Extraction and Analysis of Behavior Practices Based on GitLab MR Information

by

Seyedbehnam MASHARI

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 25, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Francis Bordeleau, Thesis Supervisor
Department of Software Engineering, École de technologie supérieure

Mr. Mohammed Sayagh, Thesis Co-supervisor
Department of Software Engineering, École de technologie supérieure

Mr. Kaiwen Zhang, President of the Board of Examiners
Department of Software Engineering, École de technologie supérieure

Mrs. Naouel Moha, Member of the jury
Department of Software Engineering, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC

APRIL 08, 2022

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGMENTS

I would like to take this opportunity to thank my main advisor Prof Francis Bordeleau who supported me in this circuitous path toward doing my thesis. I appreciate his kind support without which success in doing my master thesis could be impossible. I also would like to thank my co-advisor Dr. Mohammed Sayagh whose expertise and knowledge were an asset to our research project.

I would like to thank our industrial partner, Kaloom for their financial support of this research work. Indeed, working under the industrial supervision of such a great character as Dr. Bassem R. Guendy (CICD DevOps Specialist at Kaloom) was a unique opportunity that I was lucky to grasp.

Last but not least, I am grateful to my wife, Azadeh who patiently supported me despite the challenges of student life.

Extraction et analyse de modèles de comportement basés sur les informations GitLab MR

Seyedbehnam MASHARI

RÉSUMÉ

L’évolution constante de la technologie dans les logiciels, l'informatique et les télécommunications est sans aucun doute associée aux défis émergents concernant la capacité à fournir des logiciels plus rapidement et avec une meilleure qualité. L'adoption de DevOps par de nombreuses entreprises, dans différents domaines d'application, au cours de la dernière décennie a entraîné des progrès majeurs sur ces deux aspects. Cependant, pour atteindre les objectifs de DevOps, une approche systématique de l’amélioration continue des différentes phases du processus de livraison de logiciels doit être établie.

Cette thèse porte sur l'amélioration de la phase de revue de code du processus à partir d'un cas d'étude industriel fourni par Kaloom. L'objectif global est d'étudier différentes techniques statistiques et d'apprentissage automatique (« machine learning ») qui peuvent être utilisées pour extraire et analyser des modèles de comportement à partir des données enregistrées par GitLab pendant la phase d'examen par les pairs de la demande de fusion (Merge Request (MR)). À cette fin, nous nous concentrons sur trois questions de recherche principales : QR1) Comment le délai d'exécution des MR évolue-t-il au cours des 21 jours d'un sprint, QR2) Quels sont les comportements des ingénieurs logiciels pendant les jours de sprint, et QR3) Quels sont exceptions et valeurs aberrantes dans les données IRM. Nos principales conclusions incluent qu'il n'y a pas de corrélation entre le délai d'exécution et la taille des MR dans les groupes étudiés, que les groupes avec des tests matériels ont un délai d'exécution moyen plus élevé, et qu'il existe une très faible corrélation de Spearman entre le délai d'exécution d'un MR et le jour où un MR a été créé ou fermé. De plus, grâce à l'analyse manuelle des MR d'exception, nous avons observé que certains MR sont commentés après la fusion et que ces MR sont souvent associés à des erreurs découvertes ultérieurement lors des différentes phases de test des produits.

Mots-clés : DevOps, revue de code, la demande de fusion, techniques statistiques, d'apprentissage automatique, les comportements

Extraction and analysis of behavior practices based on GitLab MR information

Seyedbehnam MASHARI

ABSTRACT

The constantly evolving nature of technology in software, IT, and telecommunication is no doubt associated with emerging challenges regarding the ability to deliver software faster and with higher quality. The adoption of DevOps by many companies, in different application domains, in the last decade has resulted in major progress regarding both aspects. However, to reach the objectives of DevOps, a systematic approach to the continuous improvement of the different phases of the software delivery process needs to be established.

This thesis focuses on the improvement of the code review phase of the process based on an industrial case study provided by Kaloom. The overall objective is to investigate different statistical and machine learning techniques that can be used to extract and analyze behavior practices from the data recorded by GitLab during the Merge Request (MR) peer-review phase. For this purpose, we focus on three main research questions: RQ1) How does the Lead Time of MRs change over the 21 days of a sprint, RQ2) What are the behavior practices of software engineers during the sprint days, and RQ3) What are exceptions and outliers in MR data. Our main findings include that there is no correlation between the Lead Time and Size of MRs in the studied groups, that the groups with hardware tests have higher average Lead Time, and that there is a very weak Spearman correlation between the Lead Time of an MR and the day on which an MR was created or closed. Furthermore, through manual analysis of exception MRs, we observed that some MRs are commented after the merge and that these MRs are often associated with errors discovered later during the different product testing phases.

Keywords: DevOps, peer-review, Merge Request**,** statistical analysis**,** machine learning, behavior practices

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABREVIATIONS AND ACRONYMS

API	Application Programming Interface
CSS	Closed-source Software
CICD	Continuous Integration, Continuous Delivery
CP	Control Plane
DP	Data Plane
DevOps	A term given to the combination of Development and Operations
DVCS	Distributed Version Control Systems
ML	Machine Learning
MP	Management Plane
MR	MRs
MSR	Mining Software Repositories
NFV	Network Function Virtualization
OSS	Open-source Software
PF	Platform
PR	Pull Request
SDN	Software-defined Networking
WoW	Way of Working

INTRODUCTION

0.1 Context and motivation

The constantly evolving nature of technology in software, IT, and telecommunication is no doubt associated with emerging challenges. One of the key challenges in today's 5G telecom network is reducing the service creation time cycle as a key performance indicator (KPI) (D. Liu et al., 2016; Mechtri, Benyahia, & Zeghlache, 2016; Nakimuli et al., 2020). From an architecture perspective, such improvement requires the adoption of new technologies such as software-defined networking (SDN). From a software process perspective, companies developing telecom solutions need to increase their agility to deliver solutions faster and with higher quality. The adoption of DevOps by many companies in the last decade (e.g. Google, Amazon, Facebook, Netflix) (Kim, Humble, Debois, Willis, & Forsgren, 2021), as an evolution of the Agile practices, has resulted in major progress.

Despite the benefits that DevOps has provided, various challenges may arise during its implementation and evolution. As stated by Nazim Benhadid, vice-president of Cloud Infrastructure at TELUS, "The current pandemic has highlighted the importance of being able to adapt quickly to changes and to crises that arise. In this context, the use of DevOps becomes unavoidable, but the implementation and evolution of DevOps processes represent a significant challenge for companies." ("ÉTS Welcomes the Kaloom-TELUS Industrial Research Chair | ÉTS Montréal," n.d.)

Kaloom, as the industrial partner of our research work and a use case for the current study, is a software solution provider in data center networking. Their software delivery process is composed of six main phases: Coding, Dev Pipeline, Review, Product Build/Packaging, Product Integration/Testing, and Product Feature Testing. Kaloom is working on establishing a systematic approach to support the continuous improvement of their process. A first objective consisted in improving the Product Build/Packaging phase. The efforts invested in this allowed

reducing the product build time from 90 minutes at the end of 2018 to about 11 minutes in 2020.

The research of this thesis focuses on a different objective that aims at improving the Review phase of the software delivery process. This phase consists in having every code modification submitted by a developer systematically reviewed by one or more group members to check for potential issues. In this phase software, engineers use their knowledge and expertise to look for potential issues and quality factors that cannot be automatically done by the different tools used in the DevOps process, like the different types of testing tools and static code analysis tools.

There are several reasons for focusing on the improvement of this phase. First, it is one of the few phases, together with the design/coding phase, that cannot be automated and needs to be manually performed by group members. As a result, it consumes time from key software engineering resources, and we need to ensure that these resources are used as efficiently as possible.

Second, the peer review phase has a direct impact on both the first way (Flow) and the second way (Feedback) of the three ways of DevOps defined by (Kim et al., 2021). Regarding the Flow, the time and effort taken for peer review accounts for a significant portion of the development time. Regarding Feedback, the information provided by team members and subject experts during the review process plays a key role to ensure software quality. Also, the time taken for the review has a direct impact on quality; the faster the feedback the more efficient it is and the higher is the resulting quality. Moreover, the problems not found in the Review phase will later result in errors found in the different testing phases or worst will end up being discovered by product users, which would create business issues for the company. Consequently, improving the Review phase will ultimately result in improving both the flow (i.e. shorter time to deliver new functionality/value) and feedback, which leads to increased quality.

Kaloom's software delivery process is based on 3-weeks sprints and involves four main product groups: Control Plane group (CP), Data Plane group (DP), Management Plane group (MP), and Platform group (PF). All groups are using a unified super-repository in GitLab consisting of distinctive sub-project repos to submit their codes. Therefore, the same metrics used by a group can be utilized to measure code review performance across all other groups. However, since the culture, experience, and expertise of group members and on top of that, their way of working (WoW) differs, analysis of the performance metrics as well as interpretation of the results can be different group by group and should be conducted in the context of each group.

0.2 Problem and research questions

The problem addressed by this research project is the lack of methods and tools that can be used to analyze and improve the software Review phase. The overall objective is to investigate different techniques that can be used to extract and analyze behavior practices from the data recorded by GitLab during the Merge Request (MR) peer-review process.

The primary challenge in improving every process is to understand it by measuring its different aspects. To understand the Review process in different groups at Kaloom, we need to investigate how MR related metrics change over different days and weeks of the sprints, and how these metrics are related in the context of each group. Therefore, our research questions concern understanding the different aspects of Review process in different groups by measuring how Lead Time and Size of merges change on sprint days, to what extent these metrics are correlated, and understand whether developers use the Git mechanism for other purposes than reviewing code. For this purpose, we focus on three main research questions.

- **RQ1: How does the Lead Time of MRs change over the 21 days of a sprint?**
  One of the important aspects of this study is to explore how Lead Time and Size of the MRs change during sprint days in different groups at Kaloom. The goal of this research

question is to investigate and justify differences between groups regarding their different characteristics and workflow. Furthermore, we are interested to explore the correlation between MR Lead Time, Size, and Sprint Days in these groups.

- **RQ2: What are the behavior practices of software engineers during the sprint days?**

  Removing barriers of a fast-paced flow requires identifying behavior practices of both developers and reviewers in either submitting or evaluating the MRs. Some of these behavior practices could be linked with undesired habits of engineers. We want to know if there are effects or syndromes associated with the behavior of engineers over different sprint days. “End of the sprint” syndrome is a good case in point where engineers feel a sense of urgency to merge open MRs before the end of a week or sprint. Or “Friday Afternoon” effect in which created MRs on Fridays are expected to be more error-prone.

- **RQ3: What are exceptions and outliers in MR data?**

  Addressing this research question aims at the exploration of abnormal information in MRs. It includes investigation of different categories of MRs. For example, MRs with very large code changes that are accepted in less than a minute after creation, those that are commented after the merge, or the ones very far from other MRs. These MR cases should be investigated and the reason behind them should be found out. It can help decision-makers at Kaloom to distinguish avoidable and non-avoidable MR conditions to improve the review process where applicable. We are also interested to investigate if there is any relation between the occurrence of these exceptional MRs and their creation sprint day.

0.3 Methodology

To achieve the objectives and answer the research questions, this research project was decomposed into the following main tasks.

Task 1- Documentation of the Kaloom software delivery process and the specificities of each group -- This task aims at documenting and modeling the DevOps process at Kaloom with the primary focus on the peer review phase. This requires understanding the existing workflow at Kaloom and documenting the main particularities of each group that affects their way of working and behavior practices.

Task 2 – **Extraction of the MR data for each studied group** – This task aims at extracting MR data recorded by GitLab during the Review phase. This is achieved using the GitLab MR Extraction tool ("ETS DEVOPS / Gitlab-Mr-Extraction," n.d.) and grouping the extracted data in a unified CSV file for each Kaloom group. The research of this thesis is based on the MR data of two years, 2020 and 2021.

Task 3 -- **Statistical analysis of the MR Data** -- The goal of this task is to investigate how the Lead Time and Size of MRs evolve over different sprint days. This includes using the statistics correlation method to uncover relations between metrics in four studied groups. Statistical analysis is also used to compare the Lead Time and Size of MRs in each of these groups.

Task 4 -- Manual and quantitative analysis of abnormal MRs --This task aims at detection and exploration of abnormal MRs. Achieving the goals of this task requires manual investigation of exceptions and using an appropriate automatic outlier detection method. Manually detecting exceptional cases of MRs includes investigation of both Kaloom's MR data and repository. Selecting an automatic outlier detection method is done according to the distribution type of MR data.

0.4 Contributions

In line with the three main research questions addressed by the thesis, the main contributions are:

- Combination of different statistical analysis to analyse the Lead Time (LT) and Size of MRs over the 21 days of the sprints in the different product groups at Kaloom. The graph resulting from this analysis allows visualizing different behavior practices.
- Statistical analysis of the correlation of different MR-related metrics in different groups using the Spearman method.
- Manual and quantitative analysis to detect and investigate exceptions and outliers in MR data.

0.5 Research outline

This thesis is organized as follows.

Chapter 1 provides a review of the literature relevant to this thesis. It includes the analysis of the data obtained from mining software repositories (MSR), metrics related to the code review phase, methods and techniques used to analyze MR metrics, and outlier detection approaches in MSR data.

Chapter 2 discusses a background related to the software delivery process and code Review phase at Kaloom. Furthermore, we describe GitLab MR mechanism in this chapter. We also explain the main duties and characteristics of different groups at Kaloom. Moreover, we describe the MR Extraction tool, defined metrics, and the MR dataset collected through its integration on Kaloom's software super-repository.

Chapter 3 is divided according to the research questions, and related approaches used to address them. Then, findings and results are described in response to the research questions. To address RQ1, statistical data analysis, and visualizing data distribution are conducted to explore how Lead Time and Size change during 21 days of the sprint. Furthermore, the correlation of metrics is investigated, and the characteristics of each group are compared in the context of each group. To address RQ2, we conduct a set of statistical analysis to explore behavior practices of software engineers during different days of the sprint. To address RQ3, we carry out manual and quantitative analysis of MR data to detect abnormal cases in two categories of exceptions and outliers.

Finally in Discussions and Conclusions, we state conclusions, limitations, threats to validity, and possible future works associated with this study.

CHAPTER 1

LITERATURE REVIEW

In this chapter, we provide a review of relevant literature. The chapter is decomposed into four subsections each focused on the main aspect of our research.

In the first section (1.1), we review the available literature on the analysis of the data obtained from mining software repositories (MSR). In section 1.2, we discuss defined metrics related to the code review phase in the relevant research works. Then in section 1.3, we discuss the methods and techniques used to analyze MR metrics in the related articles. Finally, in section 1.4, we investigate applied outlier detection approaches in MSR data.

1.1 Analysis of the MSR Data with the focus on the peer review process

During the recent decades, data mining has had a range of applications in either the software industry or academia (Halkidi, Spinellis, Tsatsaronis, & Vazirgiannis, 2011; Husain, Low, Ng, & Ong, 2011; Xie, Thummalapenta, Lo, & Liu, 2009). Mining software repositories (MSR) - like one of these applications- has resulted in resolving research questions with the main focus on optimizing the peer-review process (Chatley & Jones, 2018; Liang & Mizuno, 2011; Mishra & Sureka, 2014; Paixao, Krinke, Han, & Harman, 2018). Various tools and frameworks have been used to mine software repositories (Chatley & Jones, 2018; Paixao, Krinke, Han, & Harman, 2018; Spadini, Aniche, & Bacchelli, 2018), most of which include using code hosting application programming interfaces (APIs) (Gousios & Zaidman, 2014; Rose, 2017; Tsay, Dabbish, & Herbsleb, 2014; Yu, Wang, Filkov, Devanbu, & Vasilescu, 2015).

These efforts are not limited to analyzing code commits of project contributors in (Farias, Novais, Ortins, Colaço, & Mendonça, 2015; Heller, Marschner, Rosenfeld, & Heer, 2011; Liang & Mizuno, 2011). An acceptable number of studies on extracting behavior practices of

software engineers based on the information obtained from MSR have been conducted. Most of these studies are in the context of Open-source software projects (OSS). Utilized techniques include both quantitative and qualitative analysis of the detailed observations of developers. Extracting contributors' personality traits on the evaluation process of pull-requests in (Iyer, Yun, Nagappan, & Hoey, 2021; Robillard, Coelho, & Murphy, 2004) is among the adopted approaches in this area.

Some research works in this context have tried to establish a link between metrics in the code review process. Systematic methods to directly relate important metrics such as Code Churn, Review Time, and Code Quality were discussed in (Bird et al., 2009; Kononenko, Baysal, Guerrouj, Cao, & Godfrey, 2015; Rigby, German, Cowen, & Storey, 2014; Shimagaki, Kamei, McIntosh, Hassan, & Ubayashi, 2016; Wagstaff, Cardie, Rogers, & Schrödl, 2001; Yu, Wang, Filkov, Devanbu, & Vasilescu, 2015). However, the popularity of pull-based development procedure has required more comprehensive studies on effective metrics, elements, and working habits of contributors in the workflow (Gousios, Pinzger, & Deursen, 2014; Gousios, Zaidman, Storey, & Deursen, 2015). Determinants to the state and time of merge during the Pull request process in Git -as the most notable Distributed Version Control System (DVCS)- have been examined(Yu, Yin, Wang, Yang, & Wang, 2016).

Georgios Gousios et al., have shown the impact of the pull request, project, and developer characteristics on Pull Request lifetime and Size in OSS projects. Furthermore, the importance of defined metrics related to the pull-request lifecycle on predicting merge decisions - as the target variable - has been scrutinized (Gousios et al., 2014).

1.2 Metrics and behavior practices of code development contributors

Not only pull-request level metrics, but also project-level, submitter-level, workflow, and continuous integration metrics have been considered across most common languages in OSS projects (Kononenko et al., 2018; Rose, 2017; Silva, Valente, & Terra, 2016; Tsay, Dabbish, & Herbsleb, 2014; Yu, Wang, Filkov, Devanbu, & Vasilescu, 2015; Yu et al., 2016). By picking out (Li, Yu, Yin, Wang, & Wang, 2017) as a reference to reach a group of relevant

papers, a list of metrics was collected. Table 1.1 provides a summary of the metrics defined in the literature classified based on their level of definition (project-level, pull-request level, submitter-level, workflow-level, and CI related). Moreover, a description of each metric and related hypotheses is mentioned in Table 1.1.

Table 1.1 Explored metrics in the context of industrial and open-source software projects in the literature review

Metrics level	**Metric**	**Description**	**Hypothesis**
Project-level	Project age	The time passed since the creation of the project	Mature projects probably lead to different contribution flows, which may affect the processing pace
	Group Size	Number of code reviewers/maintainers/integrators	Larger reviewer/integrator groups handle a larger volume of incoming PRs/MRs
	Hotness of area	the median of the number of commits to the number of changed files in all commits of a pull-request in a specific period such as a release cycle	Do reviewers need to spend more time inspecting pull-requests of hot areas to prevent possible code conflicts?
Pull-request level	Code churn	Total number of deletions and additions in a pull request	Vaster code changes may need a longer evaluation time
	Number of commits	Number of commits in a pull request	A higher number of commits may impact PR/MR Size and therefore, review time
	Description complexity	The complexity of the title and description of a pull request	

Table 1.1 Explored metrics in the context of industrial and open-source software projects in the literature review (cont'd)

Pull-request level	Test inclusion	Whether a pull request has touched a minimum of one test file	-
Submitter-level	Core group membership	Whether the submitter is a core-group member or not (defined for OSS projects)	MRs created by core group members may be merged faster
	Social distance	The proportion of group members that have had an interaction with the submitters in a specific period (defined in OSS projects)	Social distance affects MR inspection time
	Number of code hosting site's users who follow submitters	Number of followers in code hosting site as a criterion which shows the reputation of contributors (defined in OSS projects)	The more well-known contributor, the more probable code is merged
	PR author Experience	Experience of PR author based on the number of prior PRs submitted by PR author	The experience of the PR creator influences the merge status and its processing time
Workflow-level	Number of comments	Number of comments in a pull-request discussion	Pull-requests with a high number of comments are more likely to cause latency in final decision or signal disapproval
	First human response	The time it takes minutes for the first reviewer to respond to a pull request	-

Table 1.1 Explored metrics in the context of industrial and open-source software projects in the literature review (cont'd)

Workflow-level	Reviewer Workload	the total number of open pull-requests at the creation time of each pull-request	The higher number of open PRs means a larger wait time for the just created PR
	Reviewer availability	How many hours per day is the most active reviewer accessible? (The most active reviewer is chosen from the list of main reviewers)	The impact of reviewer availability on the review process duration.
	Reviewer/ integrator experience	Experience of the code reviewer based on the number of prior evaluated PRs	The experienced reviewers evaluate changes faster, reducing the decision-making and process time
	Number of assigned reviewers per issue	Self-explanatory	-
	Number of real reviewers per issue	Number of reviewers who have commented on the PR or have decided on it	-
	tags	Whether the pull request contains #issue tag and @mention tag	-
	Friday effect	Check if the PR is created on Friday	PRs created on Fridays are more likely to be error prune/buggy; (may be associated with larger processing time or higher rework commits?)

Table 1.1 Explored metrics in the context of industrial and open-source software projects in the literature review (cont'd)

Workflow-level	Technical debts	Not quantitatively measured (based on asking reviewers and contributors)	Does technical debt cause MR rejection?
CI related	CI failure	a binary indicating presence of build & test failures during the running of hosted CI services	Is the presence of CI a deciding factor for the final state of PR (approved or closed) or its latency?
	CI latency	the period from PR creation to the last commit which was tested by CI	

However, because of the Closed-source Software (CSS) nature of the Kaloom project, metrics specific to the OSS projects were intentionally filtered out. This can give a more practical perspective to explore influential metrics in our use case context.

1.3 Methods and techniques to analyze metrics

In this subsection, different methods and techniques in the literature implemented to analyze MSR data are asserted. These techniques are used to uncover the relation between defined metrics:

- Correlation methods
- MSR data visualization
- ML techniques including clustering

Correlation methods: In some of the related papers, analyzing MSR data aims at uncovering the effect of each metric on the review time in the release cycle. A primary step to achieve this is to explore how different metrics are correlated with code review time (Liang & Mizuno, 2011). Cross-Correlation between metrics is illustrated using Spearman's rank-order method for feature selection purposes (Gousios & Zaidman, 2014; Liang & Mizuno, 2011).

Data visualization: Visualization, as an effective tool, has been widely used in studies to graphically reveal general trends that are initially invisible in the extracted data from software repositories. Busy weekdays, popular languages used, the geographical spread of contributors, contribution reasons, and frequent domains in well-known sharing platforms such as GitHub have been visualized (Bertoncello, Pinto, Wiese, & Steinmacher, 2020; Kumar J., Dubey, Balaji, Rao, & Rao, 2018; Padhye, Mani, & Sinha, 2014). Uniformity and non-uniformity of Developers' Contribution graphs for a shared code have enabled monitoring work efficiency across involved members at a glance (Bradley, 2017).

ML techniques including clustering: Different ML techniques and algorithms have been used in the context of the code review process in software projects. Code reviews for GitHub's popular OSS projects have been automatically classified based on ML techniques to examine the effect of code reviewing by external contributors (Z.-X. Li, Yu, Yin, Wang, & Wang, 2017). Furthermore, ML is used to automate the code review process (Lal & Pahwa, 2017; LI et al., 2019; Shi, Li, Lo, Thung, & Huo, 2019). Some ML algorithms are implemented to analyze MSR data in the code review phase including Neural Networks, Classification models, and clustering (Meqdadi, Alhindawi, Alsakran, Saifan, & Migdadi, 2019; Namiot & Romanov, 2020). Using Hierarchical Clustering of code communities in (Thomas, Hassan, & Blostein, 2014) is another good case in point. However, there seems to be a need for conducting comprehensive research works on using ML to uncover trends and practices behind MSR data. This can include clustering of MSR data samples.

1.4 Outlier detection in MSR data

Outliers are abnormal observations far from the distribution of mass of data. Easy though it is to uncover separated anomalies in a data set with a limited number of features -especially when there are only two or three features where data can be plotted-, finding out outliers in a data set with a higher number of features requires deploying automatic detection methods. The efficiency of automatic outlier detection depends on the type of data distribution. Whereas such methods as z-score or Elliptic Envelope are good candidates to discover anomalies of data

samples with Gaussian distribution, other methods as Isolation Forest(F. T. Liu, Ting, & Zhou, 2008), One-class SVM(Amer, Goldstein, & Abdennadher, 2013; Hejazi & Singh, 2013; Hu, Liao, & Vemuri, n.d.), or DBSCAN are considered robust algorithms that need no assumption on the data distribution.

Detecting outliers in data set obtained from MSR is implemented for two different purposes in the literature:

- Improving the ML-based prediction models
- Investigation of the typical and non-typical trends and practices

These approaches are discussed below.

- **Improving the ML-based prediction models:** Reported bugs, errors, and feature requests should be prioritized during the triaging phase. By filtering out outliers from the data set, the performance of the model increases. Therefore, the higher prediction power of the model results in better prioritization and lower waiting times for the MRs (AbdelMoez, Kholief, & Elsalmy, 2013; Giger, Pinzger, & Gall, 2010; Seo & Bae, 2013).

- **Investigation of the typical and non-typical trends and practices in the submission of code change** (Alali, Kagdi, & Maletic, 2008): Normal and abnormal trends during the development activities such as commits were examined. This was conducted based on defining extreme and mild outliers about the typical Size of normal commits.

CHAPTER 2

KALOOM DEVOPS PROCESS

This chapter describes the core aspects of the Kaloom DevOps process that are directly relevant to this thesis. We initially describe Kaloom's software delivery process in section 2.1. Then we will discuss the Review process at Kaloom (section 2.2). We explain the use of GitLab and its merge request (MR) mechanism in Kaloom's peer review process (section 2.3). We discuss characteristics of the groups involved in the software delivery process at Kaloom discussed (section 2.4). Finally, we introduce the GitLab MR Extraction tool, defined metrics, and the MR dataset (section 2.5).

2.1 Kaloom's software delivery process

This subsection aims at describing the software delivery process at Kaloom. Figure 2.1 illustrates the overall software delivery process of Kaloom (Bordeleau, Cabot, Dingel, Rabil, & Renaud, 2020). It is composed of six main phases: Coding, Dev Pipeline, Review, Product Build/Packaging, Product Integration Testing, and Product Feature Testing. The product consists of multiple microservices where each microservice represents a Git submodule in the Git repository of the product.

In the Coding phase, Git commit(s) in GitLab™ are pushed toward the Dev branch of the related microservice code by the developer. In the Development pipeline, the Dev branch triggers a build on Jenkins to compile the code changes and provide feedback. In this phase, unit tests are executed during the compilation. Then, Static Code Analysis is executed by SonarQube™ to validate different aspects of the code and identify issues. In the last stage of the development pipeline, different types of tests are run based on the duties of each group. This includes running tests of the concerned microservice on:

- A simulated environment for CP and MP groups,
- A hardware test environment in DP and PF groups.

If any of these phases fail, the code commit will be tagged as unmergeable to the master branch until the problem is resolved. Once the development pipeline passes, the code change moves on for peer-review in the Review phase. Following an iterative process, comments from reviewers are required to be addressed by the developer. After that a decision is made on the MR, developer opens the MR to merge the related Dev branches to the master. This triggers the product build and packaging which includes an initial quality level (QL) test. Different QLs in the software delivery process are associated with different levels of testing. Higher QL tests in a phase of product build are the more costly and time-consuming ones. This is because higher QL tests evaluate the robustness and stability of the products.

QL0 is the basic level associated with every new build, i.e. meaning that the Product Build/Packaging phase has successfully completed. QL1 in the product feature testing phase is a basic feature test whereas QL2 as a higher-level test takes a longer time than QL1. Errors not detected in the Review phase may be marked during the QL tests.

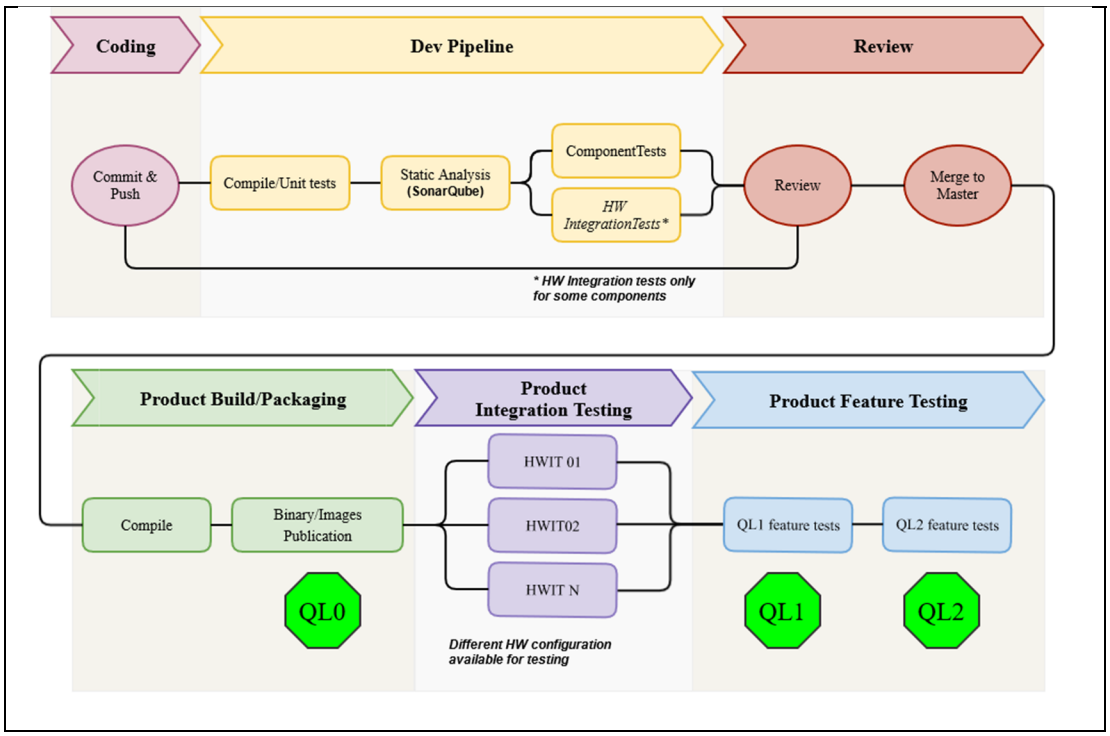

Figure 2.1 Software delivery process at Kaloom
Taken from (Bordeleau et al., 2020)

2.2 Review process at Kaloom

The goal of this chapter is to describe the Review process at Kaloom. As we previously discussed in chapter 1 of this thesis, the focus of this research with optimization purposes is on the review phase. Any effort towards detecting current behavior practices of either developers or reviewers can lead to an improvement in the review process of MRs.

As we discussed in section 2.1, the review phase in software delivery is an interactive process between developers and reviewers. Therefore, the process is usually completed after discussions between both sides. The arrow from “Review block” in the Review phase to “Commit & Push” block in the Coding phase shows the iterative nature of reviewing code and interactions between developers and reviewers.

MR is the GitLab mechanism for developers to commit code changes into a branch. These code changes are reviewed, and comments are put as feedback by the reviewer. Once all the comments and discussions are addressed by the developer, MRs are approved to be merged in the concerned branch. MRs are approved sometimes by the most experienced member of the groups or group leaders. Before the MR is approved all discussions should be resolved. GitLab prevents merging until all of the discussions and comments are resolved. Whenever the comments are addressed, the pipeline is triggered automatically.

Despite the similarities, the Review process in different groups at Kaloom differs. This is mainly because of the different characteristics, responsibilities, and used programming languages in each group. We discuss these differences in section 2.4.

2.3 GitLab and GitLab MR

Kaloom uses Gitlab at the core of its development environment and the GitLab MR mechanism for the review process. The goal of this section is to describe the GitLab MR process, and the roles and artifacts involved in it.

GitLab MR mechanism includes three main roles: Developer, Reviewer, and Maintainer. The Developer is an individual that changes codes according to coming feature requests and bugs in a software delivery platform. The Reviewer is the person who continuously and systematically inspects submitted code changes by developers. The Maintainer is responsible for integrating evaluated changes to the main branch. This mechanism also includes three main artifacts: Master Branch, Feature Branch, and merge request (MR). Master Branch is a naming convention in Git and is the default branch that holds the main software releases. Code development should occur in an assigned branch instead of the main branch. This branch is called Feature Branch and enables Developers to work on different features and bugs without disturbing the main branch. MR is a request from the Developer to merge a branch into another -for example from a Feature Branch toward the Master.

A conventional MR lifecycle in GitLab can be seen in the sequence diagram of Figure 2.2. According to its flow, the developer clones the repository and creates a feature branch to work on. After adding files from the working copy to the staging area, changes are committed to the local repo. Eventually, the commit(s) are pushed into a shared remote repo and requested to be merged to the main branch where the code reviewer can start inspection on code modifications and remark comments about changes.

A final decision is usually made through interactions between reviewer and developer to reject or approve requests. There might be also another role defined in workflow responsible for making the final decisions on code integrations to the master/main branch. For instance, in the use case of Kaloom, each group may benefit from different levels of authorities as maintainer(s) to either reject or merge the incoming requests.

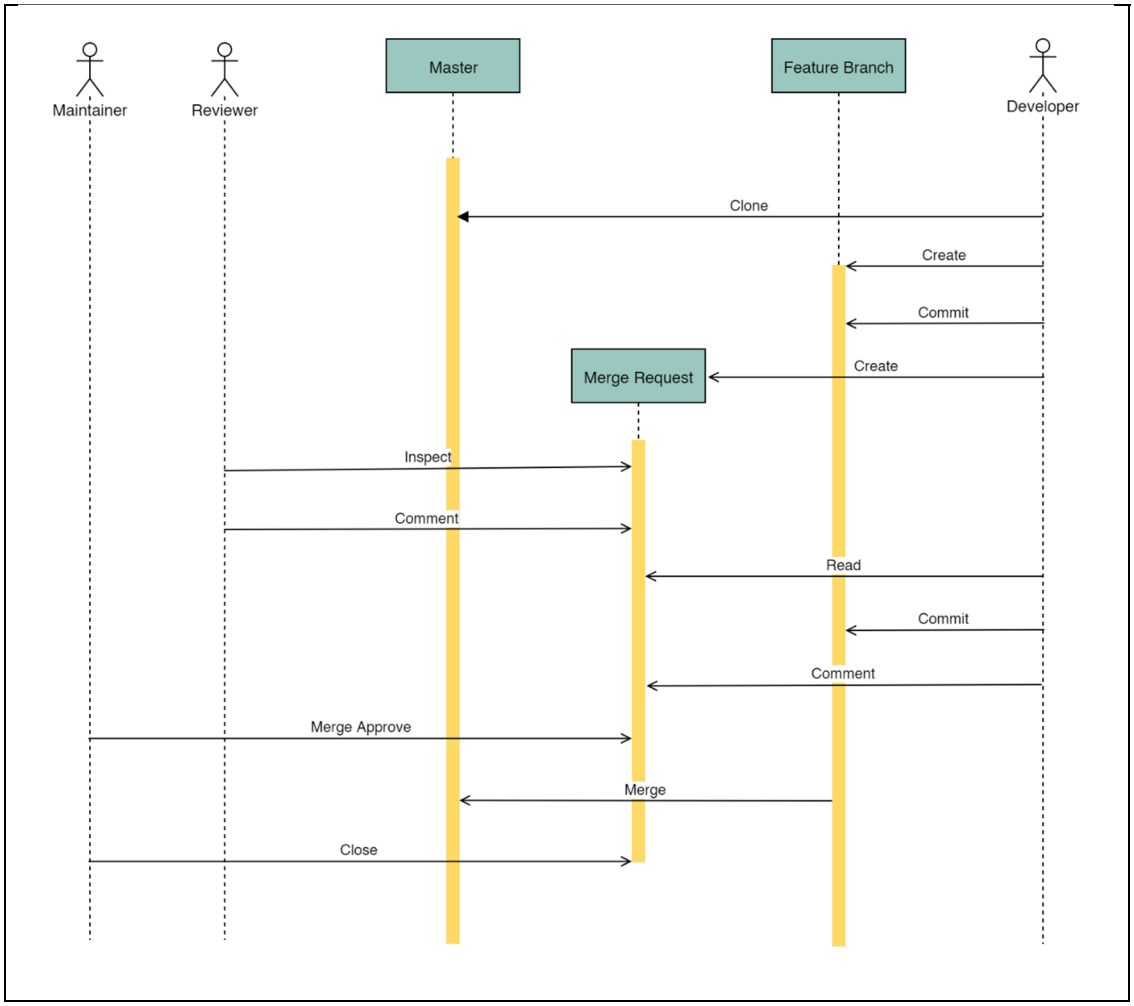

Figure 2.2 Sequence diagram of a conventional GitLab's MR process in an industrial software project

2.4 Profile of the different Kaloom groups

This section aims at collecting and documenting characteristics, responsibilities, languages, and tools for different groups at Kaloom. In addition to the main duties of each group, the type of used tests, allocated repositories, and the number of group members differ across these groups. We document these characteristics in Table 2.1 to explore specific behavior practices of these groups later in statistical analysis in section 3.2.

Table 2.1 Characteristics of each working group involved in the software delivery process at Kaloom

Group	Main responsibilities	Programming languages	Number of projects in GitLab	Type of test	Group Size	Particularities
Control Plane	Functions and processes that determine which path should be used for sending packets and frames	Golang	21 projects	SW Test	11	-
Data Plane	Functions and processes that forward packets and frames from/to interfaces based on Control Plane logic	P4 (main language), Golang (for feature test)	5 projects	HW Test	15	- No static code analysis for P4
Management Plane	Management of the fabric - Maintain the data model for the Fabric - Communication between MP and the Fabric uses YANG models	Golang (mainly)	9 projects	SW Test	7	-
Platform	Manages the low-level platform files and configurations	Red Hat Linux after OpenShift - Bash and OpenShift configuration files (YAML)	22 projects	HW tests	8	-

As it is noted in Table 2.1, each group at Kaloom is responsible for specific roles in the software delivery process. The CP and MP groups use Golang as their programming languages. Coding technology and the programming language used in the DP and PF groups are different. DP uses P4 which is a domain-specific language in programming networking devices. Red Hat Linux and Open Shift are the main technologies used in the software delivery contribution by the PF group. Therefore, the code review process in the DP and PF groups might be different than that in CP and MP groups. This is partly due to the low-level hardware coding in the DP group and working with low-level platform files and configurations in the PF group.

Each group involved in the software delivery process works on several projects in GitLab. Furthermore, the type of tests in each group varies. Whereas submitted code changes in the CP and MP go through software component tests, code submissions in DP and PF pass hardware integration tests. Notably, the lack of tools and technology for static code analysis in the P4 programing language is a specific particularity of the DP group, which can affect the interpretation of practices in this group.

2.5 GitLab MR data extraction

The goal of this section is to introduce the MR Extraction tool and the MR dataset collected through its integration on Kaloom's software super-repository. This follows by describing defined metrics and data types in the extracted MR dataset.

MR Extraction tool was designed by Julien Legault, as the other student and intern involved in this research project. The tool was extended and maintained by members involved in this research project including Julien Legault and Seyedbehnam Mashari. This tool as an OSS project ("ETS DEVOPS / Gitlab-Mr-Extraction," n.d.) created and maintained in GitLab, enables data extraction from Git repositories using API calls.

Metrics defined in the MR Extraction tool to measure important parameters of the code review process can be found in Table 2.2. Also, the description of metrics and the type of data that are obtained based on each of these metrics are described in this table. These data types include nominal, time, and count. We use count MR data for statistical analysis in chapter 3.

The MR dataset was obtained from the integration of the MR Extraction Tool into Kaloom's code super-repository. It contains MR information for the years 2020 and 2021 according to the defined metrics in the MR Extraction tool.

Table 2.2 Defined metrics in MR Extraction tool

Metric	Description	Type
State	Status of the MR which can be closed, merged, or open.	Nominal
Creation Date & Time	Time and date of MR creation.	Time
End Date & Time	Time and date of approving or closing MR.	Time
(Open) Sprint Day	The day when MR is created by the Developer.	Count
End Sprint Day	End Sprint Day is the day when MR is merged.	Count
Sprint	Each release at Kaloom consists of three sprints. This metric represents the Sprint number in which the MR is created.	Count
Number of comments after merge	The number of comments made by reviewers after that an MR is merged.	Count
Number of discussions	The number of comments made by both Reviewers and Developers, system notes, MRs, issues, and commits.	Count
Mean Time between commits	The average time between commits in the MR.	Count
Commenters	Name of commenters	Nominal
Committers	Name of committers	Count
Number of commits	Total number of commits by Developer in an MR	Count
Assigned reviewer	Name of the assigned reviewer	Nominal
True Reviewer	The reviewer with the highest number of comments on MR	Nominal
Number of initial files	Number of files upon MR opening	Count
Number of final files	Number of files after different code review iterations	Count
Rework commits	Number of commits after MR is merged	Count
Lead Time	Amount of time between the creation time of MR to its approval or closing time.	Count
Time to first interaction	The amount of time from the creation of MR until the first comment by a reviewer. If the MR is merged without comment(s), or with comment(s) after the merge, Lead Time is the time of first interaction by a reviewer.	Count
MR Size	The absolute number of insertions and deletions of code lines	Count

CHAPTER 3

MR DATA ANALYSIS

The goal of this chapter is to discuss the results of our empirical investigations of the MRs reviewed in the software delivery process at Kaloom. It includes statistical analysis of different metrics in the MR data set that are obtained by the MR Extraction tool.

We discuss how Lead Time and Size change over 21 days of sprints using statistical analysis of the MR data (section 3.1). It includes investigating the relation of Lead Time, Size, and Sprint Day in four studied groups at Kaloom. Then, we investigate behavior practices of developers and reviewers during different sprint days by conducting a set of statistical analysis (section 3.2). Finally, we analyze whether there are any exceptional usages in the MRs mechanism (section 3.3).

In particular, we discuss our results to answer the following research questions:

- **RQ1: How does the Lead Time of MRs change over the 21 days of a sprint?**

  We use statistical analysis to explore how MR Lead Time and Size change over time (section 3.1). According to our conducted analysis, firstly, we observe a very weak correlation between the Lead Time and Sprint Day of MRs in different groups. Secondly, we observe a very weak correlation between the Lead Time and Size of MRs in the studied groups. Finally, MRs of groups with hardware tests or those without automatic static code analysis take a longer time to be reviewed (longer Lead Time).

- **RQ2: What are the behavior practices of software engineers during the sprint days?**

We use a range of statistical analysis to explore behavior practices of software engineers during different days of sprints (section 3.2). These analysis include identifying the Size, Lead Time, the amount of created MRs over the sprint days, and the amount of reviewed MRs over the 21 days of a sprint.

Our results show that developers create larger MRs on Sunday of the second weeks in sprints. This can signal a specific behavior practice in developers to merge large code changes before the start of last week in the sprints. Moreover, we observe that reviewers inspect a larger number of MRs before weekends of the second week in the sprints. Finally, we observe that as the number of assigned reviews to highly engaged reviewers increase, the number of slow MRs assigned to that specific reviewer grows.

- **RQ3: What are exceptions and outliers in MR data?**

We start by analyzing and categorizing different types of exceptions and outliers. For this, we carry out manual and quantitative analysis to uncover exceptions and outliers in MR data (section 3.3). We categorize and justify exceptions of MRs. First, we observe that as a category of exceptions some MRs are commented after the merge. We found that it is because of bugs that raise after the merge of a change. Such number of MRs is not negligeable. These MRs can occur at any time of the sprint. We also detected outliers in MR data by Isolation Forest method. However, we noticed no significant information from investigation of data outliers.

For each of the research questions, we will discuss motivation, approach, and results. The observation in the results section is based on discussions with Kaloom's managers. These are initial observations that will require further investigations.

3.1 RQ1. How does the Lead Time of MRs change over the 21 days of sprints?

3.1.1 Motivation

The goal of this research question is to investigate how the Lead Time changes over the 21 days of sprints in different groups. Understanding how these metrics evolve over the days of sprints will help managers to improve different aspects of the review process such as those related to scheduling developers' and reviewers' tasks. Moreover, we investigate the correlation of different metrics with MR Lead Time to uncover the influential parameters on the merge speed. For instance, we study whether there is a correlation between the Lead Time, Size, and Sprint Day of code changes similar to the findings of prior work (Moreira Soares, de Lima Júnior, Murta, & Plastino, 2021). Therefore, managers have a better vision about whether the code has an impact on the MR lead time, so they have mechanisms in place to better control the code to be merged. In this research question, we also compare the metrics of different groups.

3.1.2 Approach

To investigate how the lead time changes over the days of sprints, we identify for each sprint day which MRs were merged and how long it took to review and merge them (i.e., their lead time). We also statistically compared whether there is any relation between the lead time of our studied MRs and the size of their respective code changes by leveraging Spearman correlation.

3.1.3 Results

Finding 1: We observe that the group with no static source code analysis has a higher average MR Lead Time. We observe an average lead time of 71, 89, 44, and 77 hours for the CP, DP, MP, and PF groups, as shown in Figure 3.1. The DP and PF groups have generally higher average MR Lead Times compared to other groups, as shown in the same Figure. The

average MR Lead Time in the DP group is 89 hours and the highest among all the studied groups. According to the information obtained from the discussion with Kaloom's executives, the DP and DF groups should pass hardware tests as a part of their development process, which make the lead time larger. These tests are usually more time-consuming than regular component tests. The DP group's project is dominantly developed with the P4 programming language, which lacks support of automatic static code analysis hence requires more human investigations. On top of that, MRs created in this group contain the lower-level routing code. Furthermore, the DP group has the largest average MR Size among the four studied groups.

While we observe that the group with the largest average lead time has the largest average MR size, we do not observe a pattern for the other groups, as shown in Figure 3.1. In fact, the group with the second largest average lead time has the smallest average MR size. Based on our information from the characteristics of each group at Kaloom, this group deals with lower-level routing codes. Because of the low abstraction of low-level programming languages, MRs containing low-level code are usually larger. We further discuss a statistical correlation between the merge size and the lead time in finding 3.

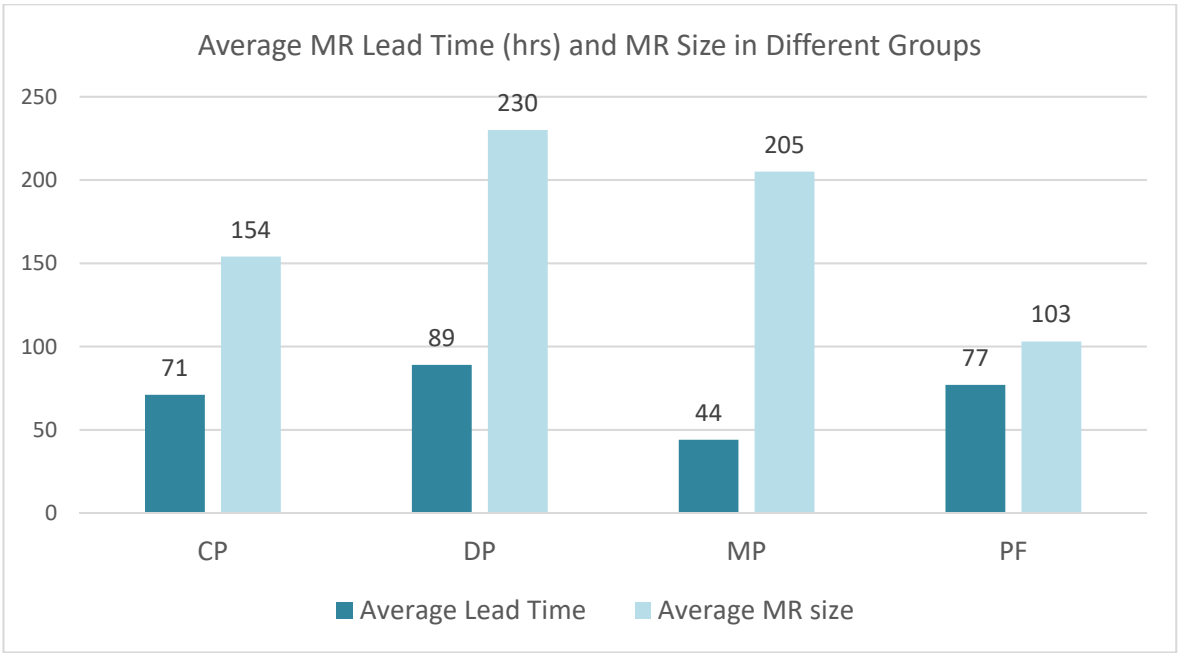

Figure 3.1 Average Lead Time and average MR Size in four studied groups

Finding 2: We observed a very weak Spearman correlation between the Lead Time of a MR and the day on which an MR was created as well as the day on which an MR was closed. The absolute value of the Spearman correlation between Lead Time and Sprint Day is 0.0017, -0.025, -0.041, and -0.02 for CP, DP, MP, and PF groups, respectively. Similarly, we observe a very weak Spearman correlation between the MR lead time and the day on which the MRs were closed. Such a Spearman correlation is -0.04, 0.012, 0.019, and 0.045 for the same last groups. Note that a Spearman correlation that ranges between zero and 0.19 is considered a very weak correlation. Although the CP and DP groups tend to have a high average Lead Time in the first weeks of sprints, as shown in Figure 3.6, that average is high due to some MRs that have a high Lead Time. For example, we observed merges that took more than a month (35 days) in these two groups. As we can observe in Figure 3.6, in the CP and DP groups, the average Lead Time of the MRs in the first week of the sprints is notably higher than that in the second and third week of sprints. In contrast, in the MP and PF groups, the average Lead Time of MRs in the first week of sprints is smaller than that in the second and third weeks. This pattern is more noticeable in the PF group. Finally, we observe that developers create a low number of MRs during the weekends (Figures 3.8, 3.9, 3.11, and 3.12). These MRs are mostly smaller than the MRs created during business weekdays.

Similarly, the Lead Time is not correlated to any of our studied metrics, as shown in Figures 3.2 to 3.5. These Figures depict the obtained correlation heatmap based on the Spearman correlation method in different groups. As shown in the Figures, there is no correlation between the Lead Time and the number of commenters, the number of commits, the number of discussions, the Size of the MRs including its number of changed files. We also interestingly observe that even the number of iterations over a MR (shown by the Rework commits) is not correlated with the lead time. Our Figures also show more details about the correlation between the other metrics.

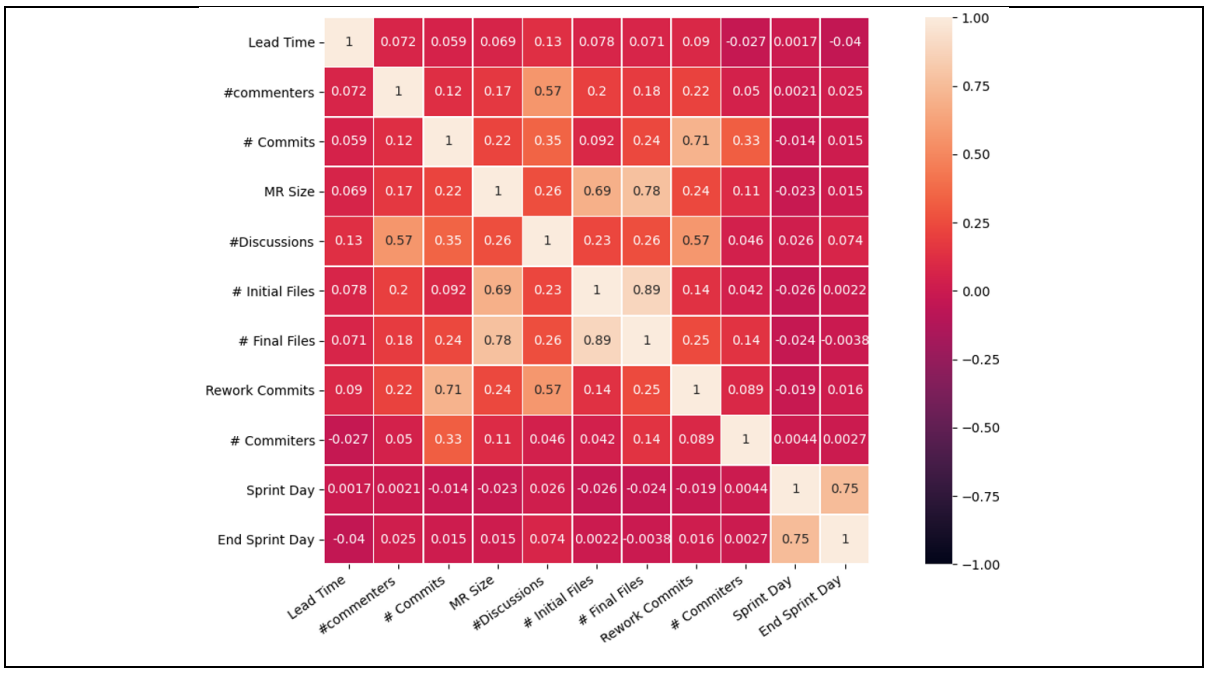

Figure 3.2 Correlation of metrics in the CP group

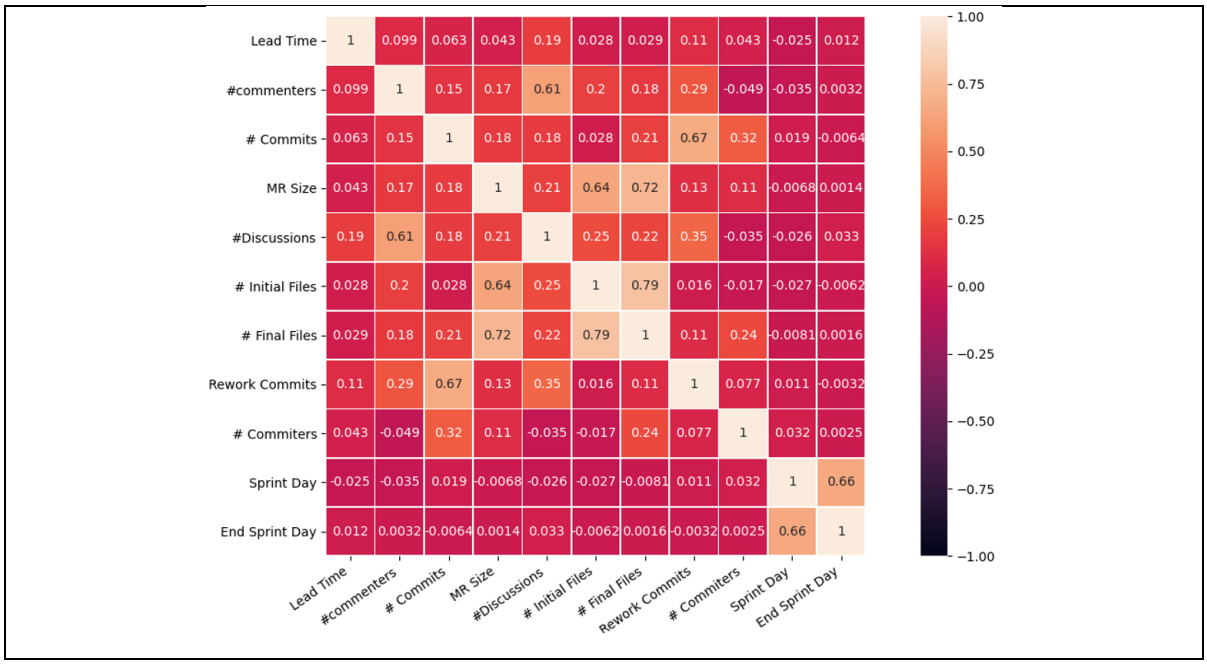

Figure 3.3 Correlation of metrics in the DP group

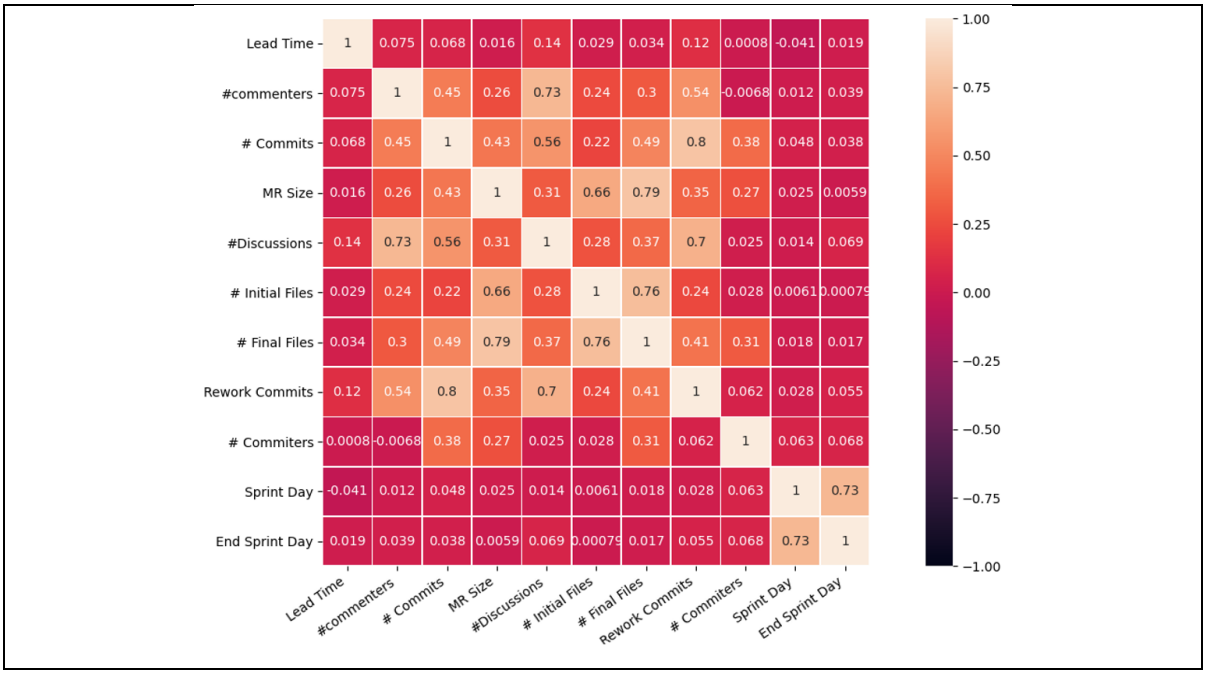

Figure 3.4 Correlation of metrics in the MP group

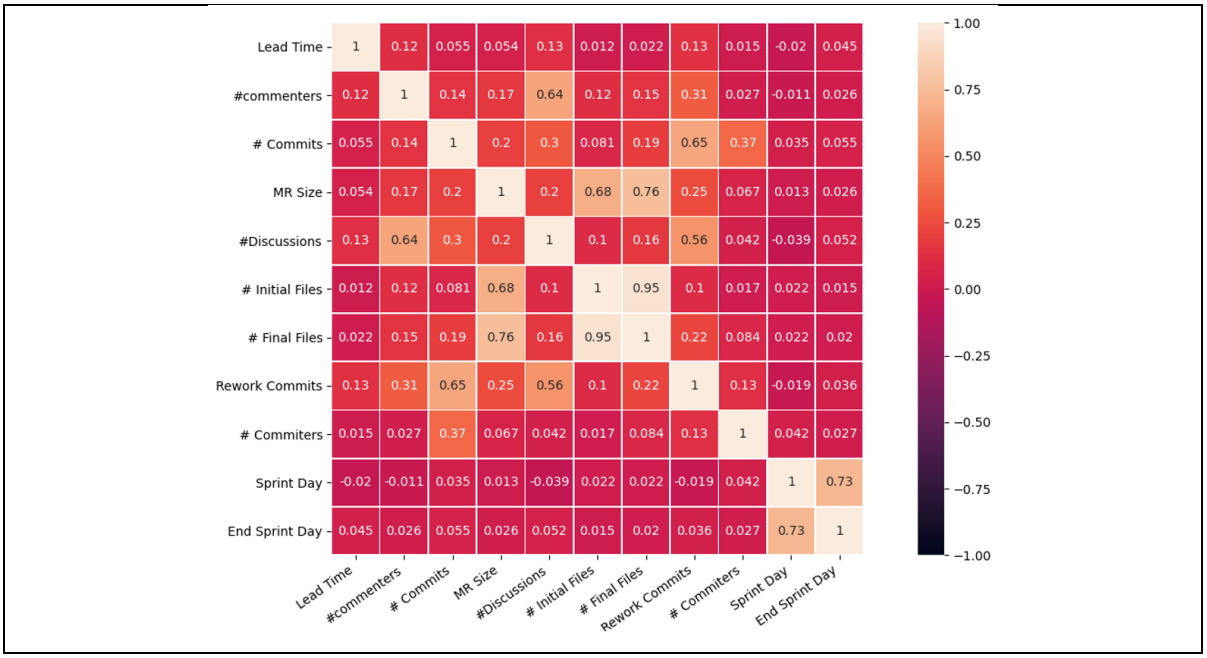

Figure 3.5 Correlation of metrics in the PF group

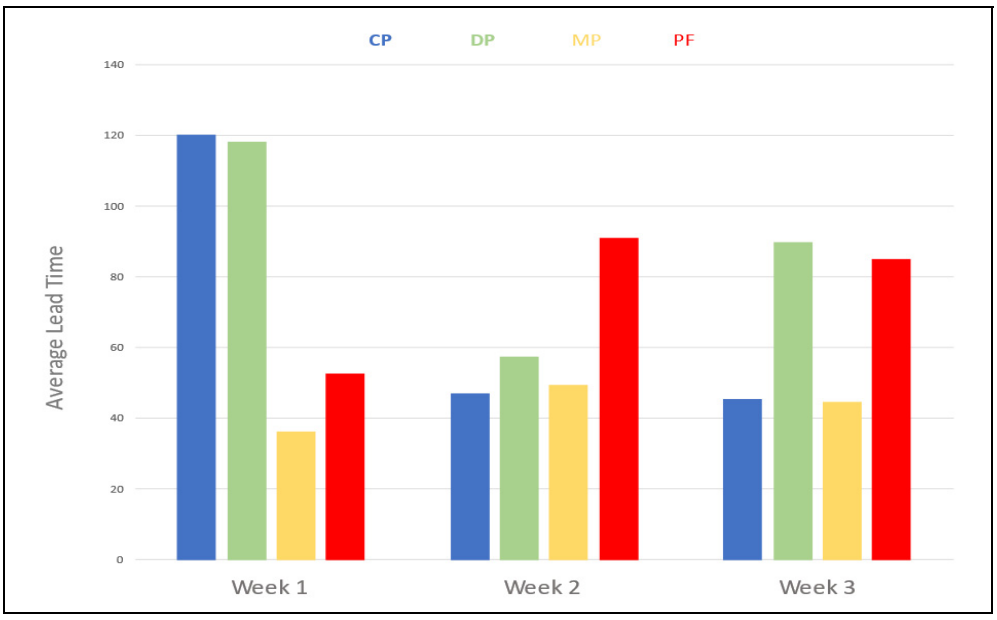

Figure 3.6 The average Lead Time (in hours) of MRs during the three weeks of the sprints in all groups. Blue, green, yellow, and red colors represent data in the CP, DP, MP, and PF groups, respectively. We consider this color mapping convention in the rest of the figures and tables in this thesis

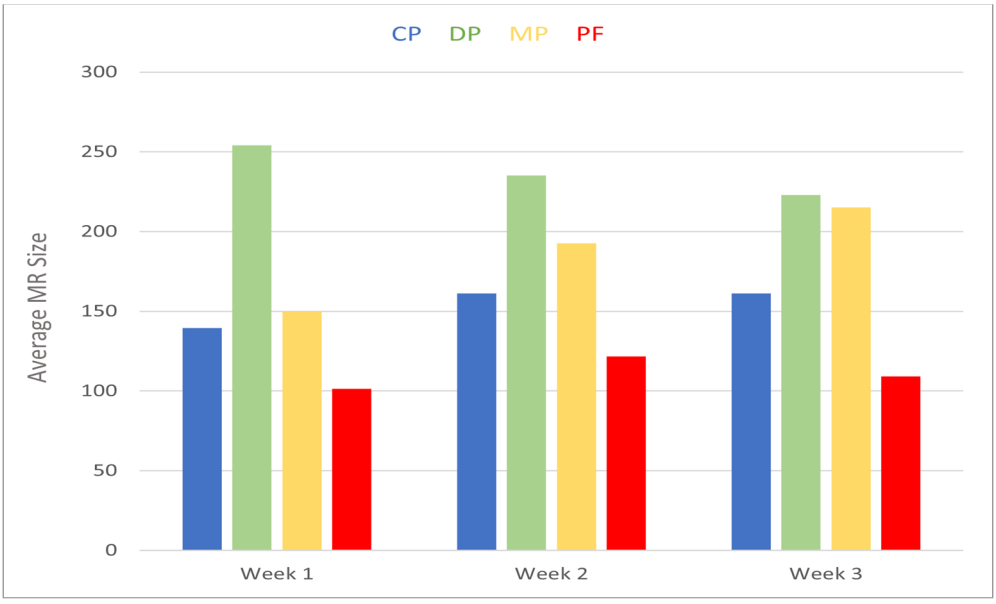

Figure 3.7 The average Size of MRs in three weeks of sprints in four studied groups

Finding 3: While prior work reported a relation between the Lead Time and the Size of the code change to merge, we do not observe any correlation between these two metrics. Spearman's correlation between these two metrics is between 0.01 and 0.07 for the four studied groups at Kaloom (Figures 3.2 to 3.5). Figure 3.7 shows the average MR Size in the three weeks of sprints in different groups. As we can observe in the Figure, the average Size of MRs in the DP group decreases over the three weeks of sprints. In contrast, the average Size of MRs in the MP group increases during the same period. Moreover, we observe no remarkable change in the average Size of MRs in three weeks of sprints in the CP and PF groups.

Furthermore, we observe no relation between the Lead Time and Size of MRs based on the distribution of data over different days of the sprints in different groups, as shown in Figures 3.8, 3.9, 3.10, and 3.11 for each of our studied groups. These Figures show the relation between the lead time, the MR size, and the Sprint Day. In these figures, the x-axis is Sprint Day, the y-axis is Lead Time (in minutes), and the size of the circles represents the MR Size. The larger the size is, the larger the circle is. Due to the high concentration of MRs with a short Lead Time near the horizontal axis, we used a zoomed view to depict the distribution of samples over different days of sprints.

As a general trend of the studied groups, we observe no relationship between the Size and Lead Time of MRs in none of the four groups. Furthermore, developers create a low number of MRs during the weekends. These MRs are mostly smaller than the MRs created during business weekdays.

As we can observe in Figure 3.8, developers in the CP group create larger MRs in the second and third weeks of the sprints than in the first week. It verifies our previous observation (Figure 3.6) where the average Size of MRs in the second and third week of the sprints in this group is larger than that in the first week of sprints. We observe similar results about the relation between the lead time, size and sprint days for the other groups, as shown in Figures 3.9, 3.10, and 3.11.

Comparing the PF group (Figure 3.11) with the other groups (Figures 3.8, 3.9, and 3.10) shows that the created MRs in the PF group are smaller than those in other groups. Furthermore, the high number of small Size MRs with high Lead Time in the PF group shows that MRs created in this group take a long time to be reviewed and tested despite their small Size. We think that this result is because that group has OS level config and deployment script which related changes might require longer time to be reviewed.

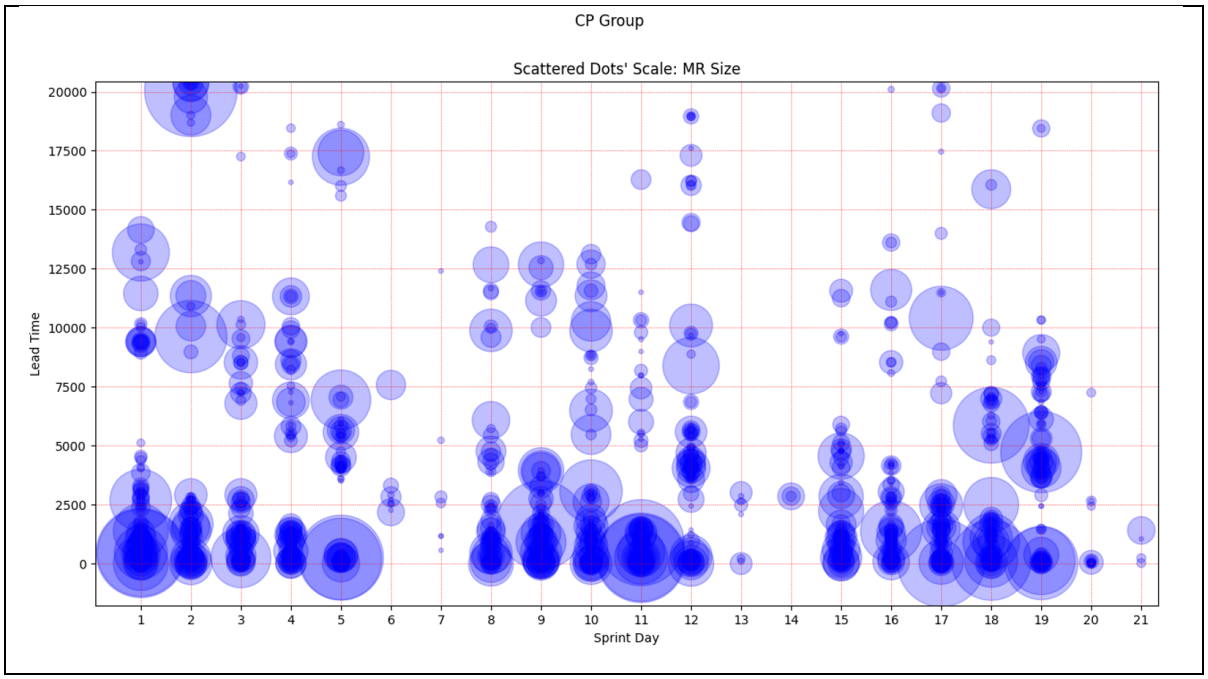

Figure 3.8. Distribution of MRs in the CP group based on the Lead Time and Size over different Sprint Days

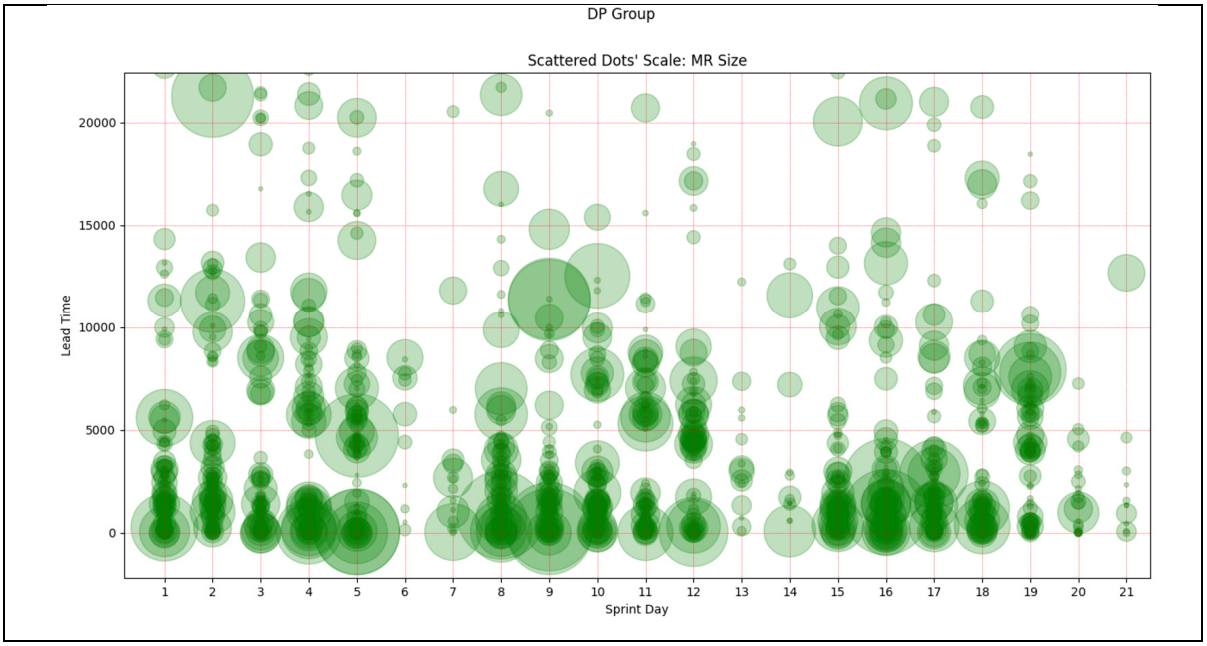

Figure 3.9 Distribution of MRs in the CP group based on the Lead Time and Size over different Sprint Days

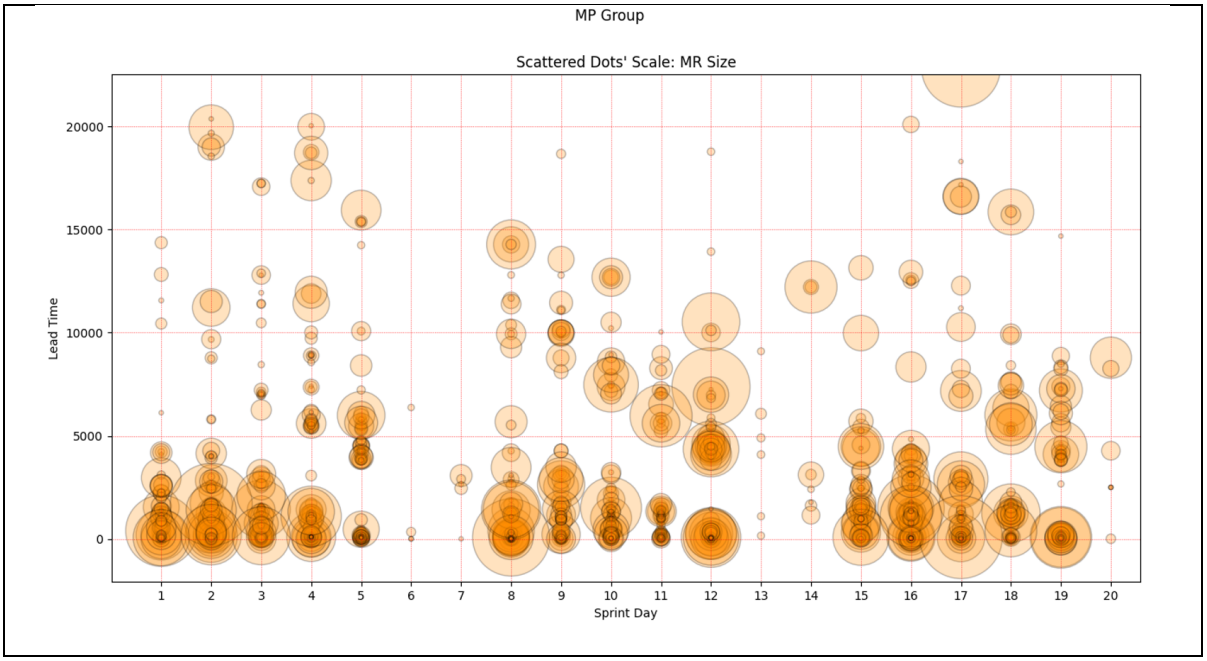

Figure 3.10 Distribution of MRs in the MP group based on the Lead Time and Size over different Sprint Days

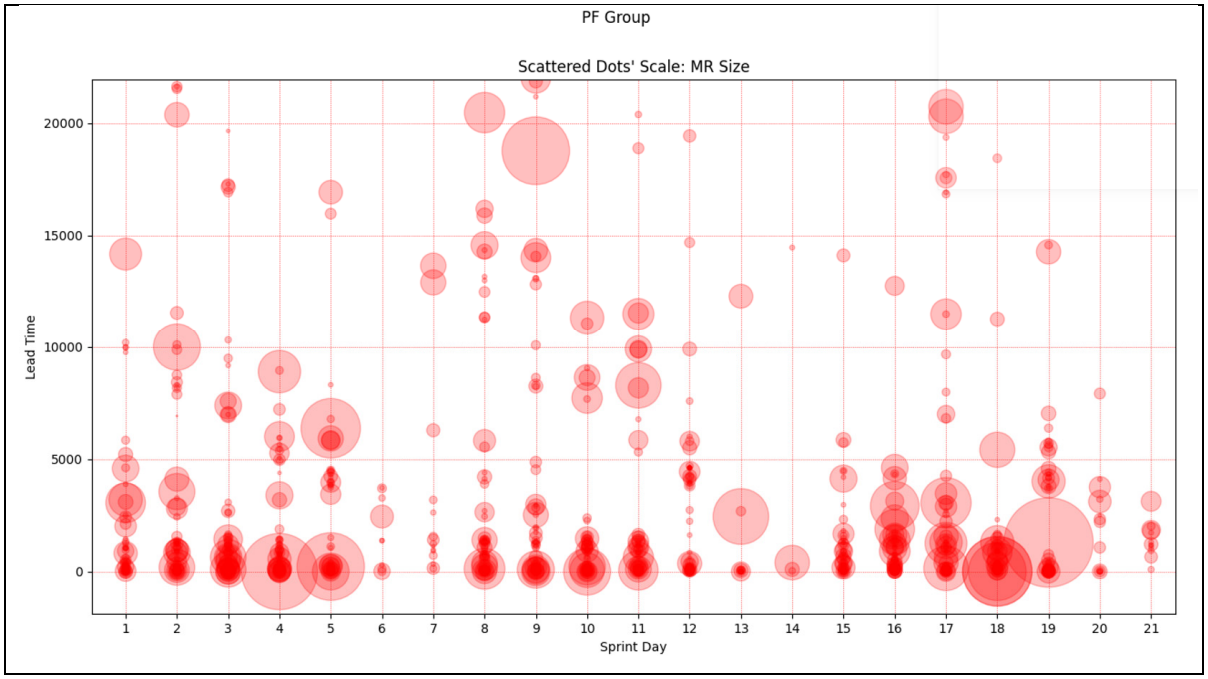

Figure 3.11 Distribution of MRs in the PF group based on the Lead Time and Size over different Sprint Days

3.2 RQ2. What are the behavior practices of software engineers during the sprint days?

3.1.4 Motivation

Behavior practices of software developers and reviewers during the code development and review process can affect the quality and time of the release. Thus, in this research question, we investigate the number of MRs and their sizes over the 21 days of a sprint, as well as the number of MRs that reviewers consider over the 21 days of a sprint. This analysis can allow managers to compare behavior practices of developers and reviewers over different days and weeks of sprints. Therefore, it helps them improve their development and review workflows of the software delivery process.

3.1.5 Approach

To explore developers' behaviour, we investigate the number and average Size of MRs created in each Open Sprint Day, which is the day in the sprints when the MR is created.

To study reviewers' behaviour, we investigate on which days reviewers close the MRs. We conduct statistical analysis to investigate how reviewers can affect MR Lead Time. We explore how the concentration of review tasks on some key personnel can cause delays in the review process of MRs. We analyze it by investigating the number of slowly reviewed MRs evaluated by highly engaged reviewers. We define Slow MRs as the top 20 percent of MRs in every group with the longest Lead Time. Regarding the ethics in research, personnel identities are kept confidential by renaming them. We use a different naming convention for the shared reviewers between different groups (we added "C" before indicating the reviewer number).

3.1.6 Results

Finding 1 – We observe that in all studied groups, developers create the largest MRs on the last day of the second week in sprints, while on the same day a lower number of MRs compared to the other days. Developers create 3, 13, 9, and 4 MRs, while these MRs have an average Size of 250.3, 845.9, 336, and 228.3 on the 14th day (Sunday of the second week) in the sprint for the CP, DP, MP, and PF groups respectively. Further results are shown in Figures 3.12, 3.13, 3.14, and 3.15 for the CP, DP, MP, and PF groups. Such a pattern might have occurred since developers request merges that are containing vast code changes right before the last week of the sprints. These extensive code changes may be related to merging a branch that developers worked on for a long time. We also observed that the MP group (Figure 3.14) and PF group (Figure 3.15) have large MR in the last week, which can be explained as developers who request to merge vast code changes to meet the defined targets in the sprints.

From Figures 3.12 to 3.15, we do not observe any clear pattern for the number of MRs created by day. We observe for the three groups that the number of opened MRs decreases over the

last four days of sprints. We also observe that as a general trend among all groups, developers create a low number of MRs at the weekends.

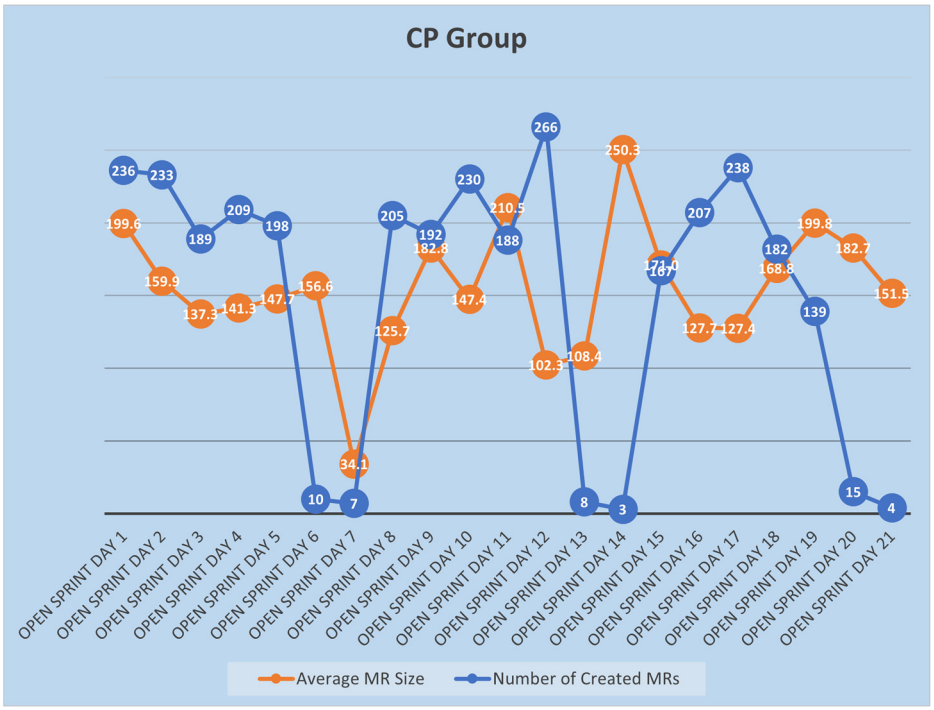

Figure 3.12 Number and average Size of MRs during different Sprint Days in the CP group

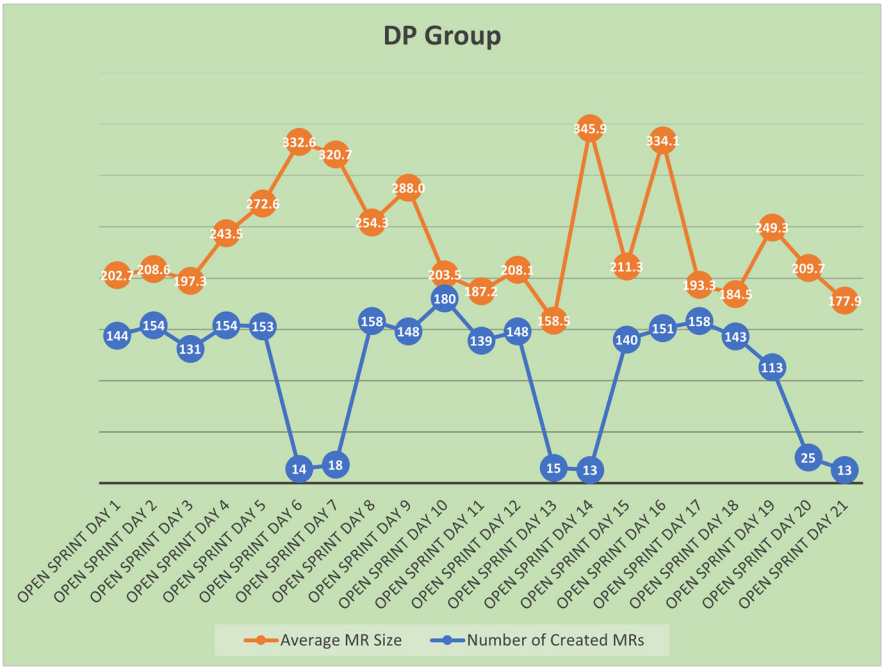

Figure 3.13 Number and average Size of MRs during different Sprint Days in the DP group

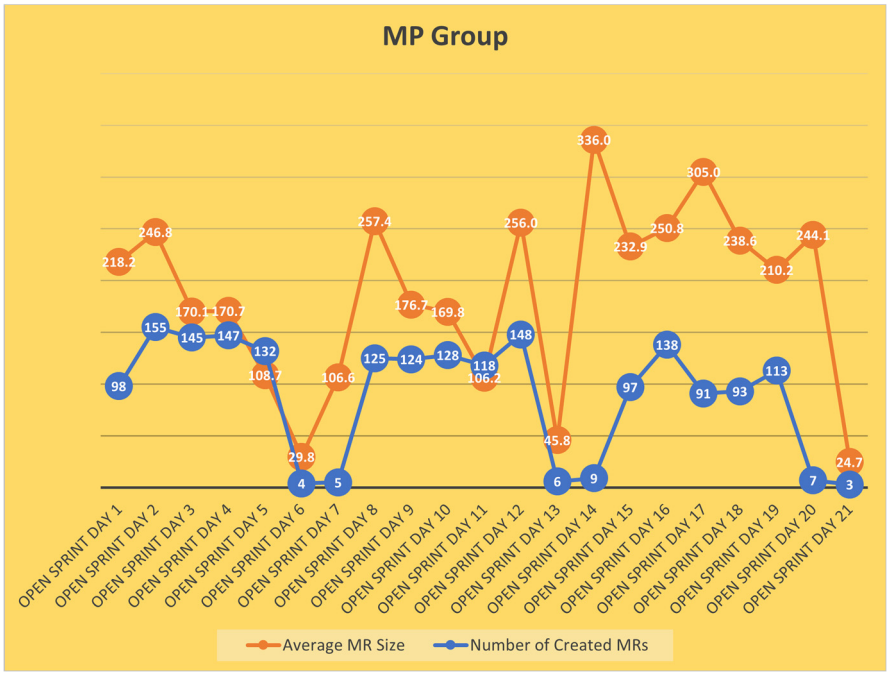

Figure 3.14 Number and average Size of MRs during different Sprint Days in the MP group

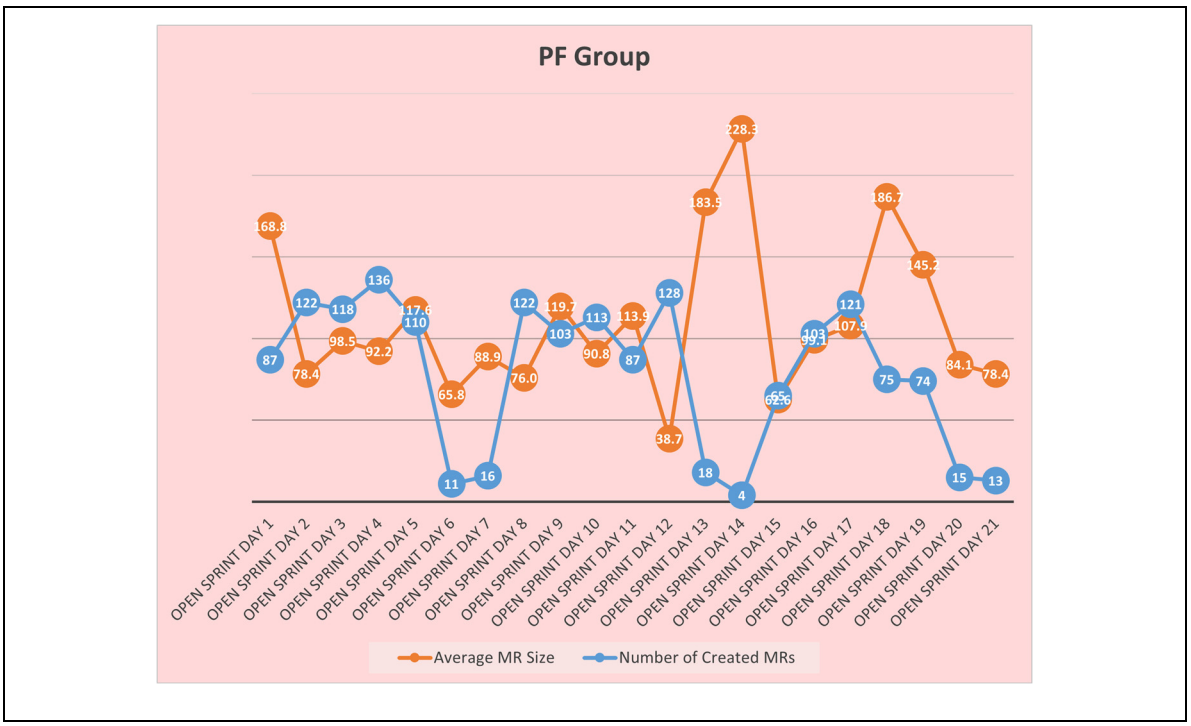

Figure 3.15 Number and average Size of MRs during different Sprint Days in the PF group

Finding 2 – We observe that reviewers inspect the highest number of MRs in the middle of the sprints and before weekends of the second week. Reviewers in the CP and MP groups reviewed 242 and 161 MRs on the 12th day of the sprints. This amount is quite larger than the average number of reviewed MRs in these two groups (148 and 89, respectively). Moreover, reviewers in the DP group reviewed 199 MRs on the 11th day of the sprints which is considerably higher than the number of MRs reviewed in other sprint days. This trend can be associated with a specific behavior practice in reviewers in these three groups where they review a high number of MRs before the start of last week in the sprints.

Figures 3.16 to 3.19 shows the number and average Lead Time of MRs reviewed during End Sprint Days with their average values in each group. These figures show the total number of reviewed MRs, and the average Lead Time of MRs in the CP group. The unbroken blue line represents the number of MRs reviewed in each End Sprint Day. The unbroken orange line shows the average Lead Time of reviewed MRs (in hours) during End Sprint Days. Two dotted

lines in blue and orange represent the average number and Lead Time (in hours) of MRs reviewed during all sprint days, respectively.

Figures 3.16 to 3.19 convey that the reviewers of different groups – like developers - evaluate a small number of MRs at the weekends. Except for the PF group, the highest number of MRs are reviewed in the middle of the sprints and before weekends of the second week in sprints (12^{th} day in CP and MP groups and 11^{th} day in DP group). Generally, the average number of MRs reviewed in the CP group is considerably higher than in other groups. The average Lead Time of MRs in the MP group is noticeably the lowest among all groups.

In the CP and DP groups, reviewers evaluate the highest number of MRs on the 11^{th} day of sprints respectively. The average Lead Time of MRs reviewed on Sunday of the first week of sprints is larger than that on other sprint days in the CP group. According to the observation of MR cases in the working repository, some of the long-lasting MRs from the previous sprints cause this high average Lead Time on the first weekend of sprints. This trend might be associated with a specific behavior practice where reviewers tend to decide on long-lasting unreviewed MRs from the previous sprint at the first weekend of sprints.

In the MP group, reviewers inspect the highest number of MRs on the 12^{th} day of the sprints. We can observe some spikes in the average Lead Time, especially in the middle of the sprints. For example, the average Lead Time of the reviewed MRs on the 8^{th}, 11^{th}, and 15^{th} days of the sprints is notably larger than the average Lead Time in the entire sprint days.

In the PF group, reviewers examine the highest number of MRs on the 8th day of sprints. As we can see in the Figure, the average Lead Time of the MRs reviewed in the second and third weeks of the sprints is larger than that in the first week.

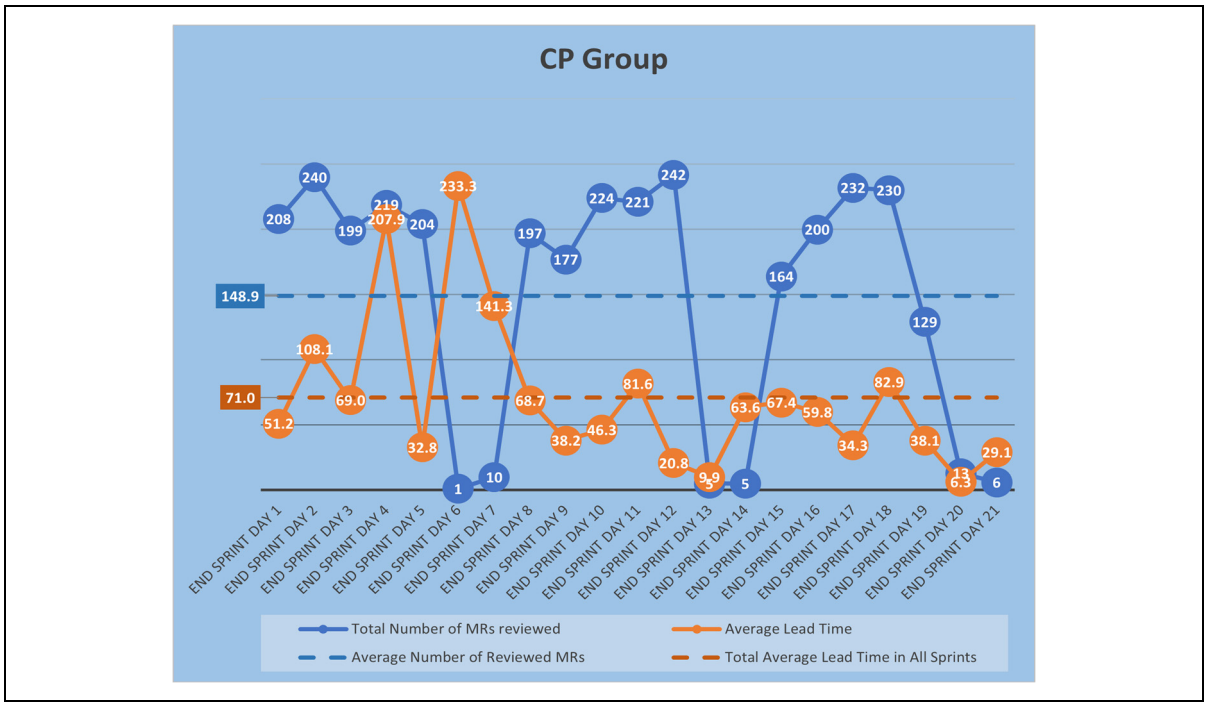

Figure 3.16 Total number of reviewed MRs and average Lead Time of MRs in CP group

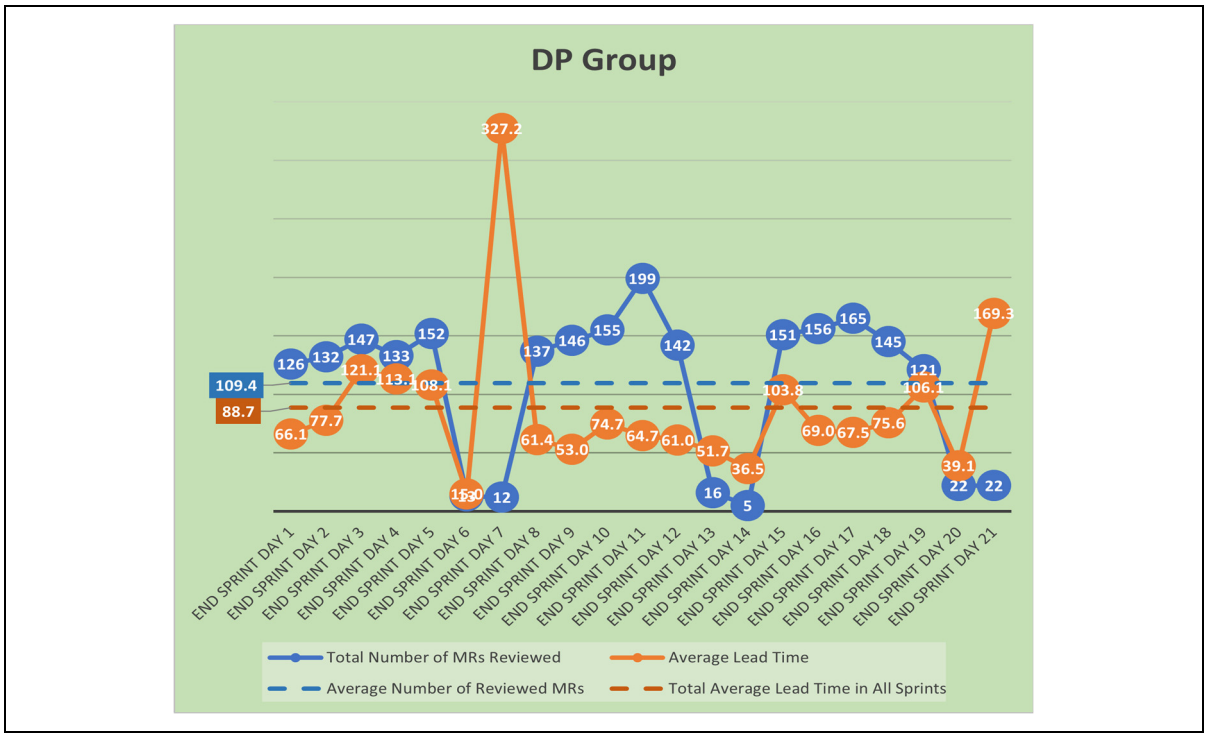

Figure 3.17 Total number of reviewed MRs and average Lead Time of MRs in DP group

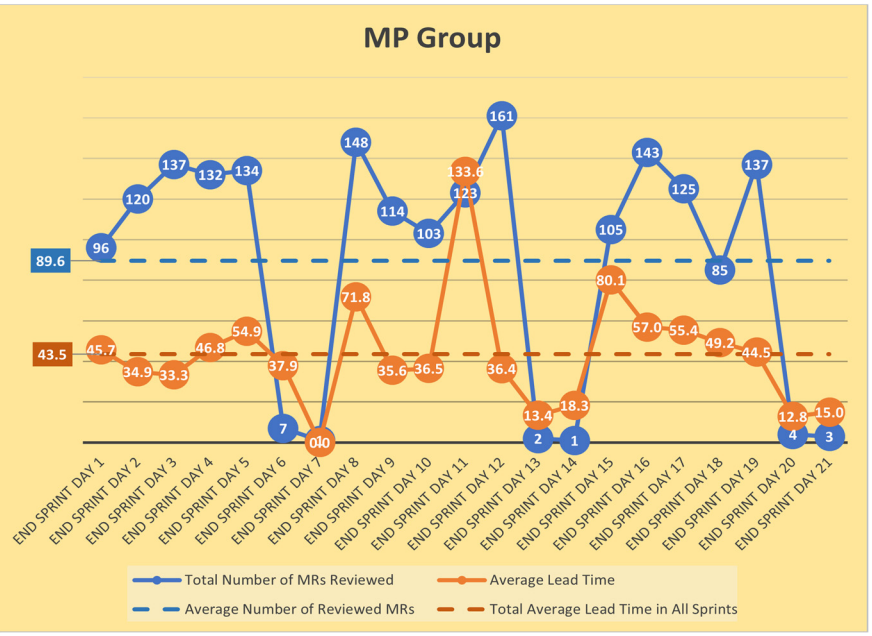

Figure 3.18 Total number of reviewed MRs and average Lead Time of MRs in MP group

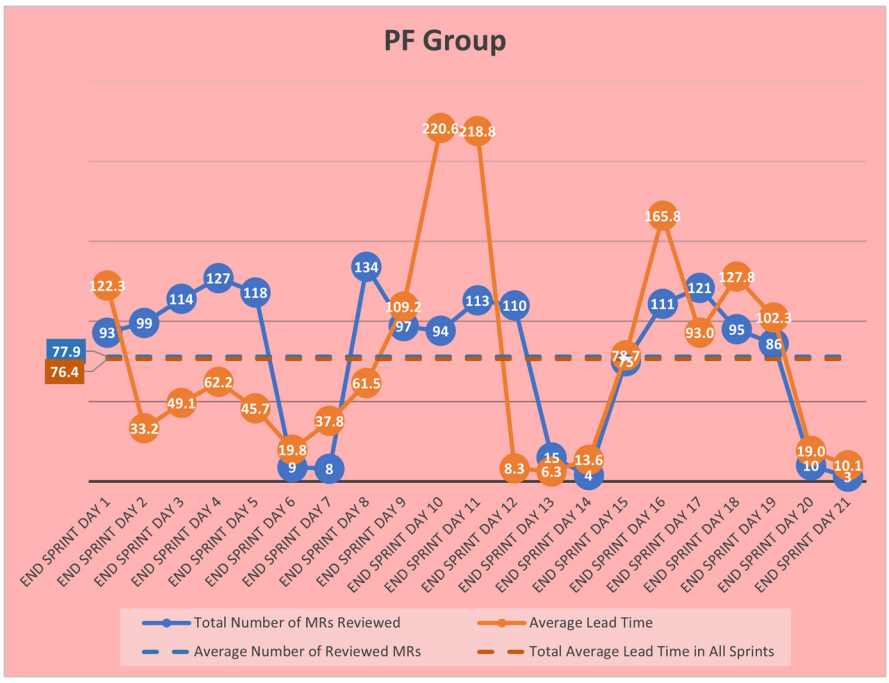

Figure 3.19 Total number of reviewed MRs and average Lead Time of MRs in PF group

Finding 3 – We observe that number of assigned review tasks to highly engaged reviewers can affect MR Lead Time. As the number of assigned MRs to reviewers grows, the number of slow MRs rises. We can conclude that the high number of assigned MRs to reviewers can increase the MR Lead Time. It can be because when the number of review tasks to reviewers increases, MRs should wait a longer period in the backlog to become reviewed. Furthermore, we observe that roughly 20 percent of unassigned MRs to specific reviewers are slow ones.

According to Figures 3.20 to 3.23, a considerable number of MRs are unassigned to specific reviewers in each group. About 20 percent of these unassigned MRs are merged slowly in the review process. It might be interpretable that when MRs are not assigned to a particular reviewer, reviewers themselves do not shoulder the inspection responsibility. Therefore, unassigned MRs remain unreviewed for a longer interval. A good practice to avoid this could be to assign the MR reviewer when creating them.

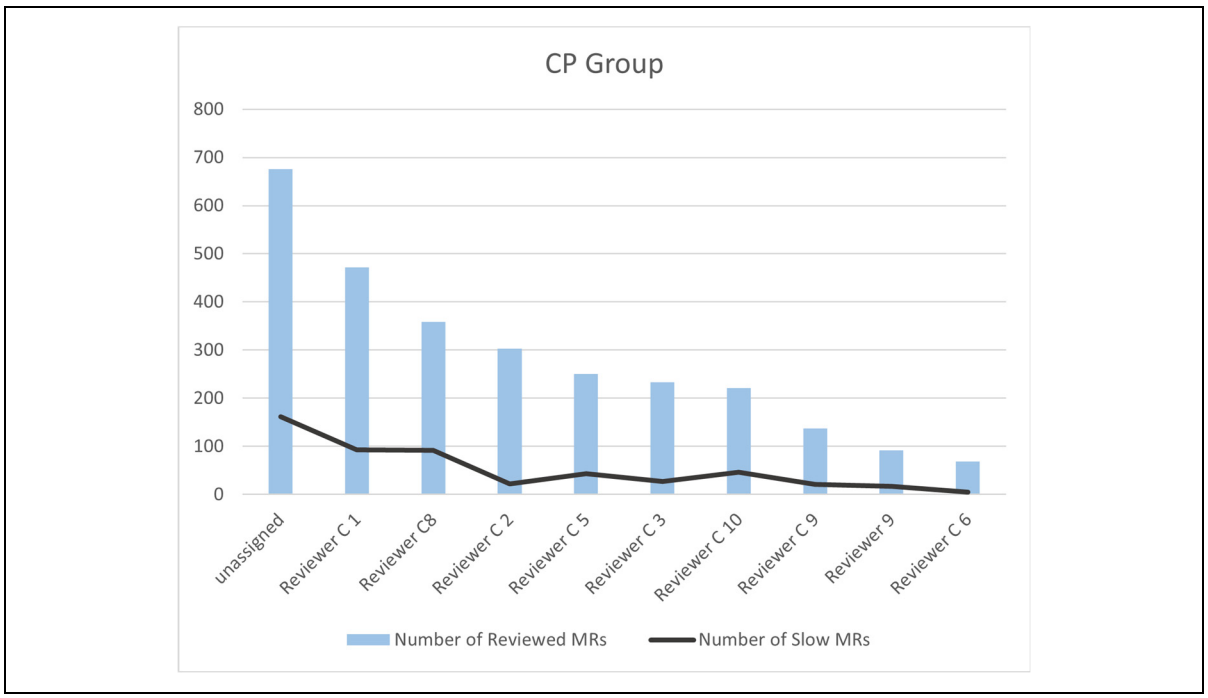

Figure 3.20 Number of MRs assigned to highly involved reviewers and the number of slowly inspected MRs by them in the CP group

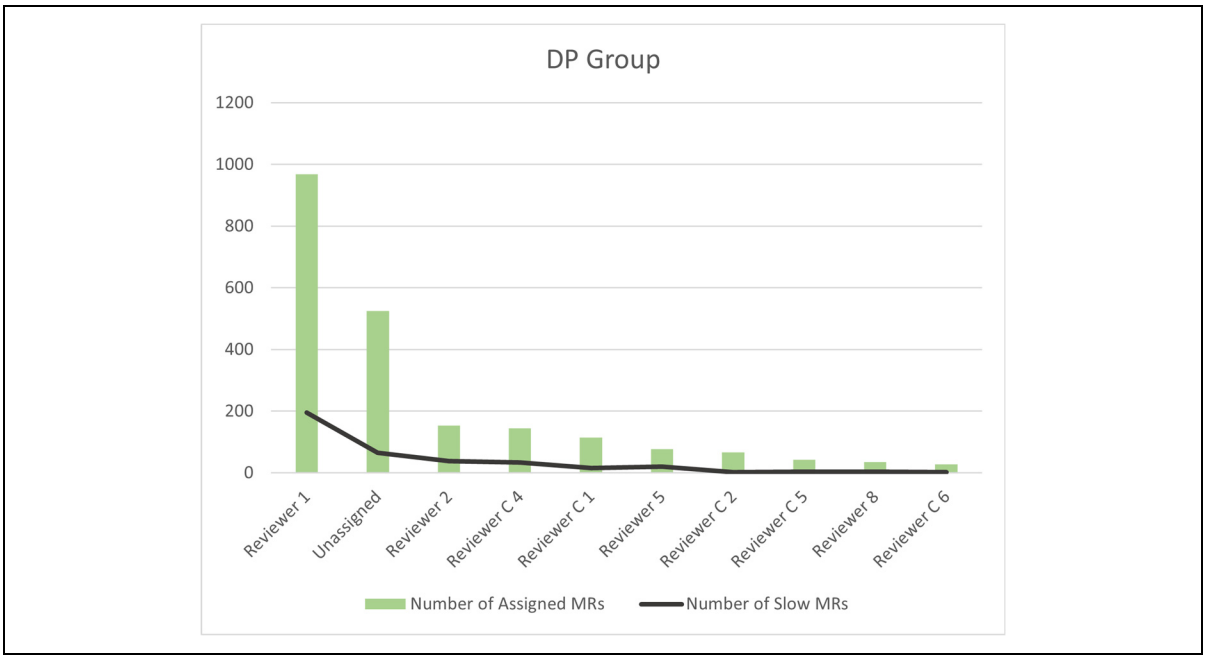

Figure 3.21 Number of MRs assigned to highly involved reviewers and the number of slowly inspected MRs by them in the DP group

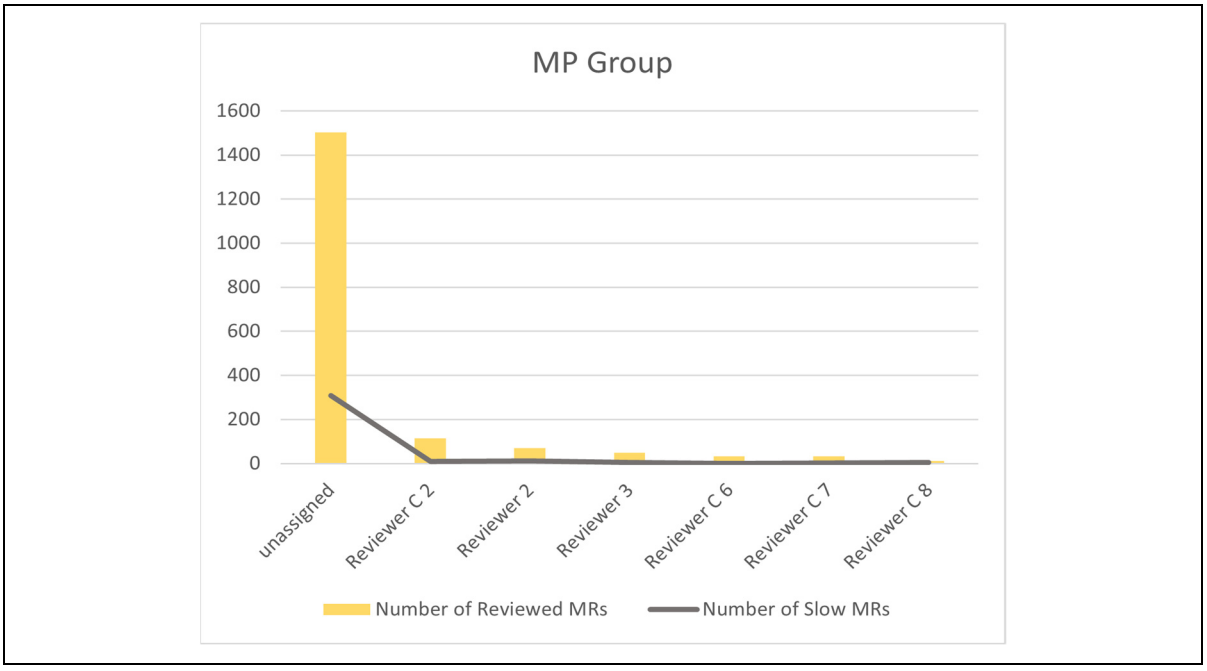

Figure 3.22 Number of MRs assigned to highly involved reviewers and the number of slowly inspected MRs by them in the MP group

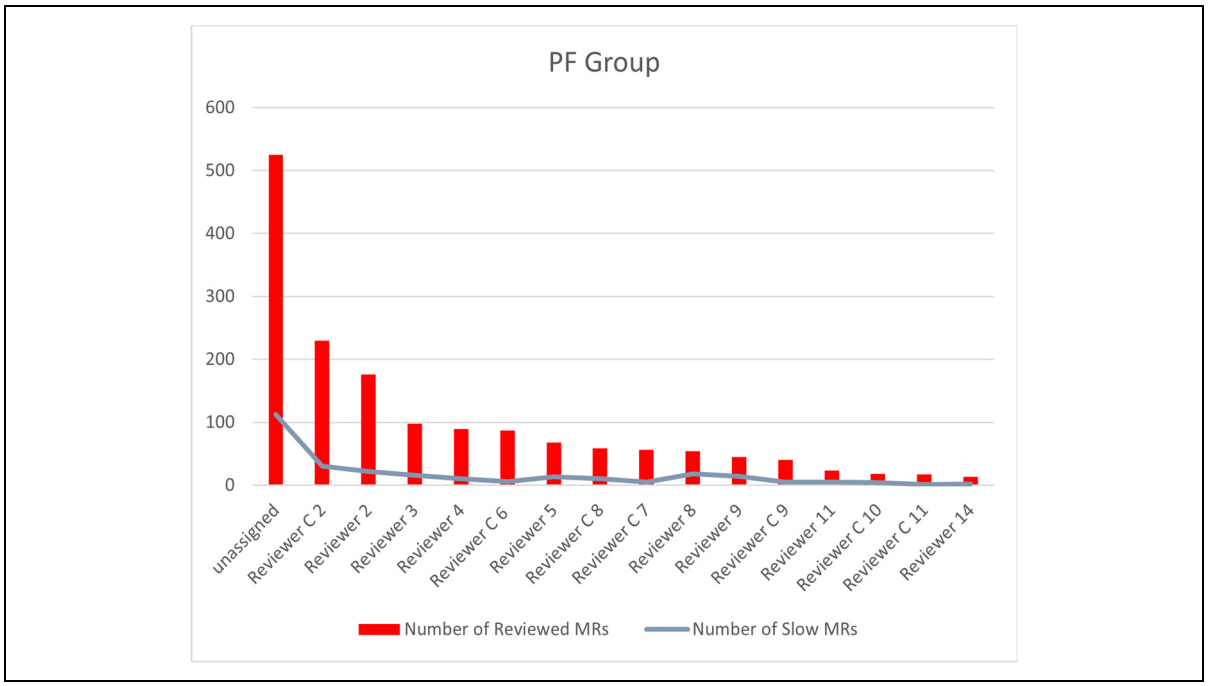

Figure 3.23 Number of MRs assigned to highly involved reviewers and the number of slowly inspected MRs by them in the PF group

According to Figures 3.20 to 3.23, in all groups except for the MP group, a notable proportion of assigned MRs to the highly engaged reviewers are slow ones. Moreover, the number of slow MRs increases as the number of assigned MRs to a reviewer grows. In other words, the busier an assigned reviewer, the longer time allocated MRs wait for an inspection.

For instance, a large number of MRs are assigned to Reviewer C2 who plays a principal role in reviewing MRs in most of the groups. The high workload of this reviewer might be the reason for the low pace of reviewed MRs by them. A solution for this undesired trend might be effectively managing toward a balanced workload between the highly engaged reviewers according to their routine review performance.

3.3 RQ3. What are exceptions and outliers in MR data?

3.1.7 Motivation

The goal of this section is to detect abnormal MRs in our dataset. Exploring the reason behind the occurrence of abnormal MRs can help companies detect, distinguish and document avoidable and unavoidable conditions in the review process of MRs.

3.1.8 Approach

To explore data anomalies in the MR dataset, we use a manual and a quantitative analysis:

In the manual analysis, we manually investigate data to observe exceptional cases of MRs in the code review process, such as large MRs that are merged very fast without exchanging comments between developers and reviewers.

In the quantitative analysis, we leverage Isolation Forest as an outlier detection algorithm to investigate whether there are categories of similar MRs, and which MRs are different (i.e., outliers) from the typical usage of MRs. We opt for Isolation Forest since we observe that data distribution in the MR data set is non-Gaussian and Isolation Forest does not rely on the assumptions related to data distribution.

3.1.9 Results

Finding 1: Our manual investigations show three exceptional usages of the MRs. We observe that developers can still comment on MRs after the merge. We also observe MRs with no code changes, and MRs with a short lead time even if some of them have a large size.

These exceptions with related justifications are collected and listed in Table 3.1.

Table 3.1 List of different categories of exceptions and their justification

Observation in dataset		Justification
Comment(s) after merge		• These are cases when an issue with the change is detected after it is merged. These issues include bugs with the MR discovered later in different QL level which ranges from QL1 within same day, or higher QL tests which can be a week later. In this case while troubleshooting the bug, the culprit is identified and commented on this change with the possible area of the code that caused the bug. • Reviewers want more clarification on the change after the change was merged to clarify and reason some areas of code change. • Some reviewers were busy with other tasks and commented out on the change after being merged. • A glitch/misfunction with automation bot account that commented after the MR was merged. • An MR branch is rebased when the commits have already been merged to the destination branch.
MR Size = 0		When working on an old feature branch, developers request to merge from master towards this existing branch whereas there might be nothing to be merged. However, they still proceed with merging this empty merge.
Fast merges with no comment	Large (Size) MRs	• Revert cases • Moving code files, libraries, or functions • Removing a chunk of code across all groups
	Small (Size) MRs	• Version update • Rolling out CICD build scripts and plugins

Finding 2: MRs with comments after the merge is more frequent in the DP and PF groups. We think that it is mainly because the detection and troubleshooting of bugs are more complex in routing and OS lower layers of these two groups. Therefore, a higher proportion of buggy MRs is not detected in the review process. However, we observe no pattern indicating that these buggy MRs have occurred on specific days of the sprints.

Figure 3.24 shows the number of MRs with at least one comment after the merge in the four studied groups.

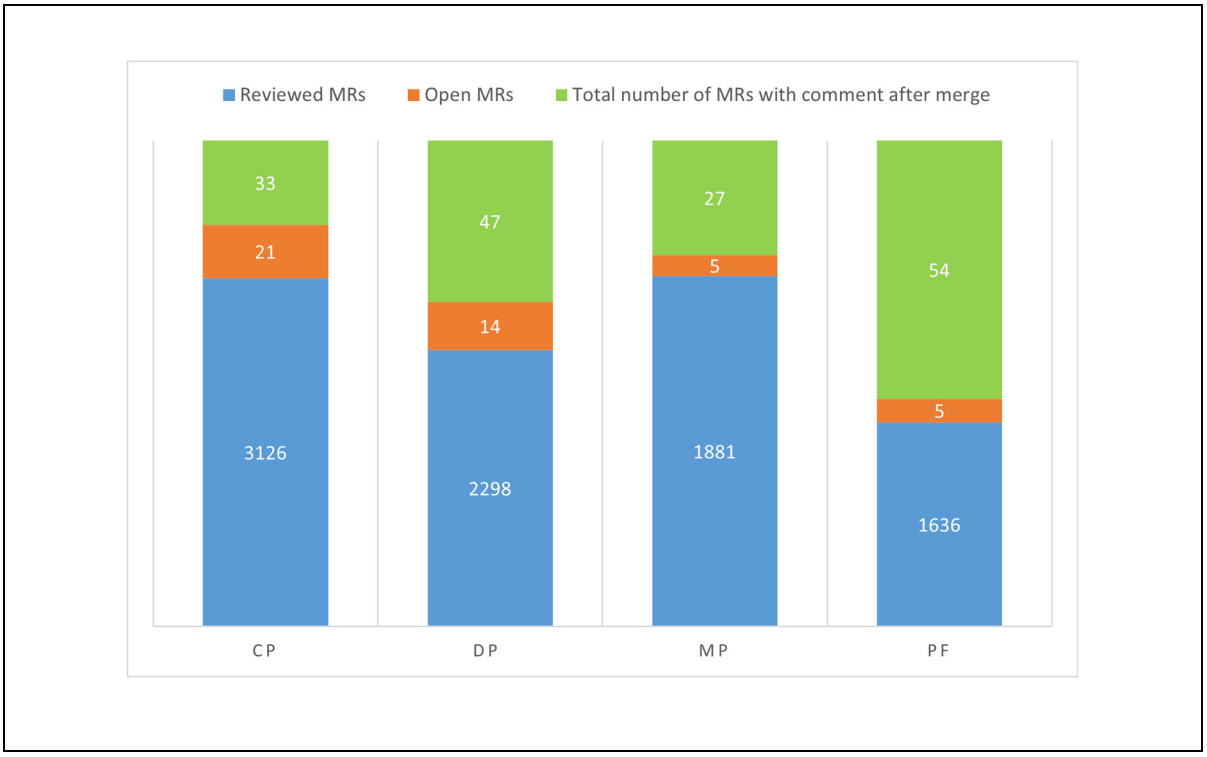

Figure 3.24 Number of MRs with comment(s) after merge in different groups

Figures 3.25, 3.26. 3.27, and 3.28 show the occurrence of MRs with comments after the merge in different groups. The PF group has the highest occurrence of this type of exception in its MRs. In these figures, exceptions are in red among the rest of the regular MRs shown in blue. As we can see in the Figure, exceptions are spread through different days of the sprints and are not specific to a specific sprint day.

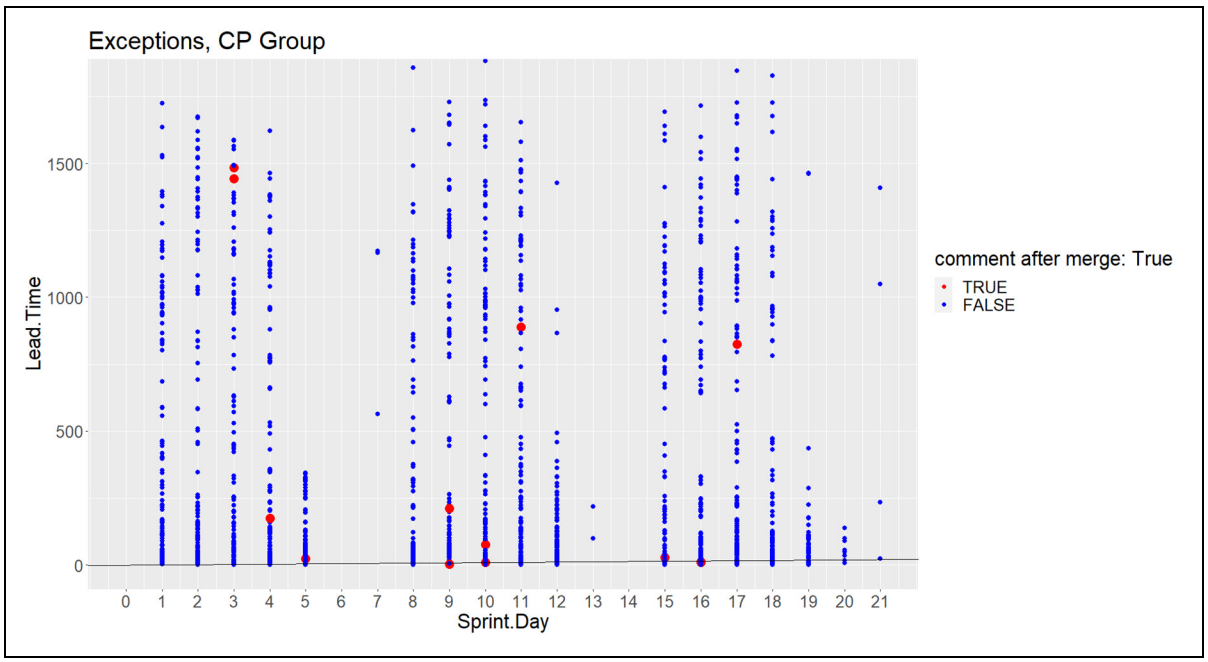

Figure 3.25 MRs with comment(s) after merge during different Sprint Days in CP group

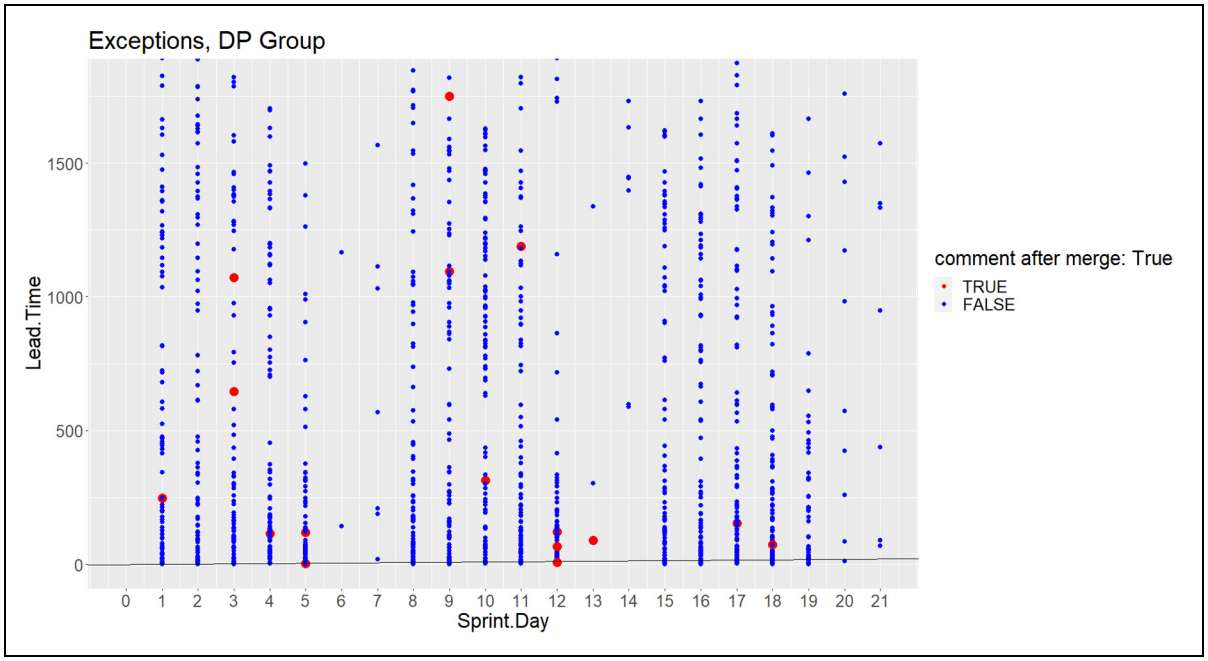

Figure 3.26 MRs with comment(s) after merge during different Sprint Days in DP group

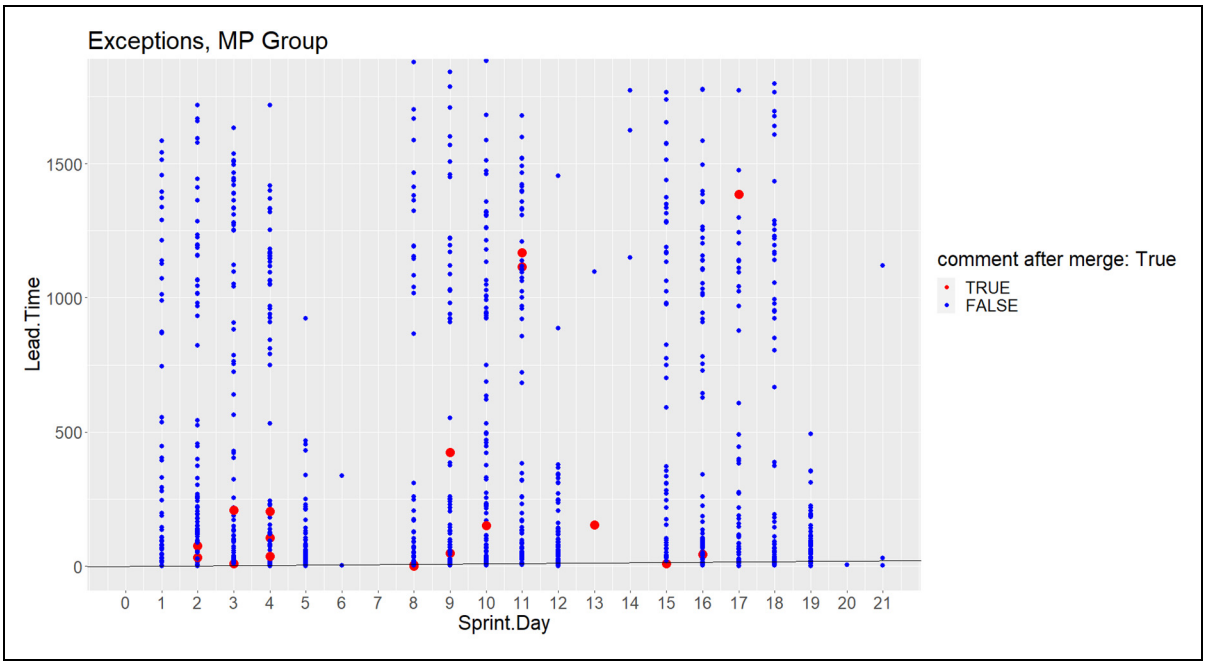

Figure 3.27 MRs with comment(s) after merge during different Sprint Days in MP group

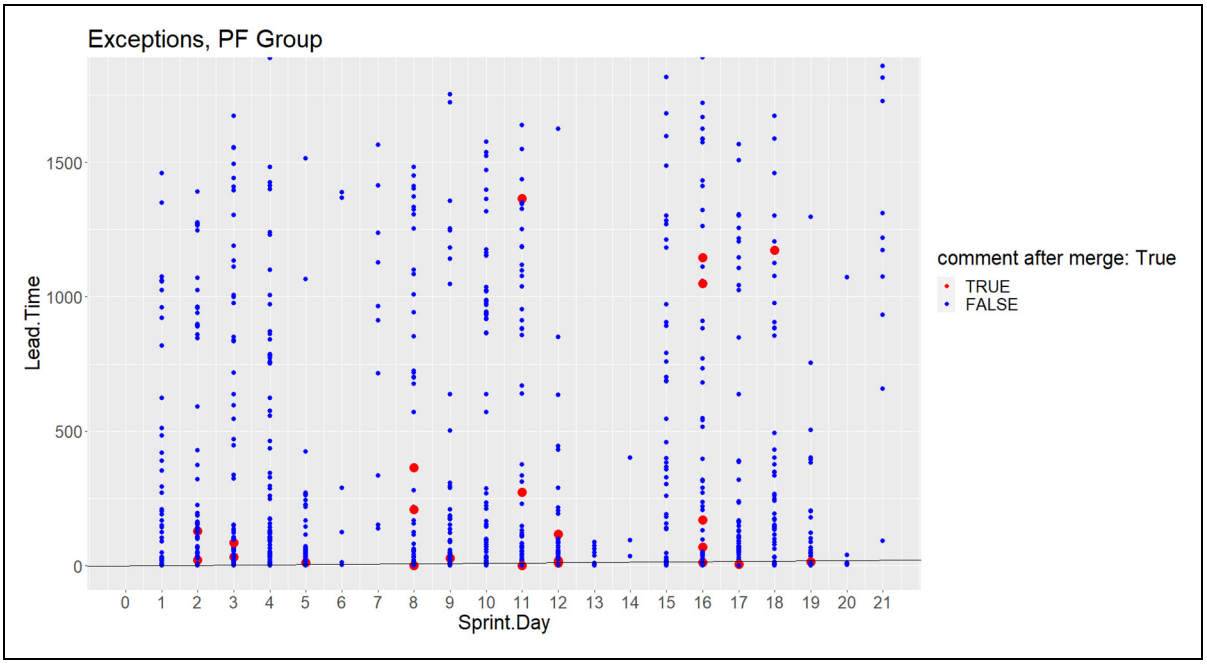

Figure 3.28 MRs with comment(s) after merge during different Sprint Days in PF group

Finding 3: Investigation of the extracted outliers from MR dataset reveals no important information about the MR data anomalies. It demonstrates the necessity of using manual investigation of MR data.

MR outliers are cases very far from the mass of data in MR dataset. We use Isolation Forest to detect outliers in MR dataset. We identified one and two percent of outliers by adjusting the contamination parameter in the Isolation Forest algorithm. Figures 3.29 and 3.30 show one and two percent of outliers detected by Isolation Forest considering the three metrics of Lead Time, Size, and Rework Commits, respectively. According to the 3D plots in this Figure, outliers (blue samples) are separated from the mass of normal samples (green samples).

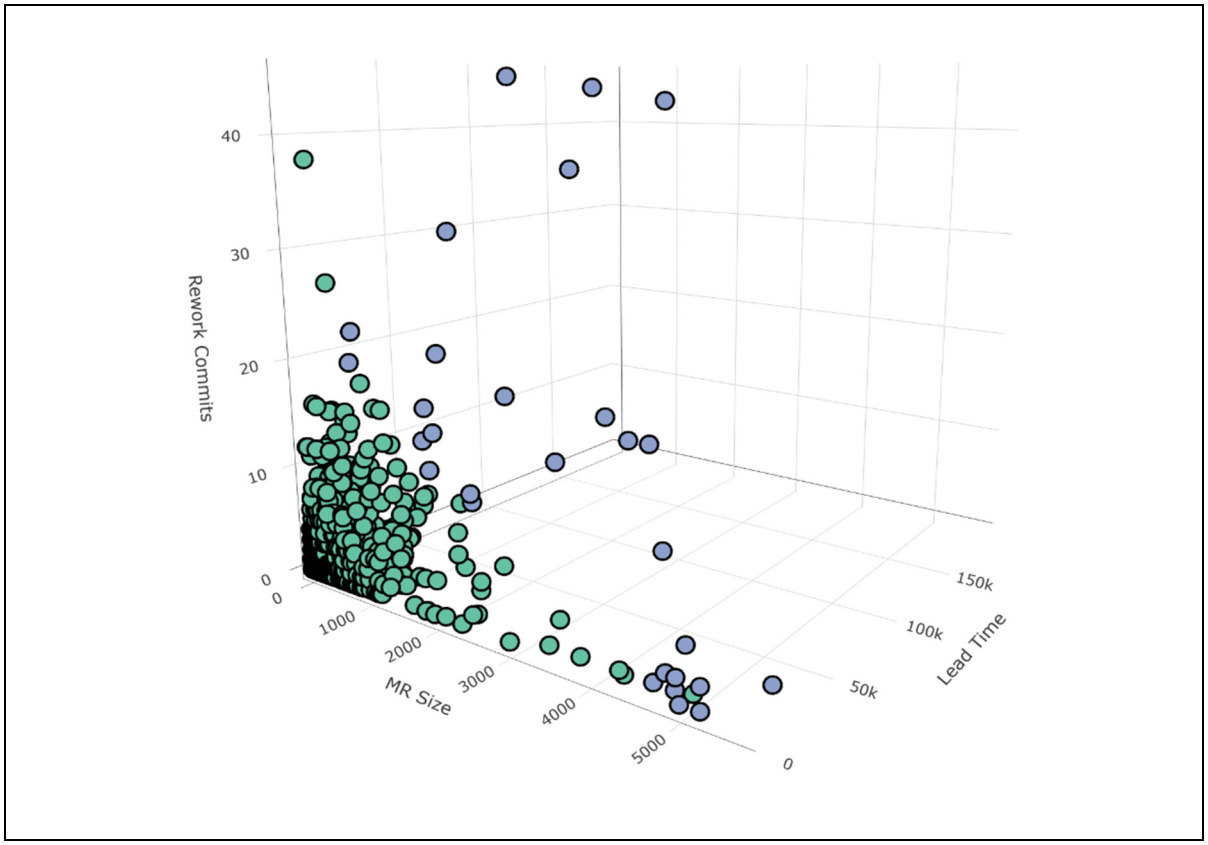

Figure 3.29 One per cent of outliers in MR data set detected by Isolation Forest

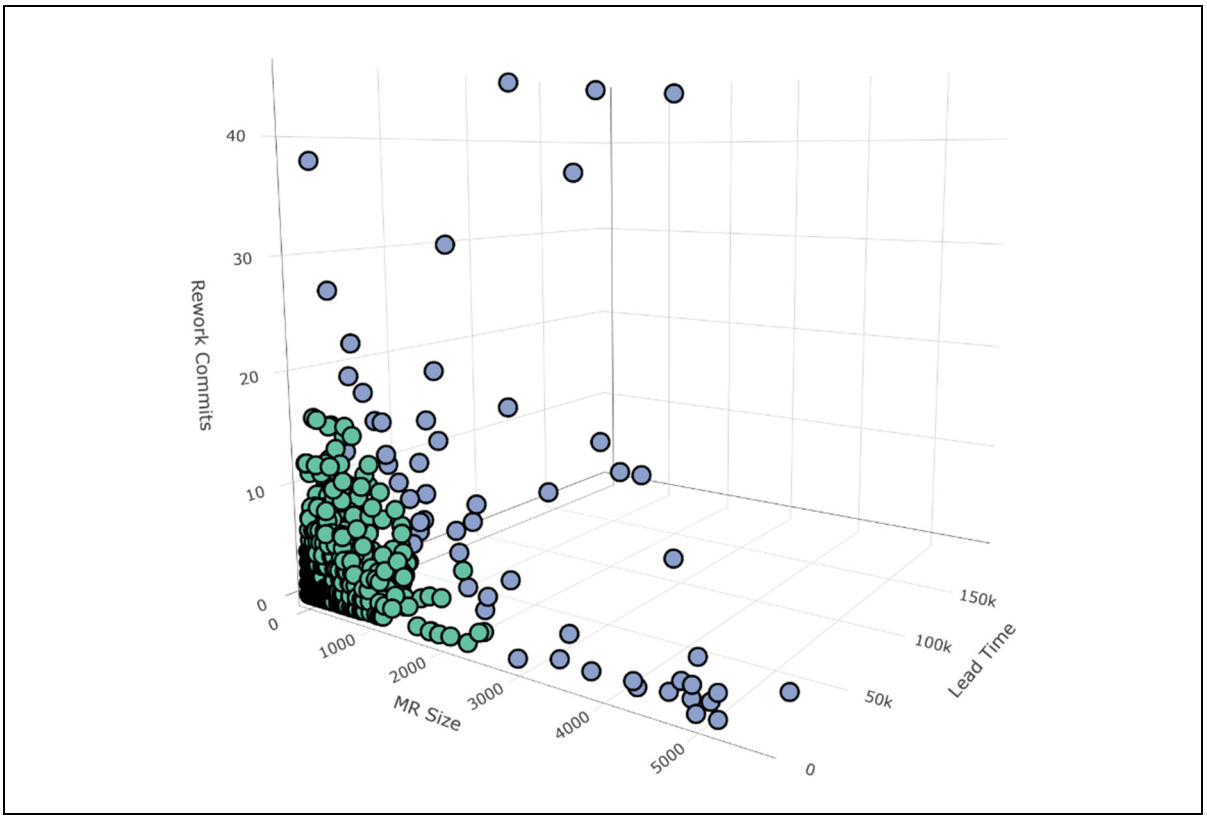

Figure 3.30 Two per cent of outliers in MR data set detected by Isolation Forest

Considering all value metrics in MR dataset, we obtained outliers for the four studied groups. According to our observation, outliers are MRs that are far from the typical MRs in a dataset. For example, an MR with a considerably high Lead Time, very large Size, and very high number of discussions is a data outlier that can be examined separately. Further analysis are required to explore data outliers in MR dataset.

DISCUSSIONS AND CONCLUSIONS

Summary

In this study, we analysed MR data extracted from GitLab at Kaloom. The goal of these analysis is to help Kaloom improve the Review phase of its software delivery process.

We explored trend of change in the Lead Time and Size of MRs during the 21 days of sprints in four groups involved in the software delivery process at Kaloom. We investigated correlation between different metrics in the four studied groups using Spearman method. We observed no correlation between the Lead Time and sprint day of MRs in different groups (less than 0.025). We also observed a very weak correlation (less than 0.07) between the Lead Time and Size of MRs in the studied groups. We explored how some specific characteristics of each group can influence Lead Time and Size of MRs. For example, we observed that DP and PF groups with hardware tests in their delivery process have higher average MR Lead Time compared to other groups. DP group with no static code analysis for its used programming language (P4) has a highest average Lead Time among all groups.

We carried out a set of statistical analysis to explore behavior practices of software engineers during different days of sprints. We compared Lead Time, Size, and number of the MRs with the average values of these metrics in each group. From developer perspective, we observed that they create large Size MRs in the last day of the second week in sprints. It can be a sign of a specific behavior practice in developers where they merge large code changes before the start of last week of the sprints. From reviewer perspective, we observed that reviewers inspect a larger number of MRs in the middle of the sprints (12^{th} day of the sprints in CP and MP groups, and 11^{th} day of the sprint in DP group). It can signal a behavior practice of reviewers in these three groups where they review a high number of MRs before the start of last week in the sprints. We also investigated how high number of assigned reviews to highly involved reviewers can lead to higher number of slow MRs. Finally, we explored abnormal data,

including exceptional MRs and data outliers, by manual and quantitative analysis. We observed different categories of exceptional MRs. We found out the reason behind their abnormal information. For example, MRs with comments after the merge, as a category of exceptions, are those with detected bugs later in different quality level (QL) tests. According to our investigation, these MRs are more commonplace in the groups in which detection and troubleshooting of bugs in their Review phase is more complicated. However, we observe no link between the occurrence of exceptional MRs and their creation day. Finally, we used Isolation Forest to quantitatively analyse abnormal MRs. We observed that MR outliers are far from typical MRs in MR dataset. However, we were not able to conclude important information from observation of outliers in MR dataset.

Research limitations

This research work has been conducted in the context of the Kaloom-Telus Industrial Research Chair at École de technologie supérieure ÉTS. These limitations are mentioned below:

- Our research work focuses on optimizing the code review process in an industrial use case. Thus, improving other aspects of the workflow in CICD pipeline including product build/packaging, product integration testing, and product feature testing are out of scope of this research work. It is no doubt true that the DevOps journey to optimize toward faster reliable flows can be expanded to other phases of the CICD pipeline in the future research topics defined by directors and executives of Research Chair.

- DevOps mindset is not to blame organization personnel, but it is based on the culture of continual learning and experimentation. Improvements are aimed to target general behavior practices instead of individual habits or traits of engineers. Therefore, exploring expertise, education, and behavior practices of personnel involved in different groups at Kaloom is not within the scope of this study.

- Qualitative analysis based on collecting information through questionnaire among both software engineers could be useful when exploring patterns and trends in various groups involved in software delivery process at Kaloom. However, according to the confidentiality of some documents and workflows in the use case company, and the privacy of their personnel, this research work is limited to manual and quantitative analysis of the extracted data from mining GitLab.

Threats to validity

Despite all efforts to mitigate any threat concerning this research work, there are some assertions which are subject to threat the validity of interpretations.

- Some of the metrics including MR Size and Number of Discussions may not be very precise. MR Size is calculated based on the accumulation of absolute number of deleted and inserted lines of codes. We were not able to find an accurate way to exclude empty lines of codes from the calculation of MR Size. Therefore, our conclusions in response to RQ1 and RQ2 about the correlation between MR Lead Time and Size, and also behavior practice of developers needs further work on the calculation of MR Size as one of the involved metrics. What's more, although number of discussions is proportional with the number of comments by reviewers, they are different. In fact, number of comments as a metric reported by GitLab API is an accumulation of system notes, commits, and comments on that MR. This metric also requires improvement in the way of calculation.

- Manual investigation of MRs may not be able to uncover all categories of exceptions in MR data set. Actually, there may be several cases of exceptional MRs that need further investigation.

- Exceptional MRs in MR dataset may affect trends of change in Lead Time and Size of MRs in different groups. Therefore, excluding exceptions from MR data may change our interpretations from patterns and trends over different sprint days.

- While our study focuses on a specific industrial context, we do not generalize our findings to other software systems or other code review platforms. Though, our study covers four different groups with dozens of projects. In addition, we found interesting results and in particular different usage practices for code review that can be replicated in other industrial contexts as well as open-source systems.

Future research works

Several ideas can be suggested based on the experience gained in this study.

Firstly, DevOps not only aims at enhancing different stages of product build in the value stream using support tools and techniques, but also intends to improve organization culture by replacing deprecated mindsets and values. Therefore, as future research work, a study could be conducted to document other qualitative metrics such as developer experience and expertise level as long as it raises no privacy concern with respect to the research ethics.

Secondly, text analysis of MR titles can be implemented to help with a better detection of data anomalies. What we observed during the process of case-by-case MR inspections was that a part of revert changes, version updates, or rolling out CICD pipeline scripts/plugins could be detected through analysis of MR title. For this purpose, ML-based text analysis techniques could be used to detect the exact words or the similarities in the MR subjects.

Finally, adding a new set of metrics would be helpful to extract work habits of engineers. These metrics can encompass more complex measurements such as “Area Hotness” mentioned in (Yu et al., 2016).

LIST OF REFERENCES

AbdelMoez, W., Kholief, M., & Elsalmy, F. M. (2013). Improving bug fix-time prediction model by filtering out outliers. *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, 359–364. https://doi.org/10.1109/TAEECE.2013.6557301

Alali, A., Kagdi, H., & Maletic, J. I. (2008). What's a Typical Commit? A Characterization of Open Source Software Repositories. *2008 16th IEEE International Conference on Program Comprehension*, 182–191. https://doi.org/10.1109/ICPC.2008.24

Amer, M., Goldstein, M., & Abdennadher, S. (2013). Enhancing one-class support vector machines for unsupervised anomaly detection. *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*, 8–15. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2500853.2500857

Bertoncello, M. V., Pinto, G., Wiese, I. S., & Steinmacher, I. (2020). Pull Requests or Commits? Which Method Should We Use to Study Contributors' Behavior? *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 592–601. https://doi.org/10.1109/SANER48275.2020.9054855

Bordeleau, F., Cabot, J., Dingel, J., Rabil, B. S., & Renaud, P. (2020). Towards Modeling Framework for DevOps: Requirements Derived from Industry Use Case. In J.-M. Bruel, M. Mazzara, & B. Meyer (Eds.), *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment* (pp. 139–151). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-39306-9_10

Bradley, N. C. (2017). Teamline: Visualizing small team code contributions. Retrieved January 5, 2022, from https://www.semanticscholar.org/paper/Teamline-%3A-Visualizing-small-team-code-Bradley/194cb980374ad73c79d4b5e582b1c4472beca371

Chatley, R., & Jones, L. (2018). Diggit: Automated code review via software repository mining. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 567–571. https://doi.org/10.1109/SANER.2018.8330261

ETS DEVOPS / gitlab-mr-extraction. (n.d.). Retrieved January 31, 2022, from GitLab website: https://gitlab.com/ets-devops/mr-analysis-tool

ÉTS welcomes the Kaloom-TELUS Industrial Research Chair | ÉTS Montréal. (n.d.). Retrieved January 18, 2022, from http://www.etsmtl.ca/en/news/2021/chaire-kaloom-telus-ets

Farias, M., Novais, R., Ortins, P., Colaço, M., & Mendonça, M. (2015). Analyzing Distributions of Emails and Commits from OSS Contributors through Mining Software Repositories—An Exploratory Study: *Proceedings of the 17th International Conference on Enterprise Information Systems*, 303–310. Barcelona, Spain: SCITEPRESS - Science and and Technology Publications. https://doi.org/10.5220/0005368603030310

Giger, E., Pinzger, M., & Gall, H. (2010). Predicting the fix time of bugs. *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*, 52–56. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1808920.1808933

Halkidi, M., Spinellis, D., Tsatsaronis, G., & Vazirgiannis, M. (2011). Data mining in software engineering. *Intelligent Data Analysis*, *15*(3), 413–441. https://doi.org/10.3233/IDA-2010-0475

Hejazi, M., & Singh, Y. P. (2013). One-Class Support Vector Machines Approach to Anomaly Detection. *Applied Artificial Intelligence*, *27*(5), 351–366. https://doi.org/10.1080/08839514.2013.785791

Heller, B., Marschner, E., Rosenfeld, E., & Heer, J. (2011). Visualizing collaboration and influence in the open-source software community. *Proceedings of the 8th Working Conference on Mining Software Repositories*, 223–226. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/1985441.1985476

Hu, W., Liao, Y., & Vemuri, V. R. (n.d.). *Robust Anomaly Detection Using Support Vector Machines*. 7.

Husain, W., Low, P. V., Ng, L. K., & Ong, Z. L. (2011). *Application of Data Mining Techniques for Improving Software Engineering*. 5.

Iyer, R. N., Yun, S. A., Nagappan, M., & Hoey, J. (2021). Effects of Personality Traits on Pull Request Acceptance. *IEEE Transactions on Software Engineering*, *47*(11), 2632–2643. https://doi.org/10.1109/TSE.2019.2960357

Kim, G., Humble, J., Debois, P., Willis, J., & Forsgren, N. (2021). *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations*. IT Revolution.

Kononenko, O., Rose, T., Baysal, O., Godfrey, M., Theisen, D., & de Water, B. (2018). Studying pull request merges: A case study of shopify's active merchant. *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 124–133. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3183519.3183542

Kumar J., M., Dubey, S., Balaji, B., Rao, D., & Rao, D. (2018). Data Visualization on GitHub Repository Parameters Using Elastic Search and Kibana. *2018 2nd International Conference on Trends in Electronics and Informatics (ICOEI)*, 554–558. https://doi.org/10.1109/ICOEI.2018.8553755

Lal, H., & Pahwa, G. (2017). Code review analysis of software system using machine learning techniques. *2017 11th International Conference on Intelligent Systems and Control (ISCO)*, 8–13. https://doi.org/10.1109/ISCO.2017.7855962

LI, H., SHI, S., THUNG, F., HUO, X., XU, B., LI, M., & LO, D. (2019). DeepReview: Automatic code review using deep multi-instance learning. *Advances in Knowledge Discovery and Data Mining: 23rd Pacific-Asia Conference, PAKDD 2019, Macau, China, April 14-17: Proceedings*, *11440*, 318–330. https://doi.org/10.1007/978-3-030-16145-3_25

Li, Z.-X., Yu, Y., Yin, G., Wang, T., & Wang, H.-M. (2017). What Are They Talking About? Analyzing Code Reviews in Pull-Based Development Model. *Journal of Computer Science and Technology*, *32*(6), 1060–1075. https://doi.org/10.1007/s11390-017-1783-2

Liang, J., & Mizuno, O. (2011). Analyzing Involvements of Reviewers through Mining a Code Review Repository. *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, 126–132. https://doi.org/10.1109/IWSM-MENSURA.2011.33

Liu, D., Wang, L., Chen, Y., Elkashlan, M., Wong, K.-K., Schober, R., & Hanzo, L. (2016). User Association in 5G Networks: A Survey and an Outlook. *IEEE Communications Surveys Tutorials*, *18*(2), 1018–1044. https://doi.org/10.1109/COMST.2016.2516538

Liu, F. T., Ting, K. M., & Zhou, Z.-H. (2008). Isolation Forest. *2008 Eighth IEEE International Conference on Data Mining*, 413–422. https://doi.org/10.1109/ICDM.2008.17

Mechtri, M., Benyahia, I. G., & Zeghlache, D. (2016). Agile service manager for 5G. *NOMS 2016 - 2016 IEEE/IFIP Network Operations and Management Symposium*, 1285–1290. https://doi.org/10.1109/NOMS.2016.7503004

Meqdadi, O., Alhindawi, N., Alsakran, J., Saifan, A., & Migdadi, H. (2019). Mining software repositories for adaptive change commits using machine learning techniques. *Information and Software Technology*, *109*, 80–91. https://doi.org/10.1016/j.infsof.2019.01.008

Mishra, R., & Sureka, A. (2014). Mining Peer Code Review System for Computing Effort and Contribution Metrics for Patch Reviewers. *2014 IEEE 4th Workshop on Mining Unstructured Data*, 11–15. https://doi.org/10.1109/MUD.2014.11

Nakimuli, W., Garcia-Reinoso, J., Nogales, B., Vidal, I., Gomes, D., & Lopez, D. (2020). Reducing Service Creation Time Leveraging on Network Function Virtualization. *IEEE Access*, *8*, 155679–155696. https://doi.org/10.1109/ACCESS.2020.3018583

Namiot, D., & Romanov, V. (2020). On Data Analysis of Software Repositories. In V. Sukhomlin & E. Zubareva (Eds.), *Convergent Cognitive Information Technologies* (pp. 263–272). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-37436-5_24

Padhye, R., Mani, S., & Sinha, V. S. (2014). A study of external community contribution to open-source projects on GitHub. *Proceedings of the 11th Working Conference on Mining Software Repositories*, 332–335. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2597073.2597113

Paixao, M., Krinke, J., Han, D., & Harman, M. (2018). CROP: Linking code reviews to source code changes. *Proceedings of the 15th International Conference on Mining Software Repositories*, 46–49. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/3196398.3196466

Robillard, M. P., Coelho, W., & Murphy, G. C. (2004). How effective developers investigate source code: An exploratory study. *IEEE Transactions on Software Engineering*, *30*(12), 889–903. https://doi.org/10.1109/TSE.2004.101

Rose, T. (2017). *Towards Understanding What Factors Affect Pull Request Merges* (Text, Carleton University). Carleton University. Retrieved from https://curve.carleton.ca/4d6605dd-eed2-4b0f-b1e1-1b92f6dea244

Seo, Y.-S., & Bae, D.-H. (2013). On the value of outlier elimination on software effort estimation research. *Empirical Software Engineering*, *18*(4), 659–698. https://doi.org/10.1007/s10664-012-9207-y

Shi, S.-T., Li, M., Lo, D., Thung, F., & Huo, X. (2019). Automatic Code Review by Learning the Revision of Source Code. *Proceedings of the AAAI Conference on Artificial Intelligence*, *33*(01), 4910–4917. https://doi.org/10.1609/aaai.v33i01.33014910

Silva, M. C. O., Valente, M. T., & Terra, R. (2016). Does Technical Debt Lead to the Rejection of Pull Requests? *ArXiv:1604.01450 [Cs]*. Retrieved from http://arxiv.org/abs/1604.01450

Thomas, S. W., Hassan, A. E., & Blostein, D. (2014). Mining Unstructured Software Repositories. In T. Mens, A. Serebrenik, & A. Cleve (Eds.), *Evolving Software Systems* (pp. 139–162). Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-45398-4_5

Tsay, J., Dabbish, L., & Herbsleb, J. (2014). Influence of social and technical factors for evaluating contribution in GitHub. *Proceedings of the 36th International Conference on Software Engineering*, 356–366. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/2568225.2568315

Xie, T., Thummalapenta, S., Lo, D., & Liu, C. (2009). Data Mining for Software Engineering. *Computer*, *42*(8), 55–62. https://doi.org/10.1109/MC.2009.256

Yu, Y., Wang, H., Filkov, V., Devanbu, P., & Vasilescu, B. (2015). Wait for It: Determinants of Pull Request Evaluation Latency on GitHub. *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 367–371. https://doi.org/10.1109/MSR.2015.42

Yu, Y., Yin, G., Wang, T., Yang, C., & Wang, H. (2016). Determinants of pull-based development in the context of continuous integration. *Science China Information Sciences*, *59*(8), 080104. https://doi.org/10.1007/s11432-016-5595-8