in practice.

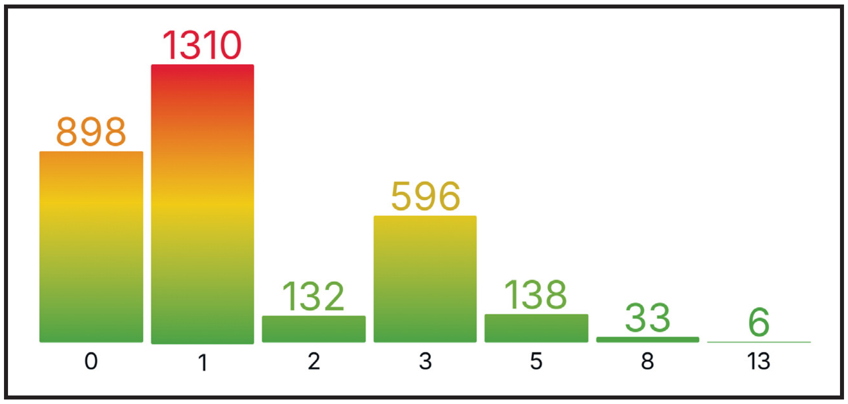


Figure 4.4 Tasks distribution over estimated categories

Non estimated tasks

After brief analysis regarding non estimated tasks and specifically those completed (having a LT), results reveal that 76% of unestimated tasks were created by bots. A proportion much higher than for all complete tasks (12%). Furthermore, 47% of unestimated tasks have a LT shorter than two days (compare to 28% for all tasks). A proportion that rises to 58% when considering a LT shorter than 4 days—the RLT associated to the estimated size 1. These insights indicate greater differences regarding these tasks than just a lack of estimation, making it relevant to not consider them in our study and keep them for a future dedicated analysis.

4.3 RQ1: *How do task failure rates evolve over time and across estimated sizes?*

4.3.1 Motivation

Understanding task failure rates is essential for optimizing workflow efficiency and proactively addressing potential bottlenecks in the TELUS development process. Analyzing failure trends over time and across different estimated sizes provide valuable insights into the underlying factors that impact task execution, enabling more informed decision-making, improved resource allocation, and the refinement of estimation and risk management strategies.

4.3.2 Approach

To investigate task failure rates, we adopt a structured analytical approach that examines both temporal trends and estimated size impact, ensuring a comprehensive understanding of the factors influencing task success. The analysis is conducted in the following steps:

Data Preparation

The total number of failed and successful tasks (1,347) differs from the overall dataset used in the study (3,148). This discrepancy arises due to several factors:

- Some tasks remain open and have not yet reached completion.
- Others lack estimation values, making them ineligible for analysis.
- Certain tasks never progressed to the *In progress* column on the Kanban board, meaning they do not have a defined Lead Time (LT) as per this study's methodology.

These exclusions ensure that the analysis focuses only on tasks for which LT can be meaningfully measured.

Data Classification

Tasks are classified as failed or successful based on their LT relative to the RLT for their respective size categories. A failure is defined as a task with a LT exceeding its associated RLT, with an additional margin (40%) to account for noise and reduce classification bias. The detailed approach is explained in the methodology chapter.

Data Analysis

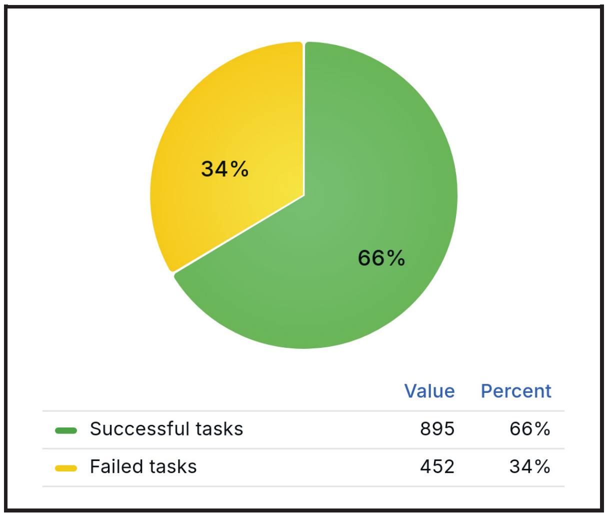
We analyze different aspects to explore the failure of tasks looking at the following:

- **Proportion Analysis by Estimated Size**. We analyze how the failure rate varies across different task sizes to determine whether task size influences failure likelihood. This step helps identify whether smaller or larger tasks are more prone to failure or if failure is independent of size.

- **Temporal Evolution of Failure Rate**. To gain further insights, we analyze the temporal evolution of task failure rates over time. By applying the same criteria (i.e., the same group of tasks and the same success definition), we track the failure rate across each quarter over the past three years. This quarterly division ensures a sufficiently large sample size for reliable statistical analysis.

4.3.3 Results

As presented in Figure 4.5, 34% of the selected tasks are classified as failed, indicating that while 66% of tasks are successful, a substantial proportion does not meet the defined success criteria. This failure rate highlights that over one in three tasks exceed their expected lead time, suggesting the presence of recurring challenges in task execution. These may stem from inaccurate estimations, unforeseen complexities, insufficient coordination, or delays introduced by dependencies on external teams or systems.



	Value	Percent
🟩 Successful tasks	895	66%
🟨 Failed tasks	452	34%

Figure 4.5 Proportion of failed tasks

Our analysis shows that task failure is not directly correlated with task size. As illustrated in Figure 4.6, the proportion of failed tasks remains relatively stable across different estimated sizes, indicating that task size alone does not determine the likelihood of failure. This suggests that other factors—such as task complexity, dependencies, or external constraints—may have a stronger influence on task success.

This finding is particularly relevant in the TELUS context, where network-related tasks, despite appearing small and straightforward in terms of estimated effort, often involve intricate configurations that are prone to errors. Even minor misconfigurations in network settings can lead to failures, requiring rework or additional troubleshooting, which is time consuming. Moreover, dependencies on other system components, coordination with multiple teams, or evolving requirements can further contribute to task failure, irrespective of initial size estimates. These insights highlight the need to consider not just task size but also task nature and complexity when assessing and mitigating failure risks.

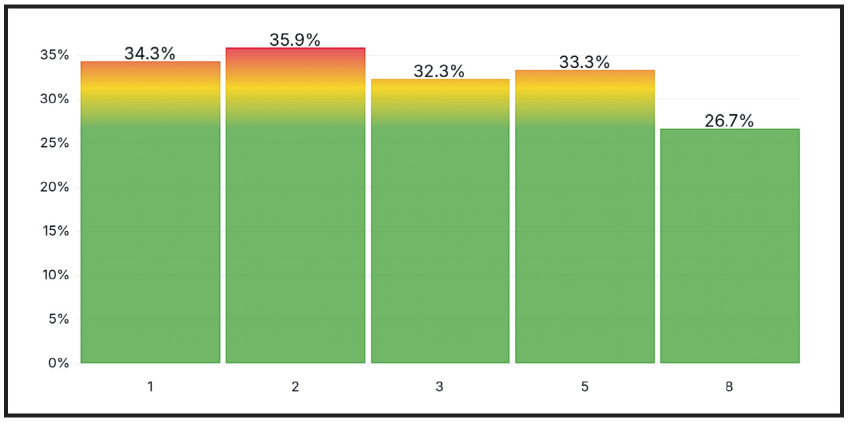


Figure 4.6 Proportion of failed tasks depending on the estimate category

This section concludes with a final analysis on the temporal evolution. After examining the results from the past few months, it is valuable to assess how these results may evolve over time.

By using the same parameters (specifically, the same group of tasks and the same definition of success) and analyzing all tasks from the past three years, it is possible to track the failure rate over the course of each quarter. Quarters are used as the time division to ensure a sufficient number of tasks for each calculation.

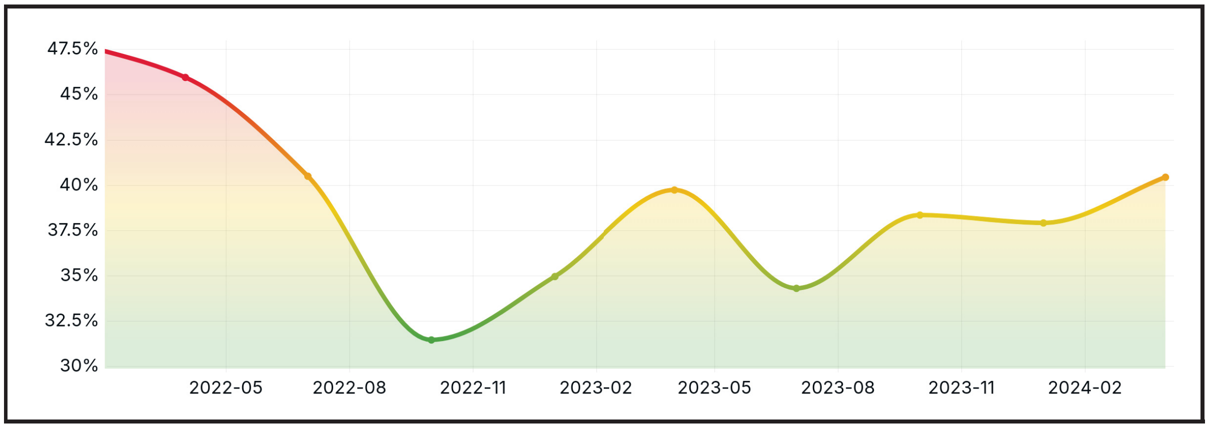


Figure 4.7 Failure rate over time

As illustrated in Figure 4.7, the failure rate is initially high but gradually decreased over time, eventually stabilizing over nearly two years. This early high failure rate may be attributed to users adapting to GitLab and the associated workflow, as the first task with an associated estimate was created at the end of 2021. This suggests that the initial phase of adoption played a role in the observed trends.

Additionally, some extreme variations in the failure rate are visible in the later periods, which could indicate anomalies or significant process changes. Investigating these fluctuations further could be valuable but falls outside the scope of this study and is suggested for future research.

Analysis conducted to address RQ1 gives useful insights to TELUS and enables the use of the success/failure metric as a basis for the next analyses. Leading to a comprehensive and valuable set of KPIs usable for task allocation analysis.

4.4 RQ2: *How does the time tasks spend in waiting columns of a Kanban board affect the overall task failure rate?*

4.4.1 Motivation

While waiting is a normal part of task progression, wait time is considered a waste that needs to be minimized. "While some waiting can be tolerated, damages for a project caused by waiting must be eliminated to have more efficient progress" (Ikonen, Kettunen, Oza & Abrahamsson, 2010, p. 4).

This RQ aims at analyzing the impact of TWC (Time spent in Waiting Columns) on the success or failure of a task. Making this analysis gives valuable information to TELUS project managers and team members about their processes and the impact of TWC in task execution.

4.4.2 Approach

To analyze TWC, the data is first examined to ensure that only unbiased and meaningful tasks are selected. The tasks are then classified based on their TWC, facilitating their analysis.

Data Preparation

All tasks resulting from the success/failure analysis (PQ1) are used except tasks with estimated size 8 and 13. This decision is taken due to the insufficient number of tasks in these categories, preventing the calculation of inaccurate results.

Data Classification

Tasks are classified using the self-defined TWC metric. Furthermore, to characterize that the use of waiting columns is not always excessive, a notion of acceptable TWC has been introduced. TWC is considered acceptable if shorter than a certain proportion of the RLT, otherwise it is considered excessive. This proportion has been set to 50% for this investigation as we assume that having more than half of the estimated time as TWC is not an excepted behavior. However, to allow accurate telemetry, this proportion must be adapted to the industry context. This categorization is summarized at follow:

1. **No TWC**, meaning that no time was spent in waiting columns.
2. **Acceptable TWC**, meaning that the time spent in waiting columns is lower than 50% of the RLT.
3. **Excessive TWC**, meaning that the time spent in waiting columns is higher than 50% of the RLT.

An illustration of this categorization is presented in Figure 4.8

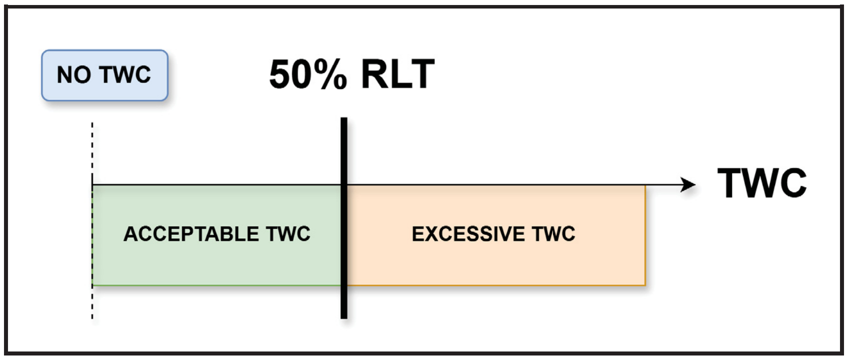


Figure 4.8 TWC Categorization

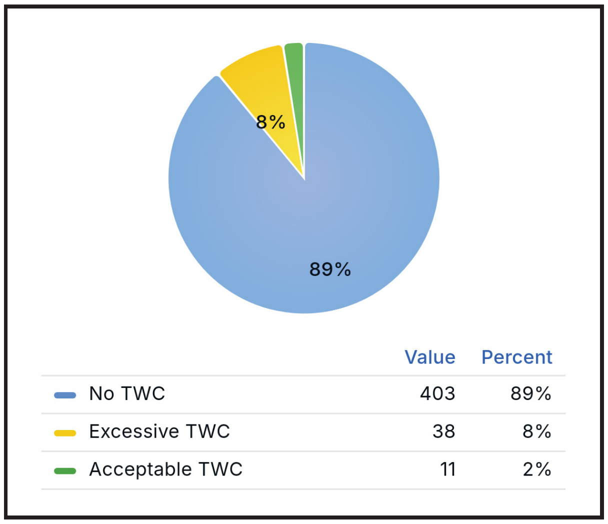
Data Analysis

The analysis of the TWC proceeds in two steps:

- **Considering only failed tasks**. This first analysis focuses on failed tasks to identify the impact of TWC on failure.
- **Considering all tasks**. This second analysis enables the validation of the results from the first analysis and their comparison with results related to all tasks.

4.4.3 Results

As presented in Figure 4.9, only 11% of failed tasks have TWC. Suggesting that the waiting columns are not used in a uniform manner. However, 78% of failed tasks with TWC have an excessive TWC, indicating difficulties in using waiting columns promptly for failed tasks. Among the 403 failed tasks, 38 have excessive TWC, representing 8% of all failed tasks. While this proportion is not negligible and warrants further analysis, it remains a minority. The analysis of the other two categories, tasks with no TWC and those with an acceptable TWC, reveals that 89% of failed tasks did not use the waiting columns at all. Furthermore, only 2% of failed tasks have an acceptable TWC. Notably, for the subset of failed tasks that did use the waiting columns (11%), 78% spent excessive time in these columns while only 22% managed to use the waiting columns without excessive TWC.



	Value	Percent
🟦 No TWC	403	89%
🟨 Excessive TWC	38	8%
🟩 Acceptable TWC	11	2%

Figure 4.9 Proportion of TWC categories in failed tasks

Results in Figure 4.10 reveals a drop in proportion of tasks with excessive TWC among task with TWC as the estimated size increases (from 80% to 0%). Indicating a tendency in TELUS context of struggling more in waiting columns with small tasks. Furthermore, results show increased use of waiting columns as the estimated size increases. Revealing that bigger tasks use more often and longer the waiting columns. This trend begins with the estimate 1 with 80% of tasks with TWC that have excessive TWC and end with the estimated size 8 where all tasks with TWC having an acceptable one. The trend is homogeneous except for the estimate 2, an estimate already flagged as particular since it is used only by some teams in TELUS.

With all tasks considered, results in Figure 4.11 reveals a lower proportion of tasks with TWC (from 11% to 7%), showing that failed tasks are more likely to have TWC. In addition, the proportion of tasks with excessive TWC among task with TWC drop from 78% in failed tasks to 59% in all tasks. Indicating a stronger trend to struggle with TWC for failed tasks. This difference arises from the higher proportion of failed tasks that spend excessive time in the waiting columns (8%) compared to all tasks (3%). These findings support the hypothesis that one of the contributing factors to task failure is excessive TWC.

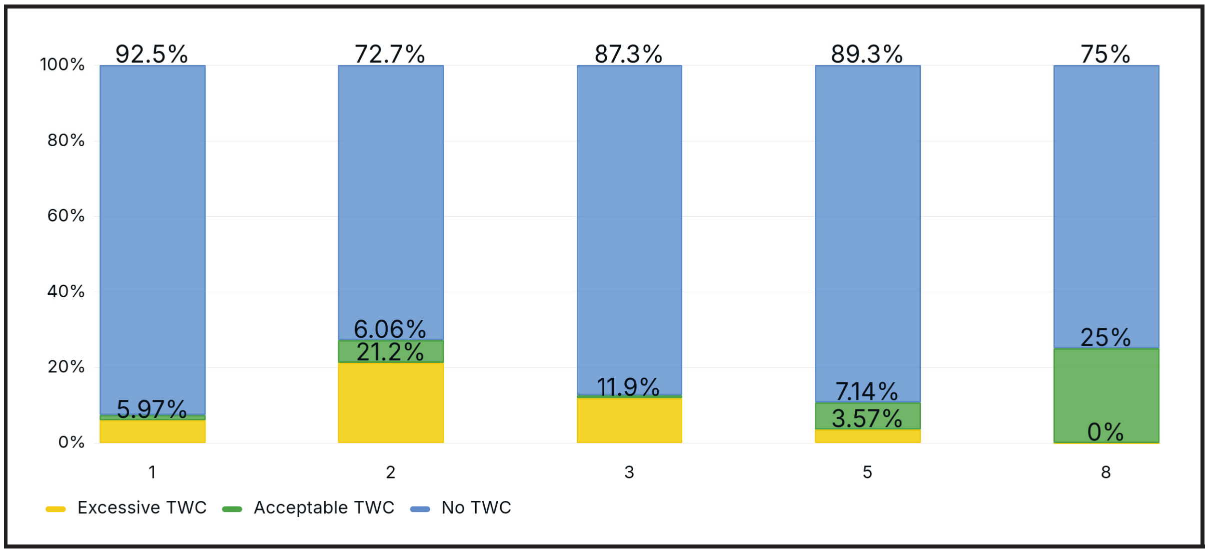
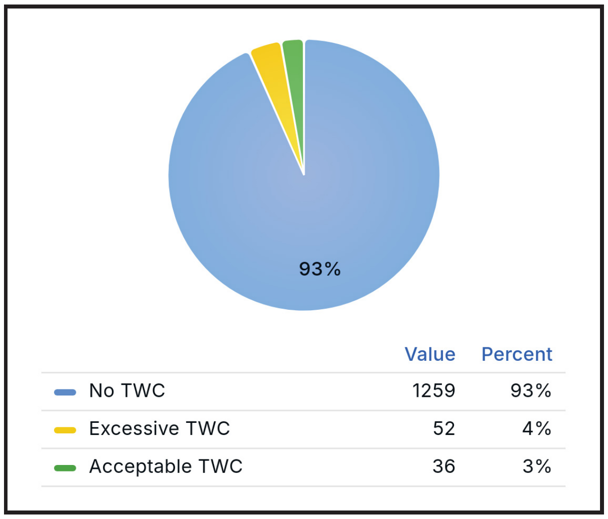


Figure 4.10 Proportion of TWC categories in failed tasks by estimated size



	Value	Percent
No TWC	1259	93%
Excessive TWC	52	4%
Acceptable TWC	36	3%

Figure 4.11 Proportion of TWC categories in all tasks

Results in Figure 4.12 confirm the same trend regarding estimates in all tasks than the first one observed in failed tasks. More precisely, there is an increase in this proportion of tasks with acceptable TWC as estimated size increases, ranging from 0.6% for tasks of estimated size

1 to 13.3% for tasks of estimated size 8. Again estimated size 2 reveals inconsistency in the trend, probably for the same reason as before.

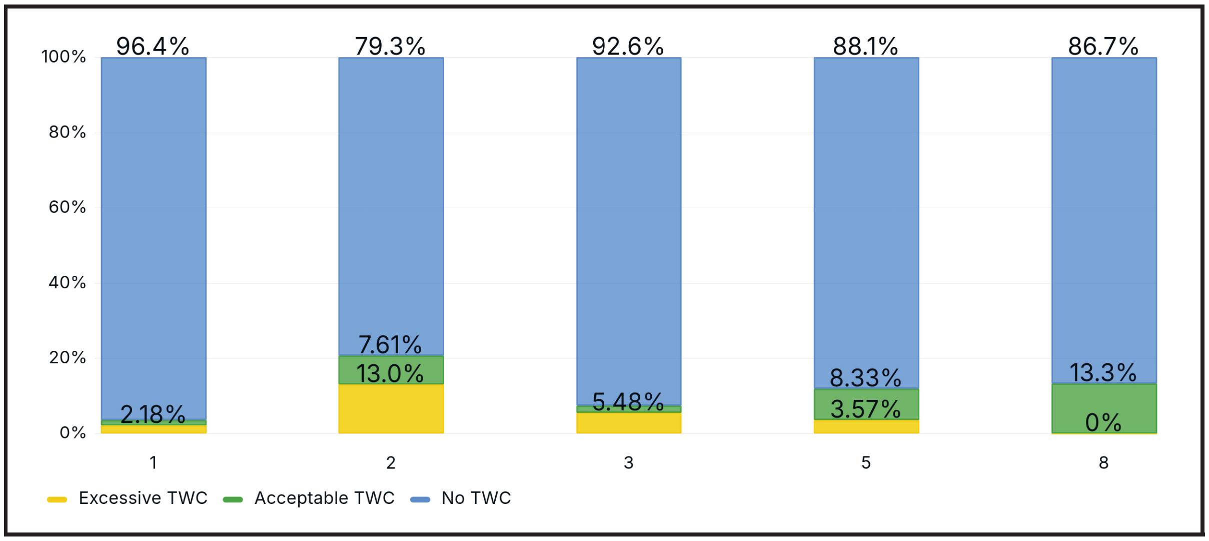
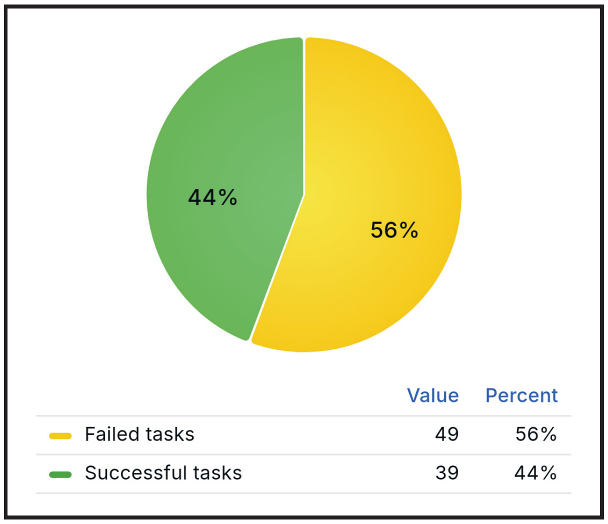


Figure 4.12 Proportion of TWC categories in all tasks by estimated size

Lastly, the proportion of failed tasks among tasks with TWC is calculated in Figure 4.13. Showing that having TWC for a task make its failure probability jump from 34% to 56%. Making the TWC, in TELUS context, as one of the behaviors impacting task failure. As a reminder, tasks with TWC only represent 6% of the tasks. Explaining why its impact is not strongly visible on first analyses.

The analysis conducted in this section allows to conclude regarding the RQ2. First, the use of the waiting columns notably increases the likelihood of failure (from 34% to 56%). More precisely, the proportion of tasks that use the waiting columns is higher in failed tasks than for all tasks (11% instead of 7%). These highlights are more linked to small estimates with a proportion of excessive TWC that decrease with increasing estimated size.
These findings indicate the relevance of further analysis and processes monitoring specific to the waiting columns in TELUS context.



	Value	Percent
🟨 Failed tasks	49	56%
🟩 Successful tasks	39	44%

Figure 4.13 Proportion of failed tasks among tasks with TWC

4.5 RQ3: *What is the impact of multitasking and workload distribution on the failure rate of tasks?*

4.5.1 Motivation

With the overall objective of improving task allocation, analyzing workload distribution and multitasking is essential. Indeed, multitasking can adversely affect productivity (Adler & Benbunan-Fich, 2012), potentially resulting in higher failure rates as people might be already overloaded (Buser & Peter, 2012). Analyzing multitasking provides meaningful insights directly related to task allocation and therefore useful for project managers, enabling the optimization of task planning and execution among TELUS teams.

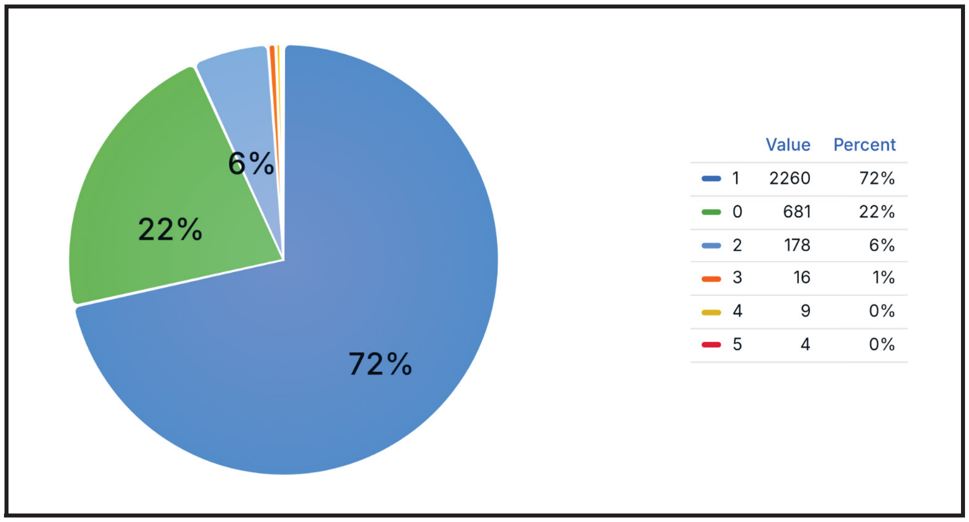
4.5.2 Approach

The defined metrics for workload and multitasking must be calculated accurately, highlighting the importance of careful data preparation. This allows the reliability of the analysis.

Data Preparation

To perform analyses, two self defined metrics are used: workload and Multitasking Value (MV). Their calculations are done in two methods.

1. **Event-based tracking**. This first method stems from a calculation of the workload along the time which serves as the basis and aims to be as precise as possible, with events related to state (tasks closing and reopening), Kanban columns, and assignments (added and removed). Leading to a highly precise but sensible solution. For this method, all tasks are used.
2. **Static information**. The second method aims to simplify the calculation with the use of static information (states, assignment, ...). Leading to easier calculation and a result not far from the reality since task reopening and assignment changes are not often made (less than 20% of tasks). This method uses only tasks with one assignee, leading to the removal of tasks with more than one assignee (7% according to Figure 4.14).



		Value	Percent
🟦	1	2260	72%
🟩	0	681	22%
🟦	2	178	6%
🟧	3	16	1%
🟨	4	9	0%
🟥	5	4	0%

Figure 4.14 Distribution of number of people assigned to tasks

Data Analysis

Analyses are performed with both calculations to verify disparities and ensure coherence.

4.5.3 Results

Figure 4.15 reveals a normal distribution of tasks along MV. High MVs (up to 18) make the calculation questionable. Regarding the overall result, neither the median nor the mean MV are notably different between failed and succeed tasks (Figure 4.16). Looking more precisely, the failure rate does not follow any consistent trend in relation to MV (Figure 4.17). Leading to the incapacity to conclude in a relation between MV and failure rate with these results.

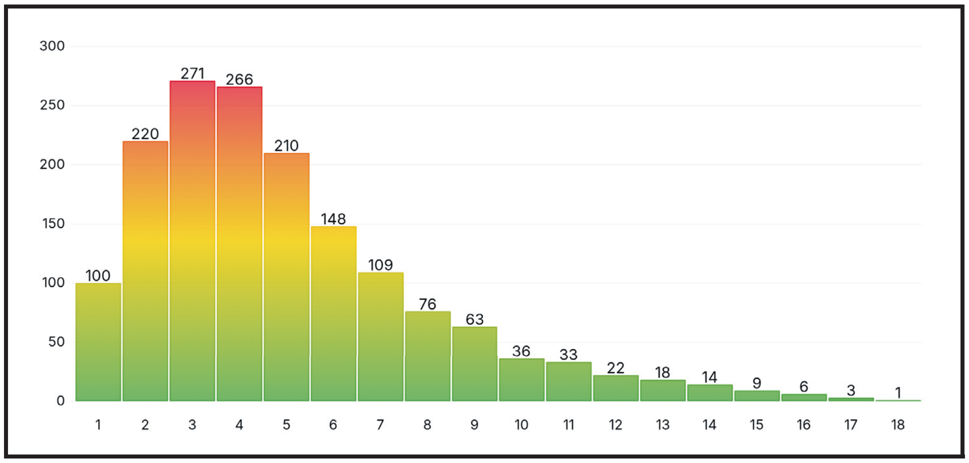


Figure 4.15 Number of tasks by MV - *First method*

Figure 4.15 reveals a wide range of MV, spanning from 1 to 18. To clarify this, a comparison with the second calculation method will help identify whether these results stem from calculation biases or if they genuinely reflect the reality of TELUS teams.

A more granular analysis, splitting the data by MV (Figure 4.17), also fails to produce conclusive results, indicating that further analysis and cross-validation with alternative calculation methods would be necessary to clarify the relationship.

Even if the second calculation method reveals in Figure 4.18 a thinner range of MV (from 18 to 11), other findings (Figures 4.19 and 4.20) remain similar and still don't allow concluding positively. The range of MV is narrower compared to the first method. This reduction suggests that the second method may provide a more accurate and reliable representation of MV. Like

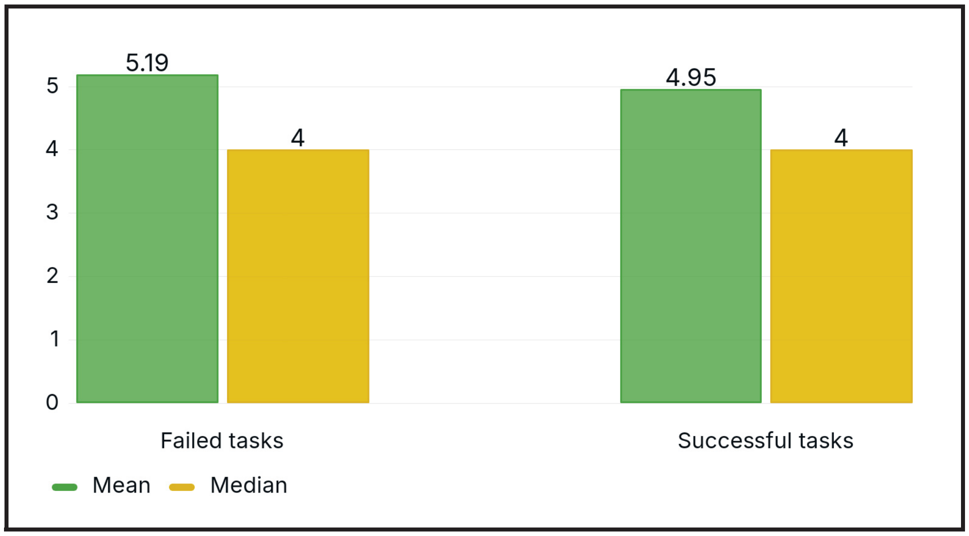


Figure 4.16 MV depending on the success status of tasks - *First method*

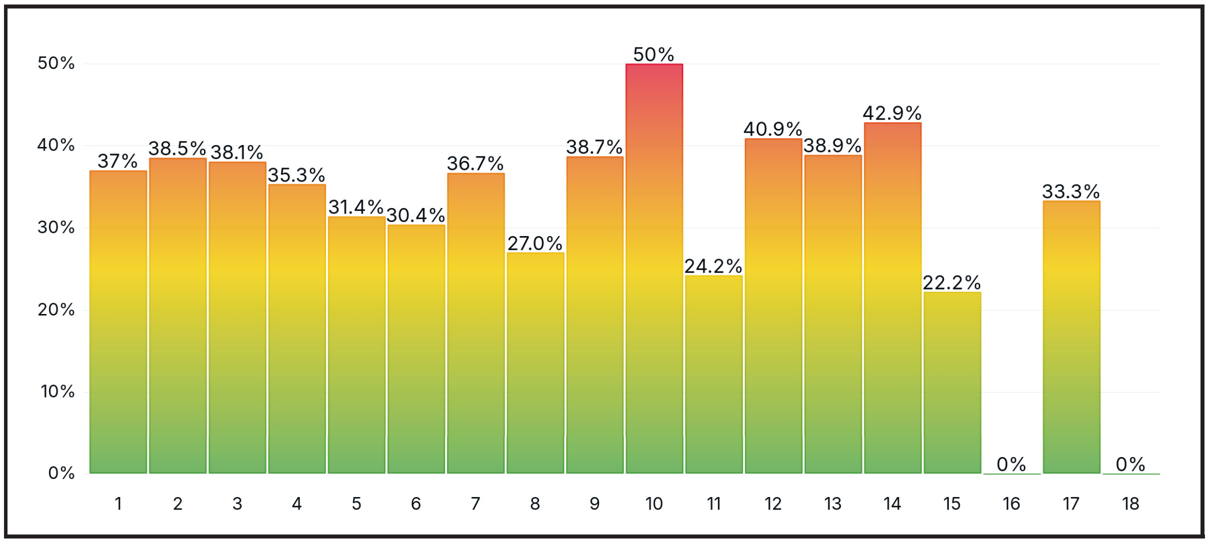


Figure 4.17 Failure rate by MV - *First method*

the first method, the distribution here also follows a trend resembling a normal distribution, reinforcing its consistency.

Figure 4.19 shows no significant difference in MV between failed and successful tasks, whether analyzing the median or the mean. This result does not support the hypothesis.

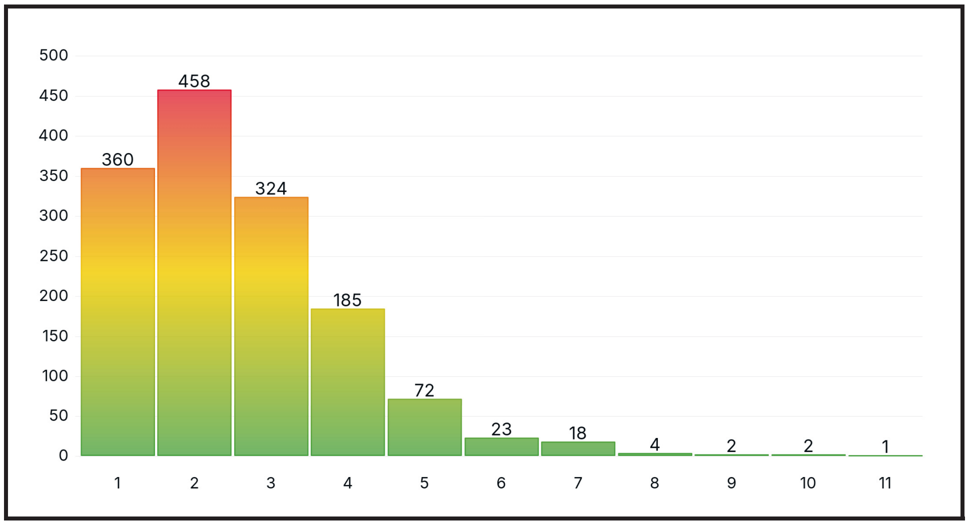


Figure 4.18 Number of tasks by MV - *Second method*

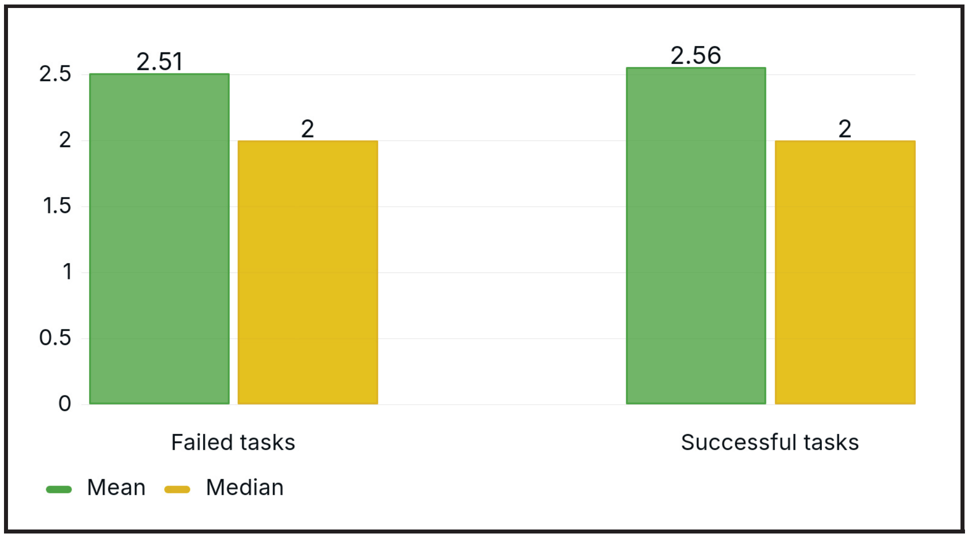


Figure 4.19 MV depending on the success status of tasks - *Second method*

A more detailed breakdown by MV, illustrated in Figure 4.20, provides additional insights but fails to yield conclusive evidence. For example, higher failure rates are observed for MV 6, which could suggest a link between higher MV and task failure. However, the failure rates for other MVs is very homogeneous, contradicting the hypothesis of a correlation between MV and failure rate.

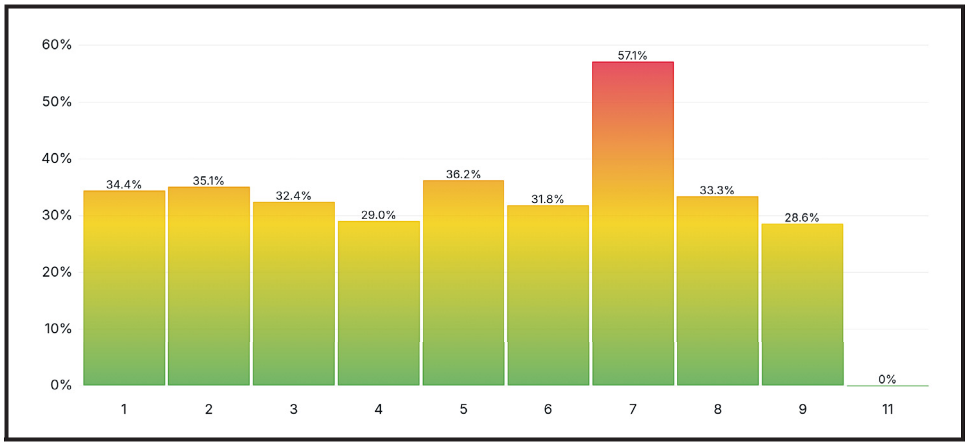


Figure 4.20 Failure rate by MV - *Second method*

In summary, this analysis does not provide sufficient evidence to confirm a correlation between multitasking and task failure. While the hypothesis seemed logical, the data used and the calculations performed do not support this conclusion. Despite initial expectations, the results highlight the complexity of task failure causes and the need for further investigation beyond multitasking alone. This result provides to TELUS and other project managers keys to understand the difficulties linked to multitasking analysis and its potential need to further and more specialized data.

4.6 RQ4: *How does the underestimation of task influence the likelihood of task failure?*

4.6.1 Motivation

Underestimation is logically assumed as one of the reasons for task failure. Indeed, task estimates in software development projects often tend to be underestimated (Usman, Britto, Damm & Börstler, 2018). Therefore, RQ4 aims to investigate the notion of underestimation, enabling the finding of useful insights to improve task planning and estimation. This with the mindset that task estimation is the basis for well-managed task allocation.

4.6.2 Approach

To investigate task underestimation, we first validate the data to ensure its relevance and reliability for this metric. An analysis is then performed to uncover key insights.

Data Preparation

Having the self defined metric of underestimation is great but not sufficient to use it as a single truth. We must ensure that it is feasible. For this purpose, it is important to verify the evolution of the RLTs along estimates to be sure having a growing value as a disparity could falsify calculations.

These RLTs, calculated across all selected tasks (1347), are visualized in Figure 4.21, serving as a basis for validating the chosen definition.

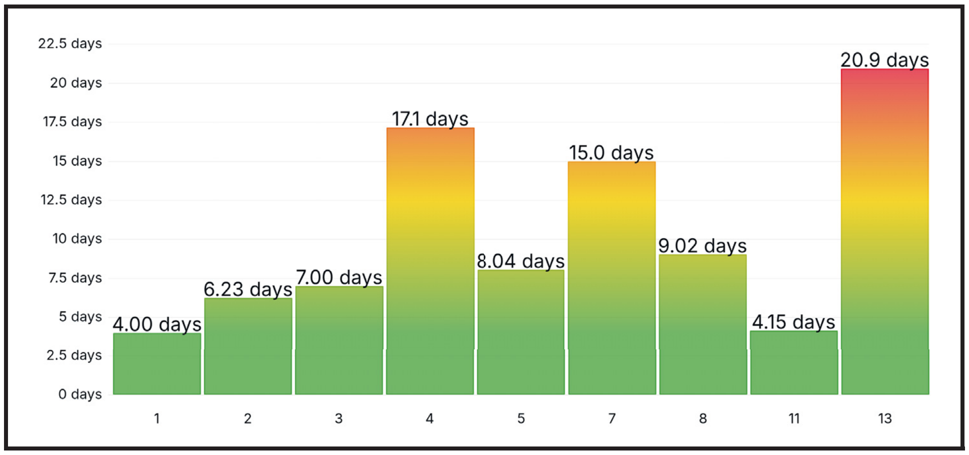


Figure 4.21 RLTs by estimate

The graph reveals that LT generally increases with task size, as expected. However, certain sizes deviate from this trend. Upon closer examination, two key observations emerge:

- The sizes that deviate from the expected pattern are not part of the official TELUS guideline that is based on Fibonacci sequence (Chapter 1 for more details). Consequently, these sizes were not intended for use and likely represent inconsistencies in adherence to the guidelines.
- The number of tasks used in the calculations for these outside of guideline sizes is significantly low. The Figure 4.22 highlights that sizes 4, 7, and 11 are represented by very few tasks.

This limited data can result in unreliable median LT calculations, further contributing to the observed irregularities.

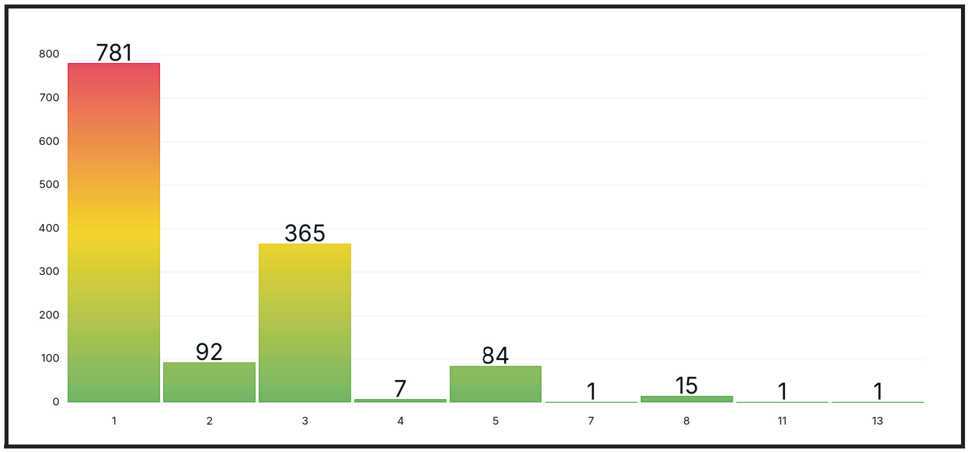


Figure 4.22 Tasks taken in calculation of the RLTs

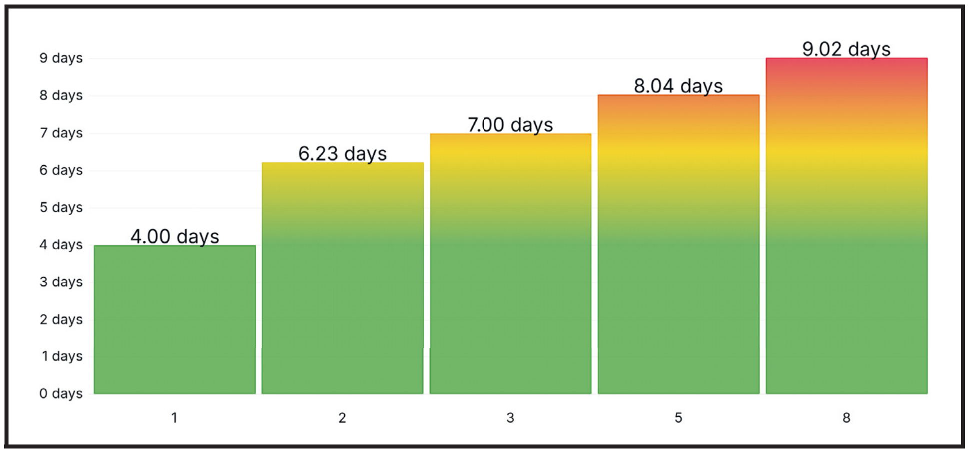


Figure 4.23 RLTs by estimate with only selected estimate

To ensure the reliability of the analysis, only tasks with estimated sizes of 1, 2, 3, 5, and 8 are included in the subsequent steps. These sizes provide two key advantages: they ensure a sufficiently large number of tasks for the calculations, resulting in meaningful and unbiased results, and they align exclusively with the TELUS guidelines for task estimation. The updated version of the median LTs graph, now limited to these selected sizes, is presented in Figure 4.23. The RLT now increases consistently with the estimated size.

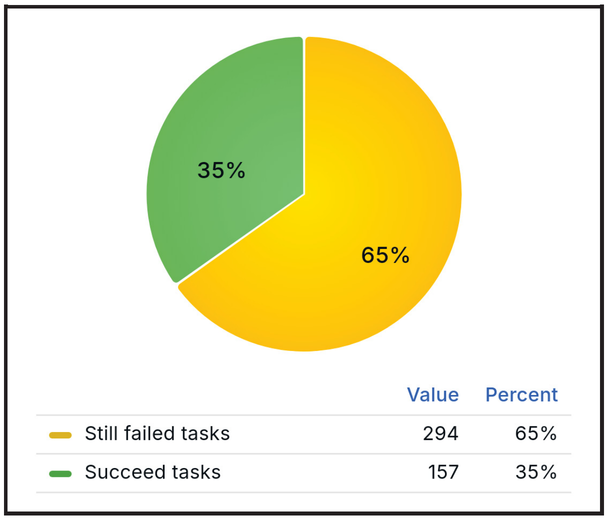
Data Analysis

To address RQ4, the relation between underestimation and failure rate is analyzed, both in multiple dimensions:

- In a global way and in an estimated size specific analysis.
- With only failed tasks and with a general analysis on all tasks. Indeed, even if only failed tasks can be underestimated according to our defined metric, some behaviors can be overrepresented in failed tasks, making the analysis of all tasks relevant for validation purposes.

4.6.3 Results

Global analysis, presented in Figure 4.24, indicates that 35% of failed tasks are underestimated (this means that the task would have been a success if it had been estimated one size larger). This finding supports underestimation as a possible behavior leading to task failure. Making it important to monitor and analyze in TELUS context.



	Value	Percent
🟨 Still failed tasks	294	65%
🟩 Succeed tasks	157	35%

Figure 4.24 Proportion of underestimated tasks in failed tasks

The estimate specific analysis highlights a global decrease in the proportion of underestimated tasks when the estimated size increases (Figures 4.25 and 4.26). Leading to the conclusion that tasks with small estimates are the most prone to underestimation. Suggesting that

small estimates might sometimes be applied too readily, even when a higher size would more accurately reflect the task's complexity. This is a useful information for project managers at TELUS who aim to improve task estimation, leading to better allocation. An exception arises for estimated size 2 that might be due to bias with its use, as already discussed.

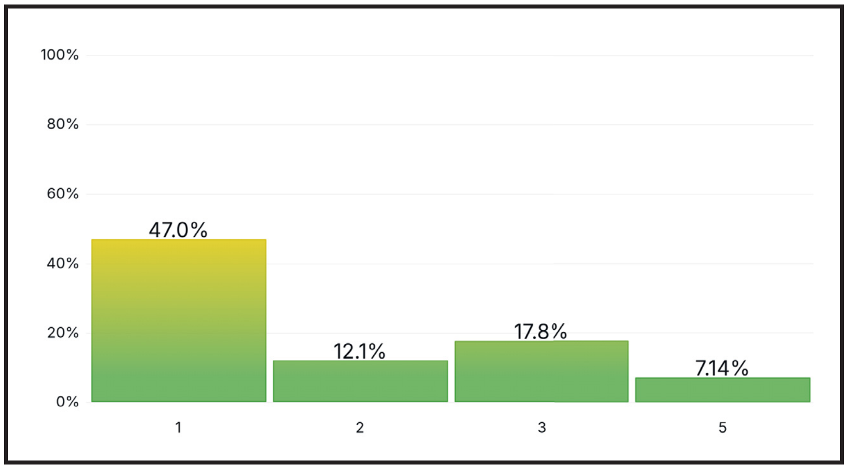


Figure 4.25 Proportion of underestimated tasks in failed tasks grouped by estimate

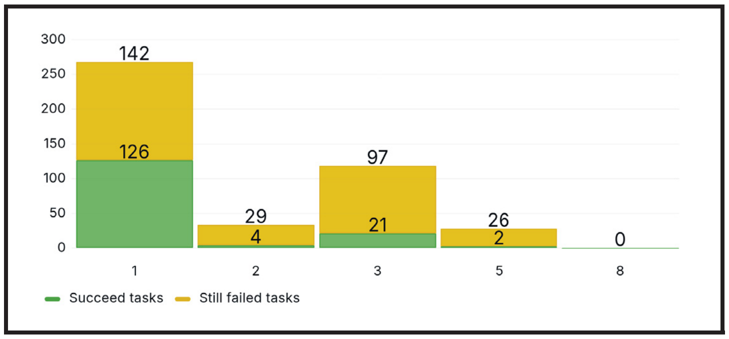


Figure 4.26 Number of underestimated tasks in failed tasks grouped by estimate

Figure 4.27 confirms the previous findings, adding a validation on all tasks.

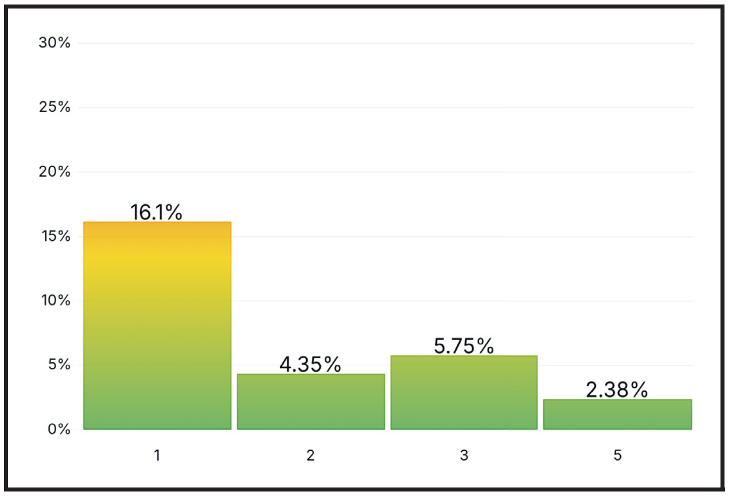


Figure 4.27 Proportion of underestimated tasks in all tasks grouped by estimate

In summary, underestimation is a significant factor among failed tasks, enabling to answer the RQ4 in a positive way. More precisely, its prevalence varies by estimated size and tasks estimated as size 1 are particularly prone to underestimation. This tendency may be due to a tendency to default to this size too frequently during estimation.

This work highlights the importance of estimation as a fundamental prerequisite for task allocation. Making task estimation one of the important factors to monitor and analyze in order to improve task allocation.

CHAPTER 5

IMPLEMENTATION OF THE FRAMEWORK

5.1 Introduction

Following the development of a data extraction tool for internal purposes of this research, we extended its capabilities into a fully operational, ready-to-use framework. This effort led to the creation of an open-source project, which is publicly available on GitLab at the following link: *https://gitlab.com/ets-devops/kanban-data-retrieval-tool*. The objective of this framework is to provide a readily deployable solution for anyone seeking to analyze projects, Kanban boards, and issues in GitLab.

The framework consists of two parts. The main one is the data retrieval tool, which enables the extraction and processing of data from GitLab and can function independently. The second part encompasses all supplementary components required to ensure usability, including a local database and a local Grafana instance with a pre-configured dashboard, offering a seamless and comprehensive user experience.

5.2 Main Part: The Data Retrieval Tool

The core component of the framework is the data retrieval tool, which connects to the GitLab API to extract relevant data, process it, and store it in a designated database. Its underlying functionality is extensively detailed in Chapter 3. Therefore, the focus here is not on exploring its inner workings in depth but on providing practical guidance on its usage and contribution.

5.2.1 Usage

Configuration

To use the data retrieval tool, it must be properly configured. This is achieved through two distinct methods:

- **Configuration file**. The data retrieval tool is highly configurable, allowing users to specify whether data should be stored in JSON files or a database. Additionally, it enables the selection of specific elements to retrieve and the definition of the retrieval scope. Three predefined scopes are available: *all*, *groups*, and *projects*. If the chosen scope is either *groups* or *projects*, a corresponding list must be provided to specify which entities should be included in the data extraction. This flexibility is particularly useful for avoiding the retrieval of unnecessary data from the entire GitLab instance. The configuration settings are stored in a YAML file, with a default `config.yaml` file available in the project's root directory. Users can modify this file according to their specific requirements.
- **Environment variables**. Unlike the configuration file, which primarily manages non-sensitive and frequently changing settings, certain critical parameters—such as the target GitLab instance and the destination database—contain sensitive information and typically remain constant. These parameters are managed through environment variables. While a common approach is to define them in a dedicated environment file, users are free to choose the mechanism that best suits their setup.

Execution

The data retrieval tool can be executed through three different methods, each suited to specific use cases:

1. **Using a package manager**. For users who prefer not to use Docker, the tool can be run directly by cloning the repository onto a local machine. To do so, it is necessary to install the package manager *Poetry*, which manages dependencies and facilitates execution. This approach is particularly useful for those intending to modify the codebase. Before running the tool, ensure that the `.env` and `config.yaml` files are present in the project's root directory.
2. **Using the Docker image**. Alternatively, the tool can be executed via its pre-built Docker image, eliminating the need to clone the repository.

   The official image, `damiendesvent/kanban-data-retrieval-tool`, can be pulled from

Docker Hub[1] and run directly. However, users must provide the appropriate configuration file and environment variables. This method is ideal for seamless execution when database storage or a Grafana instance is not required.

3. **Using Docker Compose**. To get the full capabilities of the framework, the recommended approach is to use Docker Compose. This method automates the deployment of all required components, including the data retrieval tool, database, and visualization tools. Users simply need to execute the provided `docker-compose.yml` file, ensuring a streamlined and comprehensive setup.

Detailed instructions for each execution method can be found in the project's `README` file.

5.2.2 Contribution

As an open-source project, contributions are both possible and strongly encouraged. To ensure consistency and maintainability, contributors should adhere to key principles, which are outlined below.

Codebase Organization

The codebase is structured into modules grouped in the directory based on their functionality. Maintaining this organization is essential to preserving modularity and minimizing dependencies between modules. Contributors should ensure that new additions align with this structure to facilitate maintainability and scalability.

Package Management

The project uses *Poetry* as its package manager, enabling seamless integration with environments and dependency management. All operations related to dependency installation, environment setup, and project execution, including Python and Git commands, should be performed using Poetry.

[1]Docker Hub is a widely used, publicly accessible container registry.

Testing

To prevent regressions and ensure reliability, the project includes extensive unit tests along with a selection of end-to-end tests. These tests must evolve in parallel with the tool's development to maintain adequate coverage and prevent errors. Contributors should update or create tests as needed when modifying the codebase.

Pre-commit Hooks

To enforce code quality, contributors must use pre-commit hooks. This tool automatically performs various checks before each commit, including linting, formatting, and running tests. Detailed installation and usage instructions for the pre-commit hooks are available in the project `README`.

Branching Model

The project follows the *Git Flow* branching model, where each modification must be developed in a dedicated branch before being merged into the *develop* branch via a merge request. Once changes are reviewed, tested, and approved, they can be merged into the *main* branch. This workflow ensures high code quality and facilitates collaboration. Additionally, this process is particularly critical, as the project's Docker image is automatically generated from the *main* branch.

Issue Tracking

To propose a feature, request a modification, or report a bug, contributors should open an issue with a precise and detailed description. Reported issues will be addressed as soon as possible.

5.3 The Complete Framework

Beyond merely providing a tool for data extraction, this framework offers a fully integrated, ready-to-use solution that includes all necessary components for analyzing and monitoring processes effectively.

5.3.1 Overview

The framework consists of the previously discussed data retrieval tool, along with a PostgreSQL database and a Grafana instance, both preconfigured for local use. The overall architecture is illustrated in Figure 5.1.

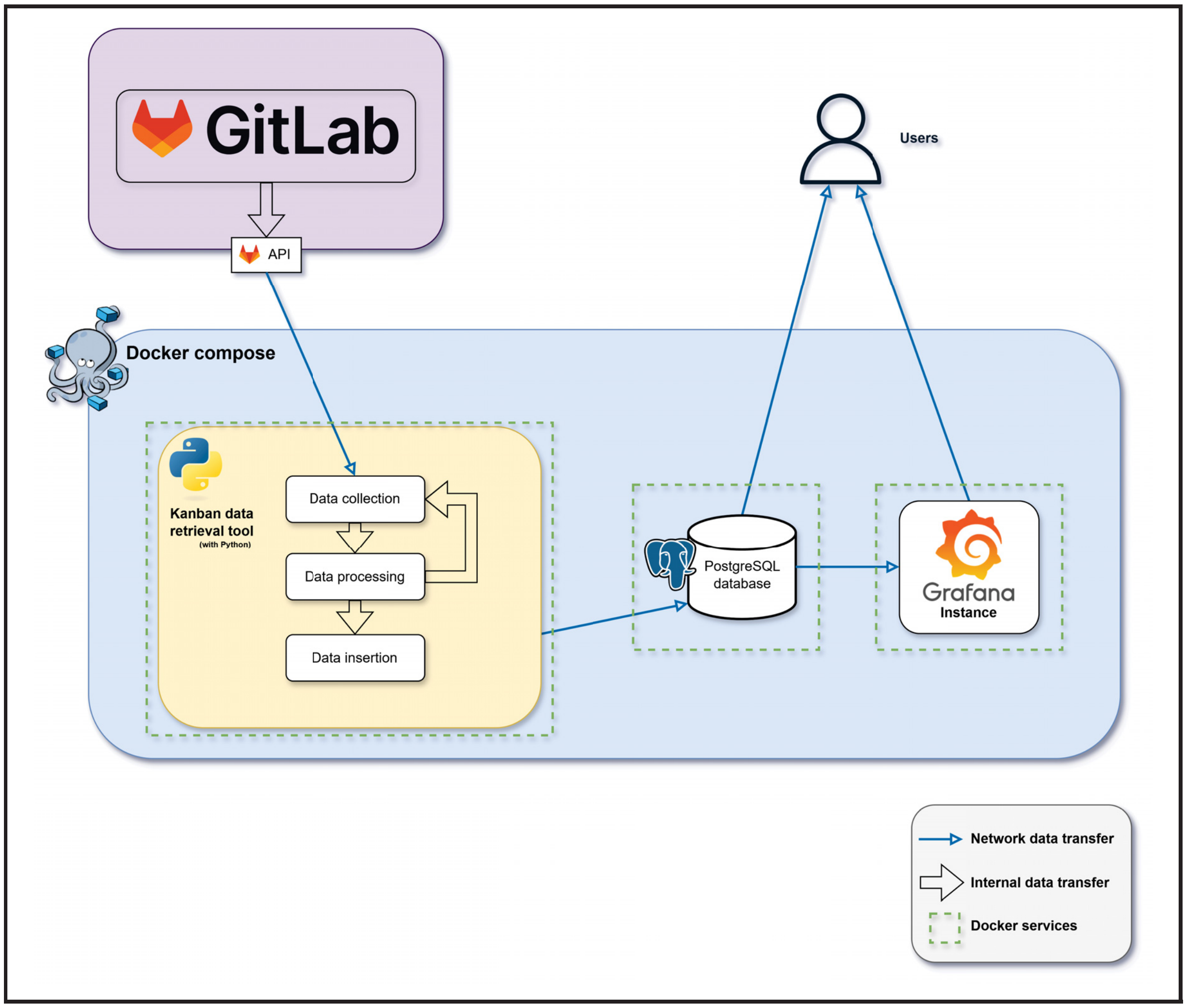


Figure 5.1 Framework architecture

A key strength of this framework is its ease of use. Users do not need to configure individual components manually—once deployed, it requires only a few minutes before insightful analyses and visualizations of Kanban processes become available.

5.3.2 Components

The following sections provide a detailed overview of the main components that support the data retrieval tool.

Database

The database service uses the official PostgreSQL image to create a database instance. Since all tables are managed directly by the data retrieval tool, there is no need for manual table creation during database initialization. However, some setup steps remain necessary, including configuring the appropriate network settings and user roles.

Several initialization scripts are executed upon database set up to ensure optimal functionality. These include the creation of a dedicated user with restricted read-only access for Grafana, as well as the setup of useful database views.

Dashboard

A crucial aspect of the framework is its ability to visualize data and metrics. This is achieved through a Grafana service, which deploys an instance using the official Grafana image with predefined configurations.

The service comes with a preloaded data source and a default dashboard, allowing users to immediately visualize key metrics without additional configuration. If an alternative database is used instead of the provided PostgreSQL instance, the data source settings can be modified accordingly. Additionally, the dashboard itself can be customized to meet specific needs.

The current default dashboard includes various insights related to projects and issues, along with more detailed analyses focusing on all the metrics defined in this thesis. Therefore, this dashboard enables project managers to conduct analyses similar to those presented in this thesis in minutes.

Moreover, the framework includes an optional fourth service in the Docker Compose manifest: a Grafana image renderer. While not required, this service can be useful for generating dashboard snapshots and exporting visualizations.

Snapshots of the dashboard provided by the framework are illustrated in the figures below.

Figure 5.2 shows different diagrams used to analyze issue weights. The top left bar-chart indicates the number of issues by weight. Once again, note that the weight 0 refers to unestimated tasks. The top right corner gives statistics related to the weight such as the mean or the median for example. These information are useful to better understand the use of the weights in the considered dataset. Below, multiple pie-charts represent the proportion of estimated/unestimated tasks. Each one with a different subset of data, beginning with all issues, and then continuing with only issues that came to the Kanban board and so on.

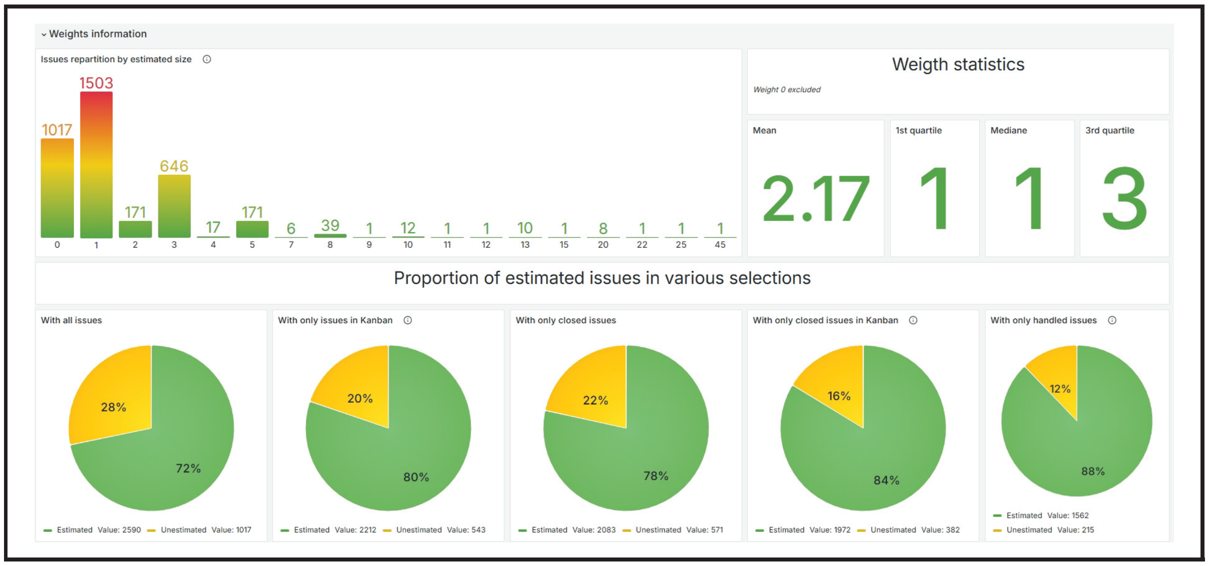


Figure 5.2 Snapshot with information related to the weights

Figure 5.3 is about the time that issues have spent in different columns. For example, the top left bar-chart diagram indicates that, based on the median percentage of time spent in a column, failed issues of size 1 spent 24% of their time in the *To do* column, 50% in the On-Hold column, 12% in the *blocked* column, and so on. The bar-chart diagram below provide the same information for success issues. This type of diagram allows visually comparing the time spent in each column by success and failed issues. The diagram at the bottom provide a general view of the time spent in each column when considering all issues.

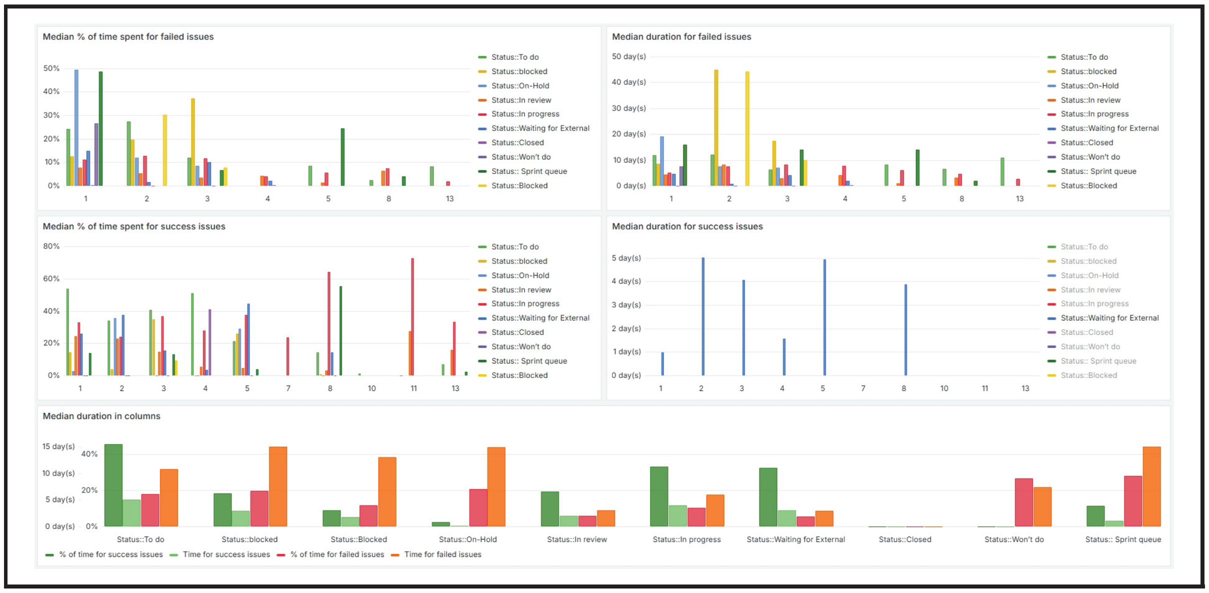

Median % of time spent for failed issues
Median duration for failed issues
Median % of time spent for success issues
Median duration for success issues
Median duration in columns
Status::To do
Status::blocked
Status::On-Hold
Status::In review
Status::In progress
Status::Waiting for External
Status::Closed
Status::Won't do
Status:: Sprint queue
Status::Blocked
% of time for success issues
Time for success issues
% of time for failed issues
Time for failed issues

Figure 5.3 Snapshot with information related to the columns of the Kanban board

Figure 5.4 shows information about the MV and its relation with the failure rate. With information on the failure rate regarding the MV on the left, the median and mean MV for success/failed tasks on the middle, and the distribution of issues among MV on the right; this visualization provides insights on the relation between multitasking and failure.

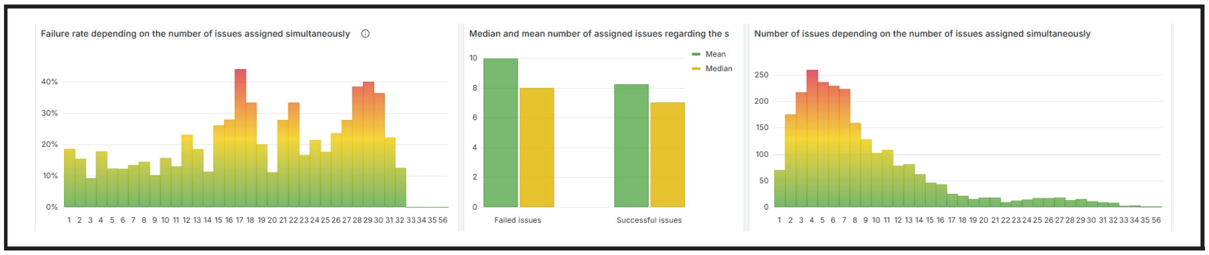
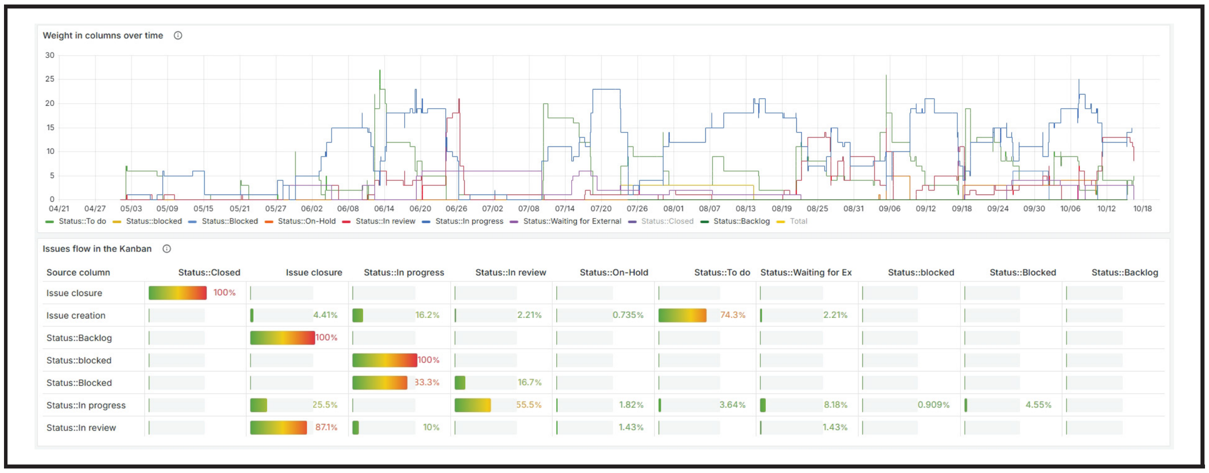

Failure rate depending on the number of issues assigned simultaneously
Median and mean number of assigned issues regarding the s
Mean
Median
Failed issues
Successful issues
Number of issues depending on the number of issues assigned simultaneously

Figure 5.4 Snapshot with information related to the multitasking value

Lastly, Figure 5.5 gives information about the use of Kanban board columns. The top chart gives the weight in columns over time, in other words the sum of all issue's weights in each column. Allowing to visualize evolution, track work progress, and detect potential accumulations or bottlenecks. The bottom chart gives the flow of all issues between columns, indicating, for a given source column, all moves made and their proportion. For example, the second line indicates that in all newly opened issues (*Issue creation* and *Issue closure* are not real columns in the Kanban boards but represent issue's state) 4.4% were directly closed, 10.2% were put in the *In progress* column, 74.3% in the *To do* column, etc. This allow people to better understand how the Kanban board is used.



Issues flow in the Kanban

Source column	Status::Closed	Issue closure	Status::In progress	Status::In review	Status::On-Hold	Status::To do	Status::Waiting for Ex	Status::blocked	Status::Blocked	Status::Backlog
Issue closure	100%									
Issue creation		4.41%	16.2%	2.21%	0.735%	74.3%	2.21%			
Status::Backlog		100%								
Status::blocked			100%							
Status::Blocked			33.3%	16.7%						
Status::In progress		25.5%		55.5%	1.82%	3.64%	8.18%	0.909%	4.55%	
Status::In review		87.1%	10%		1.43%		1.43%			

Figure 5.5 Snapshot with information about the weight in columns over time and the flow of issues

5.3.3 Deployment

To deploy the framework, users simply need to execute Docker Compose with the provided manifest. The project's codebase is not required, as the framework uses the official Docker image available on Docker Hub.

However, certain configuration files are necessary for execution. These include the configuration file and environment variables file required by the data retrieval tool, as well as the *Grafana* and *database* directories, which contain essential setup files.

One again, further details can be found in the project's `README` file.

5.4 Generalization

While the framework is designed to work seamlessly as a local solution, it is also fully compatible with an external database and an existing Grafana instance. By simply adjusting the configuration, it can be easily integrated into an existing infrastructure without requiring any modifications to the code.

Moreover, while the data retrieval tool is primarily built for historical data collection, it includes a filtering mechanism that allows it to retrieve only newly available data. This feature enables a form of near continuous analysis when configured to run automatically at a chosen frequency.

5.5 Threats to Validity

As with any research, it is essential to address key methodological considerations to ensure the relevance, validity, ethical integrity, and confidentiality of the study. In the following sections, the major aspects of these concerns will be thoroughly discussed, providing a clear understanding of the methods underpinning this research.

5.5.1 Data Privacy

Due to the nature of this research, a significant amount of data has been collected from TELUS GitLab, encompassing project, issue, and user-related information. Although this data is not classified as highly sensitive, ensuring its integrity and privacy remains a priority.

To maintain confidentiality, the data is stored exclusively for analysis purposes and solely during the research period. It is kept in a single, secure location—on an encrypted local disk. Furthermore, no raw data will be made publicly available; only high-level calculations derived from this data are be presented in the thesis.

Upon completion of this research, the data will be transferred to a secure storage system managed by the DevOps IRC and permanently deleted from its local storage.

5.5.2 Data Anonymization

A critical aspect of data privacy pertains to user information, which involves both privacy and ethical considerations. To address this concern, raw user data is used solely for data processing, with only a unique user identifier retained for analysis purposes. This identifier is used to examine the extent to which actions and tasks are associated with the same or different users. However, to ensure privacy, this identifier does not appear in the research results.

5.5.3 Generalizability

While the partnership with TELUS has undoubtedly provided significant advantages, facilitating data collection and enabling communication with employees, it is important to consider the extent to which this research is not solely tailored to TELUS. As previously mentioned, the data used in this study is derived exclusively from TELUS and will not be made publicly available, which may initially suggest that the results are specific to TELUS context. However, the research is grounded in general hypotheses designed to identify overarching behaviors and patterns that influence task allocation. These findings can therefore be adapted or further explored in a broader context.

Moreover, beyond presenting numerical results, this thesis offers a comprehensive set of guidelines and an implemented methodology for analyzing task allocation. This open-source resource is publicly available and provides a valuable tool for any organization using GitLab as task management software, making it relevant beyond the TELUS context.

CONCLUSION AND RECOMMENDATIONS

The problem addresses by this thesis is the lack of systematic support for improving task allocation in DevOps processes. The research was conducted in collaboration with TELUS, a leading telecommunications provider in Canada, and aims at developing a structured approach to analyze task allocation and provide actionable recommendations for optimizing DevOps processes. By investigating key factors that influence task allocation and analyzing existing workflows, this study provides both practical insights and a solid framework for analyzing task allocation in a DevOps environment. The collaboration with TELUS was instrumental in grounding the research in real-world data, allowing the exploration of task allocation dynamics in an industry context.

Through the analysis of Kanban workflows, based on 3148 issues from 137 projects in TELUS, and the development of a task categorization framework, this research identifies critical patterns and behaviors that influence task allocation decisions. Results reveal that 34% of tasks failed to meet their estimates, with a failure rate remaining consistent across estimated size and over time. The findings also indicate that the failure rate of tasks that spend time in waiting columns (*Blocked*, *On Hold*, and *Waiting for External*) increases from 34% to 56%, a trend more prevalent in tasks with small estimates. Additionally, multitasking has been analyzed as a potential reason for task failure, yet results did not allow us to reach any conclusion from this investigation. Finally, results show that over one third of failed tasks (35%) are underestimated, with this tendency also being more pronounced in smaller estimates. These findings provide actionable recommendations useful for project managers to improve task allocation processes.

The contributions of this thesis are comprehensive, with a focus on the development of an open-source framework for task allocation analysis. This framework is designed not only to address TELUS's specific needs but also to offer broad applicability for organizations using GitLab as their task management tool, thus extending its relevance beyond the TELUS context.

By providing a robust framework for monitoring task allocation, the research identifies key inefficiencies and gaps in existing DevOps practices, offering actionable insights for their resolution. Furthermore, the framework's ability to track task allocation metrics forms the basis for future optimization efforts, allowing for a more data-driven approach to task allocation. The findings from this study also lay the groundwork for the development of automated systems that can detect and mitigate behaviors impeding efficient task allocation. These systems hold promise for improving productivity and fostering collaboration within DevOps teams, both at TELUS and in other industry contexts.

In conclusion, although this thesis does not propose a groundbreaking solution, it makes a meaningful contribution to the field by providing actionable frameworks and tools that can be readily applied in industry settings. The findings of this research are not only relevant to TELUS but also have broader implications for organizations seeking to optimize task allocation in DevOps environments. As the demand for more efficient and scalable task management practices continues to grow, the approaches and recommendations presented in this study will play a key role in shaping the future of task allocation in software development.

Future research in this area should explore the implementation of automated solutions for continuous task allocation and evaluation, particularly in the context of large-scale organizations. Additionally, further studies could investigate the integration of advanced machine learning models to enhance task estimation and allocation decisions, potentially leading to even greater improvements in task allocation efficiency and overall team performance.

DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES IN THE WRITING PROCESS

During the preparation of this work the authors used ChatGPT (OpenAI, 2025) in order to improve writing quality, translate, and rephrase. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

LIST OF REFERENCES

Adler, R. F. & Benbunan-Fich, R. (2012). Juggling on a high wire: Multitasking effects on performance. *International Journal of Human-Computer Studies*, 70(2), 156–168. doi: 10.1016/j.ijhcs.2011.10.003.

Al-Fraihat, D., Sharrab, Y., Al-Ghuwairi, A.-R., Alzabut, H., Beshara, M. & Algarni, A. (2024). Utilizing machine learning algorithms for task allocation in distributed agile software development. *Heliyon*, 10(21), e39926. doi: 10.1016/j.heliyon.2024.e39926.

Albrecht, A. & Gaffney, J. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, SE-9(6), 639–648. doi: 10.1109/TSE.1983.235271. Conference Name: IEEE Transactions on Software Engineering.

Amrit, C. (2005). Coordination in software development: the problem of task allocation. *Proceedings of the 2005 workshop on Human and social factors of software engineering*, (HSSE '05), 1–7. doi: 10.1145/1083106.1083107.

Arifin, H. H., Daengdej, J. & Khanh, N. T. (2017). An Empirical Study of Effort-Size and Effort-Time in Expert-Based Estimations. *2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pp. 35–40. doi: 10.1109/IWESEP.2017.21.

Aslam, W. & Ijaz, F. (2018). A Quantitative Framework for Task Allocation in Distributed Agile Software Development. *IEEE Access*, 6, 15380–15390. doi: 10.1109/ACCESS.2018.2803685. Conference Name: IEEE Access.

Birrell, A. D. (1989). *An introduction to programming with threads*. Digital Systems Research Center.

Brice, J. (2021). Ali Tizghadam | Mobile Magazine. Retrieved on 2025-01-24 from: https://mobile-magazine.com/interviews/ali-tizghadam-principal-technology-architect-telus-0.

Buser, T. & Peter, N. (2012). Multitasking. *Experimental Economics*, 15(4), 641–655. doi: 10.1007/s10683-012-9318-8.

Coelho, E. & Basu, A. (2012). Effort estimation in agile software development using story points. *International Journal of Applied Information Systems (IJAIS)*, 3(7), 7–10. Publisher: Citeseer.

Colley, D., Stanier, C. & Asaduzzaman, M. (2018). The Impact of Object-Relational Mapping Frameworks on Relational Query Performance. *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*, pp. 47–52. doi: 10.1109/iCCECOME.2018.8659222.

Corona, E. & Pani, F. E. (2013). A review of lean-kanban approaches in the software development. *WSEAS transactions on information science and applications*, 10(1), 1–13.

de Almeida, M. A., Lounis, H. & Melo, W. L. (1998). An investigation on the use of machine learned models for estimating correction costs. *Proceedings of the 20th international conference on Software engineering*, pp. 473–476.

Dima, A. M. & Maassen, M. A. (2018). From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. *Journal of International Studies*, 11(2), 315–325. Retrieved from: https://www.ceeol.com/search/article-detail?id=718102. Publisher: Fundacja Centrum Badań Socjologicznych.

Duc Anh, N., Cruzes, D. S., Conradi, R. & Ayala, C. (2011). Empirical validation of human factors in predicting issue lead time in open source projects. *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pp. 1–10.

Duggan, J., Byrne, J. & Lyons, G. (2004). A task allocation optimizer for software construction. *IEEE Software*, 21(3), 76–82. doi: 10.1109/MS.2004.1293077. Conference Name: IEEE Software.

El-Emam, K., Goldenson, D., McCurley, J. & Herbsleb, J. (2001). Modelling the likelihood of software process improvement: An exploratory study. *Empirical Software Engineering*, 6, 207–229.

Fabriek, M., Brand, M., Brinkkemper, S., Harmsen, F. & Helms, R. (2008). Reasons for Success and Failure in Offshore Software Development Projects. *16th European Conference on Information Systems, ECIS 2008*, 446-457.

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

Gamma, E., Vlissides, J., Helm, R., Johnson, R. & Ralph, J. (1995). *Design patterns: elements of reusable object-oriented software*. Reading, Mass: Addison-Wesley.

Gong, A., Yang, K., Lyu, J. & Li, X. (2024). A two-stage reinforcement learning-based approach for multi-entity task allocation. *Engineering Applications of Artificial Intelligence*, 136, 108906. doi: 10.1016/j.engappai.2024.108906.

Goswami, I. & Urminsky, O. (2020). More time, more work: How time limits bias estimates of task scope and project duration. *Judgment and Decision Making*, 15(6), 994–1008. doi: 10.1017/S1930297500008196.

Gupta, S., Sikka, G. & Verma, H. (2011). Recent methods for software effort estimation by analogy. *ACM SIGSOFT Software Engineering Notes*, 36(4), 1–5. doi: 10.1145/1988997.1989016.

Ikonen, M., Kettunen, P., Oza, N. & Abrahamsson, P. (2010). Exploring the Sources of Waste in Kanban Software Development Projects. *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications*, pp. 376-381. doi: 10.1109/SEAA.2010.40.

Imtiaz, S. & Ikram, N. (2017). Dynamics of task allocation in global software development. *Journal of Software: Evolution and Process*, 29(1), e1832. doi: 10.1002/smr.1832. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.1832.

Kim, G., Humble, J., Debois, P., Willis, J., Forsgren, N. & Allspaw, J. (2021). *The DevOps handbook : how to create world-class agility, reliability, & security in technology organizations* (ed. Second edition). Portland, OR: IT Revolution Press.

Kraut, R. E. & Streeter, L. A. (1995). Coordination in software development. *Commun. ACM*, 38(3), 69–81. doi: 10.1145/203330.203345.

Lamersdorf, A., Munch, J. & Rombach, D. (2009). A Survey on the State of the Practice in Distributed Software Development: Criteria for Task Allocation. *2009 Fourth IEEE International Conference on Global Software Engineering*, pp. 41–50. doi: 10.1109/ICGSE.2009.12.

Lin, C. & Snyder, L. (2008). *Principles of Parallel Programming* (ed. 1st). USA: Addison-Wesley Publishing Company.

Lin, J. (2013). Context-aware task allocation for distributed agile team. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 758–761. doi: 10.1109/ASE.2013.6693151.

Mahmood, S., Anwer, S., Niazi, M., Alshayeb, M. & Richardson, I. (2017). Key factors that influence task allocation in global software development. *Information and Software Technology*, 91, 102–122. doi: 10.1016/j.infsof.2017.06.009.

Mallidi, R. K. & Sharma, M. (2021). Study on agile story point estimation techniques and challenges. *Int. J. Comput. Appl*, 174(13), 9–14.

Mayez, M., Nagaty, K. & Hamdy, A. [arXiv:2202.01713]. (2022). Developer Load Normalization Using Iterative Kuhn-Munkres Algorithm: An Optimization Triaging Approach. arXiv. Retrieved on 2024-10-15 from: http://arxiv.org/abs/2202.01713.

Nagappan, N., Murphy, B. & Basili, V. (2008). The influence of organizational structure on software quality: an empirical case study. *Proceedings of the 30th international conference on Software engineering*, (ICSE ’08), 521–530. doi: 10.1145/1368088.1368160.

Nakra, V., Dave, A., Chenchala, P. K. & Agarwal, A. (2023). Enhancing Software Project Management and Task Allocation with AI and Machine Learning. *International Journal on Recent and Innovation Trends in Computing and Communication*, 11, 1171–1178.

Nundlall, C. & Nagowah, S. (2021). Task allocation and coordination in distributed agile software development: a systematic review. *International Journal of Information Technology*, 13(1), 321–30. doi: 10.1007/s41870-020-00543-4. Place: Germany Publisher: Springer.

OpenAI. (2025). https://chatgpt.com/ [Large Language Model (LLM)]. Retrieved on 2025-01-15 from: https://chatgpt.com/.

Poppendieck, M. & Cusumano, M. A. (2012). Lean Software Development: A Tutorial. *IEEE Software*, 29(5), 26–32. doi: 10.1109/MS.2012.107. Conference Name: IEEE Software.

Rajbahadur, G. K., Wang, S., Kamei, Y. & Hassan, A. E. (2019). Impact of discretization noise of the dependent variable on machine learning classifiers in software engineering. *IEEE Transactions on Software Engineering*, 47(7), 1414–1430.

Roque, L., Araújo, A. A., Dantas, A., Saraiva, R. & Souza, J. (2016). Human Resource Allocation in Agile Software Projects Based on Task Similarities. *Search Based Software Engineering*, pp. 291–297. doi: 10.1007/978-3-319-47106-8_25.

Shafiq, S., Mashkoor, A., Mayr-Dorn, C. & Egyed, A. (2021). TaskAllocator: A Recommendation Approach for Role-based Tasks Allocation in Agile Software Development. *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*, pp. 39–49. doi: 10.1109/ICSSP-ICGSE52873.2021.00014.

Torres, A., Galante, R., Pimenta, M. S. & Martins, A. J. B. (2017). Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, 82, 1–18. doi: 10.1016/j.infsof.2016.09.009.

Trudel, S. & Boisvert, M. (2011). *Choisir l'agilité*. [S.l.]: Dunod. Retrieved from: https://research.ebsco.com/linkprocessor/plink?id=826e63e4-6107-36b4-8909-ec91fba52433.

Tsai, H.-T., Moskowitz, H. & Lee, L.-H. (2003). Human resource selection for software development projects using Taguchi's parameter design. *European Journal of Operational Research*, 151(1), 167–180. doi: 10.1016/S0377-2217(02)00600-8.

Usman, M., Britto, R., Damm, L.-O. & Börstler, J. (2018). Effort estimation in large-scale software development: An industrial case study. *Information and Software Technology*, 99, 21–40. doi: 10.1016/j.infsof.2018.02.009.

Vacanti, D. & Coleman, J. (2020). Kanban Guide. Retrieved on 2025-01-24 from: https://kanbanguides.org/english/.

Vanderster, D. C., Dimopoulos, N. J., Parra-Hernandez, R. & Sobie, R. J. (2009). Resource allocation on computational grids using a utility model and the knapsack problem. *Future Generation Computer Systems*, 25(1), 35–50. doi: 10.1016/j.future.2008.07.006.

Wang, X., Conboy, K. & Cawley, O. (2012). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *Journal of Systems and Software*, 85(6), 1287–1299. doi: 10.1016/j.jss.2012.01.061.

William, P., Kumar, P., Chhabra, G. S. & Vengatesan, K. (2021). Task Allocation in Distributed Agile Software Development using Machine Learning Approach. *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, 1, 168–172. doi: 10.1109/CENTCON52345.2021.9688114.