

From Trunk-Based to Merge Requests: A Field Study at Adyen

Master's Thesis

The Adyen logo is rendered in a bold, green, lowercase sans-serif font. The letters are thick and blocky, with a distinctive design where the 'y' has a long, horizontal tail that extends to the right, and the 'e' has a similar tail. The overall appearance is clean and modern.

Toon de Boer

From Trunk-Based to Merge Requests: A Field Study at Adyen

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Toon de Boer
born in Velsbroek, the Netherlands



Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl



Adyen
Simon Carmiggeltstraat 6-50
Amsterdam, the Netherlands
www.adyen.com

From Trunk-Based to Merge Requests: A Field Study at Adyen

Author: Toon de Boer
Student id: 4575091

Abstract

Many development models exist, but finding which one is the right for a specific project or software company is difficult. Every project has its requirements and might need its own development model. The most popular development models are trunk-based development and merge requests. There are no clear science-based guidelines on when to adopt one model or the other and challenges that teams face when migrating from one to another. We perform a field study as this master thesis aims to provide more understanding on the impact of migrating from one development model to another at a large company. More specifically, a migration from trunk-based development to merge request-based development at a large software engineering company. During this research, we interview 19 developers, eleven before the migration and eight after the migration, survey 46 developers to triangulate our findings of the interview before the migration, and analyzed the differences in the code reviews made by developers before and after the migration. We show what benefits and challenges developers experience using the trunk-based model and what they expect from the merge request-based model before the migration. Also, we show the change of motivation for code reviews after the migration. Moreover, quantitative data shows that code reviews are completed faster and with more code comments in the merge request-based model. Finally, we provide the perceptions of developers after the migration.

Thesis Committee:

Chair: Prof. Dr. A.E. Zaidman, Faculty EEMCS, TU Delft
University supervisor: Dr. M. Aniche, Faculty EEMCS, TU Delft
Company supervisor: Y. Radomskyi, Adyen
Committee Member: Dr. C.B. Poulsen, Faculty EEMCS, TU Delft

Preface

This thesis is a result of nine months of research on a graduation project for the degree of Master of Science in Computer Science at the Delft University of Technology. Although this thesis does not conclude my study as I will finalize my study program in Madrid next semester, this will be the end of my time as a computer science student in the Netherlands. This thesis would not have been possible without the support of many people, whom I would like to thank.

First of all, professor, supervisor, and colleague Maurício Aniche, thank you for everything you have done for me. When I introduced myself in my first email more than a year ago, you immediately replied with possible research topics and you always stayed very responsive and helpful. Your enthusiasm, insights, and feedback guided me throughout this project more than anything else.

Furthermore, I would like to thank Andy Zaidman and Casper Poulsen for being part of my thesis committee.

Moreover, I would like to thank all the professors I have had over the last five years studying in Delft for teaching me. And thanks to all my fellow students with whom I have had interesting conversations, did projects, and became friends. Without each other, the last years would have been very hard.

I would like to thank Adyen for allowing me to conduct research at their company during the COVID-19 pandemic and this project would not have been possible without the help of many colleagues at Adyen. I would like to thank Yuri Radomskyi for being my supervisor at Adyen and the Development and Testing Tools team for making me feel part of the team and helping me during my research. I would also like to thank all the developers at Adyen who freed their calendars to participate in the interviews and the survey. Without their responsiveness and willingness to help, this thesis would not have been possible.

Last but not least, I would like to thank my family and friends for supporting me during my thesis and study over the years.

Toon de Boer
Delft, the Netherlands
November 1, 2021

Contents

Preface	iii
Contents	v
List of Figures	vii
List of Tables	vii
1 Introduction	1
2 Background and Related Work	3
2.1 Adyen	3
2.2 Related Work	6
3 The Developers' Expectations on the Migration Towards Merge Requests	13
3.1 Methodology	14
3.2 Code Reviews	16
3.3 Development Speed	22
3.4 Merge Conflicts	25
3.5 Security & Safety	27
3.6 Feature Branches	29
3.7 Development Models	32
4 Differences in How Developers Review Code in Trunk-Based and in MRs	37
4.1 Methodology	37
4.2 Qualitative Analysis on Code Comments	39
4.3 Quantitative Analysis: Review Times	40
4.4 Quantitative Analysis: Number of Code comments	45
5 The Developers' Perceptions on the New Merge Request Model	49
5.1 Methodology	49

CONTENTS

5.2	Perceptions on Code Reviews	51
5.3	Development Speed	53
5.4	Merge Conflicts	55
5.5	GitLab	56
5.6	Merge Requests	57
5.7	Migration	58
6	Discussion	61
6.1	Implications	61
6.2	Recommendations	63
6.3	How to expand to the community	63
6.4	Threats to Validity	64
7	Conclusion	67
	Bibliography	71
A	Interview Before the Migration	75
B	Survey	77
C	Interview After the Migration	85

List of Figures

2.1	Trunk-Based Flow.	4
2.2	Merge Request-Based Flow.	5
3.1	Methodology.	14
4.1	Proportion of comments by category.	39
4.2	Days to complete.	41
4.3	Days to complete.	41
4.4	Days to complete.	42
4.5	Days to complete within one month.	42
4.6	Hours to complete within one week.	43
4.7	Hours to complete within one day.	43
4.8	Minutes to complete within one hour.	44
4.9	Days to complete.	44
4.10	Days to complete.	45
4.11	Comments per review.	46
4.12	Comments per review.	46
4.13	Comments per review.	47
4.14	Comments per review.	47
4.15	Comments per review.	48
4.16	Comments per review.	48
5.1	Methodology.	50

List of Tables

3.1	Interview participants.	15
3.2	Responses to the survey (part 1).	17
3.3	Responses to the survey (part 2).	20
3.4	Responses to the survey (part 3).	23
3.5	Responses to the survey (part 4).	25
3.6	Responses to the survey (part 5).	27
3.7	Responses to the survey (part 6).	30
3.8	Responses to the survey (part 7).	33
4.1	Code Comment Categories.	38
4.2	Times between the creation of the review and the closing of the review.	40
4.3	Comments per review.	45
5.1	Interview participants after migration.	51

Chapter 1

Introduction

Every software development company has to pick a development model with the two most popular ones being the trunk-based model and the merge requests. Each model has their own advantages and disadvantages. The trunk-based model is designed for working with all developers on one branch, which is ideal to launch fast and iterate. However, in the trunk-based model, code reviews are not mandatory which means that every developer can commit bad quality code to the repository. The merge request-based model encourages feature branches and requires code reviews before changes are committed to the master branch, which adds an extra level of security to the product. But in the merge request model, the development speed decreases because merge requests are waiting for their approvals to be merged to master.

There has not been a field study on such migration of development model at a large company done before to the best of our knowledge. However, some studies explored the merge request-based model specifically or the importance of code reviews. In our related work section, we will explore researches done in the field of merge requests and code reviews. Reviews before and after commits are compared [25] but not one technique is considered superior. And the main motivations for code reviews are to improve the code and understand what the code is about [4]. Moreover, research shows the merging time in open source projects, where 60% of merge requests are merged within one day [14].

Deciding which development model to use, in particular, the one that will bring most value for a specific company in its specific context, is a challenging decision. Adyen, our case study, decided to move to a merge-request model due to code quality reasons. Code being reviewed after the merge was not desired as too many errors occurred in the master branch that needed to be fixed before the code went in production. Developers expected that code reviews would increase code quality and reduce the number of errors.

In this thesis, we study the impact that such a migration had at Adyen. More specifically, we collect the developers' perceptions and expectations before and after the migration, compare the code reviews that developers performed in both models as well as the time it took developers to do their reviews. To that aim, we propose the following research questions:

RQ1 What are the expectation of migrating from a trunk-based development model to a merge request-based development model at Adyen?

RQ2 What has changed in code reviews after migration from trunk-based to merge request-based at Adyen?

RQ3 How did the developers perceive the migration from trunk-based development to merge request-based development at Adyen?

To answer these research questions we used qualitative and quantitative research methods. First, we conduct eleven semi-structured interviews with developers to get an understanding of the problem. After that, we survey developers to generalize our findings. We will analyze the code reviews from the trunk-based models and merge requests to get an understanding of what the migration has for impact on the code reviews. Finally, we will interview eight developers that have used the merge request-based model at Adyen to get their perceptions on the migration.

The results show that developers faced challenges in reviewing code in the trunk-based model, which resulted in errors on the master branch that were caught only during testing. These errors had to be fixed immediately to prevent them from going to production, while they could have been prevented when the responsible code was reviewed before the merge. Also, commits get mixed up in the master branch instead of sorted on the feature. The benefit of trunk-based development is the fast development speed that it provides. Results of the quantitative analysis show that the code reviews are completed faster and more comments are placed in the merge request-based model. Finally, developers who experienced the merge request-based model prefer working with merge requests as they perceive an increase in code quality. They also feel that better discussions emerge from mandatory code reviews, and more knowledge is shared among developers.

This research makes the following contributions:

- An understanding of the benefits and challenges of the trunk-based model and the expectations of developers before the migration of a trunk-based model to a merge request-based model.
- A qualitative and quantitative analysis of the code reviews in a trunk-based model compared to the merge request-based model.
- The perception by developers of migrating development model at a large company and the benefits and challenges of the merge request-development model.

The structure for the remainder of the thesis is as follows. First, we will provide background information on the project and discuss the related work in chapter 2. Then we will discuss the work done in three chapters. First, we will discuss the trunk-based model and the expectations of the migration in chapter 3. Secondly, we will discuss the qualitative and quantitative analysis done on code reviews in the trunk-based model compared to the merge request-based model in chapter 4. Thirdly, the perceptions of this migration will be discussed in chapter 5. After that, a discussion with recommendations and threats to validity will be provided in chapter 6. Finally, we will conclude the thesis in chapter 7 and propose future work.

Chapter 2

Background and Related Work

Merge requests and code reviews are not a new practice used in the software engineering domain. In this chapter, we will provide background information to this research and discuss previous research done on merge requests, code reviews, and empirical research within computer science.

2.1 Adyen

First of all, to understand this research we will explain the trunk-based development model and the merge request-based development model as they are implemented by Adyen. After that, we give an overview of the different tools that are being used, which are relevant to this research.

2.1.1 About Adyen

Adyen¹ is a payments company offering payment services to retail merchants and is responsible for transferring money from a shopper's bank account to a merchant's bank account. This can be done either online (eCommerce) or in-store (point of sale) and Adyen provides one single solution which includes both. Examples of eCommerce transactions are payments made in webshops, in-app, or subscriptions, which can be done with a credit card, iDeal, WeChat Pay, and much more. Point of sale (POS) transactions are physical payments made on payment terminals with for example card or phone. All sales channels connect to the same online platform, which gives merchants with both an online webshop and physical stores better insights.

2.1.2 Trunk-Based Development Model

The trunk-based development model is a model where there is a single master 'trunk' branch on which every developer contributes code. There are no separate branches, so when a developer pushes a commit, this code is immediately on the master branch.

¹adyen.com

Adyen preMerge: Submit flow

Trunk based development with pre-push validation: Adyen preMerge

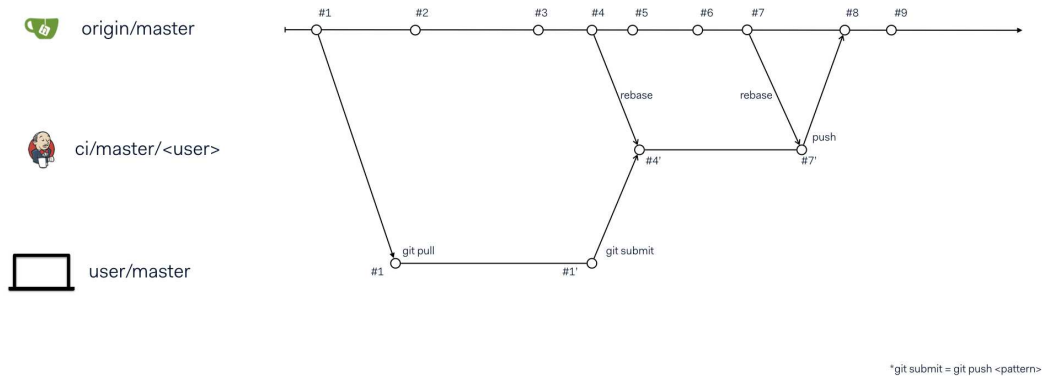


Figure 2.1: Trunk-Based Flow.

For Adyen, it works slightly differently, which is illustrated in Figure 2.1. The flow starts with a developer pulling the git repository to its local machine, this version is illustrated with #1. Next, the developer makes changes to the code and ends up with version #1', while in the meantime other code contributions have been made to the master (#2, #3, #4). When the developer is done with their code changes, they do a `git submit` command, which will rebase and pull the current master, which is on version #4 and runs the pipeline with all the tests on the Jenkins server. During this time, new contributions may be merged into master (#5, #6, #7). When the pipeline is complete and all the tests succeeded, a new rebase and pull from the master is executed automatically with the newest version #7 and immediately pushed to master. This contribution is now the newest version #8. After the merge, the code review is opened and a random reviewer gets assigned to review the code.

2.1.3 Merge Request-Based Development Model

The merge request-based development model is a model where feature branches are encouraged and to get one's code into the master branch, one needs to create a merge request. This merge request can contain any number of commits. When the merge request is approved by the reviewer, the code will be merged into the master branch.

An example workflow is shown in Figure 2.2. Again, the flow starts with pulling the repository to your local machine, noted with version #1. Locally the developer can make multiple commits, shown in the *local* swimlane with version #2'. When the developer does a `git push`, they need to specify the branch name with the format `mr/master/<user>/<branch-name>` otherwise the merge request will not be merged to master. The merge request needs to be created on the GitLab site and will be put in *Draft*, which means that no reviewers will

Gitlab: Merge Request flow

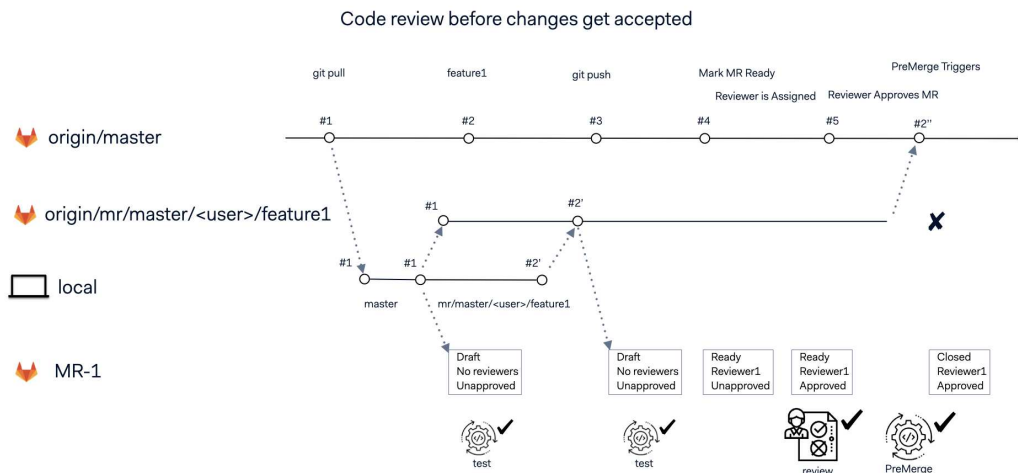


Figure 2.2: Merge Request-Based Flow.

get auto-assigned and the merge request cannot be merged. With every push, the pipeline is run for each new commit as shown in the *MR-1* swimlane. When the merge request is marked *Ready*, a code reviewer gets automatically assigned to the merge request. When the reviewer approves the merge request, the merge pipeline is started, which will rebase the master branch, run all the checks and tests, merge the merge request, close the merge request, and remove the merge request branch. During this time, new contributions could have been made to the master (#2, #3, #4, #5). The new version of this example is noted as #2" in the master branch.

To simplify this approach, Adyen provides an alternate flow which replaces *git push* with the custom *git mr submit* command, similar to the *git submit* command from the trunk-based model. The *git mr submit* command automatically creates a local merge request branch with the right format, moves commits from master to this branch, pushes commits to the remote branch, creates the merge request in *Ready* state (reviewers get automatically assigned and the merge request automatically merge on approval), and the repository is switched back to master, so the developer can continue on the next merge request. This alternative flow saves the developer extra work and time.

2.1.4 Tools

At Adyen, different tools were used for the trunk-based model and the merge request-based model. With the trunk-based model Gitea² was used as the remote GIT repository, while in the merge request-based model GitLab³ is used. The continuous integration pipeline

²gitea.io

³gitlab.com

runs in Jenkins⁴ for both models. In the trunk-based approach, code reviews are created in Upsource⁵. For the merge request-based approach the reviews are done in GitLab itself.

2.2 Related Work

Merge requests are widely used in the software engineering industry. This section is dedicated to a selection of research done in these areas related to merge requests and code reviews as an important part that comes along with merge requests is the code review. On a short note, this empirical research is based on the methods described by the lectures on Empirical Software Engineering by Ferrari [12] and papers that conduct empirical research [1, 2, 18, 29, 35, 37].

2.2.1 Merge Requests

There is much research done on the merge request-based model. In this subsection, we discuss relevant papers to this thesis on merge requests.

Exploring Merge Requests

Gousios et al. have researched merge requests on open source software [14]. This paper explores how pull-based software development works on the GHTorrent corpus [13] and selected GitHub projects. The paper shows that a relatively small number of factors affect the decision to merge a pull request and the time to process it.

Distributed Version Control Systems (DVCS) enables a potential contributor to submit a set of changes to a software project. There are two strategies, *shared repository* and *pull requests*. A shared repository (trunk-based) allows contributors to push their changes back to the central branch. With pull requests (merge requests), when a set of changes is ready to be submitted to the main repository a merge request is created, which specifies a local branch to be merged with the master branch. Merge requests can be used as a requirement and design discussion tool or as a progress tracking tool towards the fulfillment of project releases.

There are three ways to merge [10]. Automatically when there are no conflicts to the base repository. Branch merging, merge fork into the base repository, or cherry-picking by merging only a few selected commits. And a merger can create a textual difference between the upstream and the merge request branch, which they then apply to the upstream branch. In the last case, both history and authorship information is lost.

Merge request usage is increasing in absolute numbers, even though the proportion of repositories using merge requests has decreased slightly. This might be due to a lot of individual projects as large projects count the same as small projects. An interesting finding is that interviewed developers state that the presence of tests in merge requests is a major factor for their acceptance, but this does not seem to affect the merge decision nor the merge time. On the other hand, quantitative analysis shows that code reviews affect the time to

⁴jenkins.io

⁵upsource.jetbrains.com

merge a merge request and the decision to merge a merge request is mainly influenced by whether the merge request modifies recently modified code. The time to merge is influenced by the developer's previous track record, the size of the project and its test coverage, and the project's openness to external contributions. Moreover, 53% of merge requests are rejected for reasons having to do with the distributed nature of merge request-based development. Only 13% of the merge requests are rejected due to technical reasons. Findings show that 80% of merge requests are merged within four days, 60% in less than a day, and 30% within one hour (independent of project size). In this thesis, we will also look at the completion times of merge requests in section 4.3. The advice given by Gousios et al. is to keep merge requests as short as possible and to invest in a comprehensive test suite.

Work Practices and Challenges of Merge Requests

Another research on challenges in merge request-based development focuses on the role of the integrator [15]. In this paper, Gousios et al. conduct an exploratory qualitative study involving a large-scale survey of 749 integrators, to which quantitative data is added from the integrator's project.

For the study, data is gathered from GitHub using the GHTorrent database [13]. The researchers conducted a two-round (pilot and main) survey with 21 and 749 participants respectively. The two key factors integrators are concerned with are quality and prioritization.

In the pilot round, the researchers analyzed the results and identified the themes of quality and prioritization, which were addressed by including related questions in the second round. Both surveys were split into three sections and were intermixed throughout the survey: demographic information, multiple-choice or Liker-scale questions, and open-ended questions. Some multiple-choice questions also had the 'other' option. The survey took approximately 15 minutes. For the analysis of the open questions, manual coding was applied. At least one and up to three codes were applied to each answer.

The results show that at least half of the integrators use merge requests to discuss new features. This is the GitHub-promoted way of working with merge requests, where a merge request is opened as early as possible to invite discussion on the developed feature (The merge request will be marked as WIP, Work In Progress). Another reason to use merge requests is that there is a policy in place that, for example, each merge request requires at least 2 reviews before it is merged. Moreover, results show that 75% of the integrators use inline code comments, and the most important signals used by integrators when deciding on whether to merge a merge request are code quality, code style, project fit, technical fit, and testing.

Integrators decide to accept a contribution based on its quality and its degree of fit to the project's roadmap and technical design. If the merge request fixes a serious bug with minimal changes, it is more likely to be accepted. Also, conformance to the project style and architecture, test coverage, and small merge requests with good documentation is preferred. One-fifth of the integrators prioritize merge requests based on their criticality (bug fixes), their urgency (new features), and their size. Also, contributors known to the integrators tend to get higher priority.

2. BACKGROUND AND RELATED WORK

There are multiple technical and social challenges of a merge request-based development model. Technical challenges are that reviewers are not always available or merge requests contain multiple features and affect multiple areas of the project. Also, there could be a lack of knowledge on merge conflicts among the contributors. A social challenge is that multiple communication channels are used, integrators find it difficult to synchronize between multiple sources.

Gousios et al. also researched work practices and challenges of merge requests focused on the contributor's perspective [16]. The research shows that contributors have a strong interest in maintaining awareness of project status to get inspiration and avoid duplicating work. Moreover, they often use communication channels external to merge requests and the biggest contributor report is the poor responsiveness from integrators.

Feature Branches

Bird et al. focuses on *git* and its features to create branches and have a decentralized source code management [8]. Continuing on this research, Bird and Zimmermann did an empirical research feature branches at Microsoft [7] to assess the cost and benefit of branches to aid in several branch-related scenarios.

Barr et al. find that feature branches allow developers to collaborate on tasks in different branches while enjoying reduced interference from developers working on other tasks, even if those tasks are strongly coupled to theirs [6].

Development Speed

Jiang et al. [19] performed a study on the Linux kernel project, which is using the merge request-based model, and found that patches developed by more experienced developers are more easily accepted and faster reviewed and integrated. Moreover, through time, the contributions became more frequent and code reviews took less time.

2.2.2 Code Review

Code reviews are an important part of merge requests as the reviewer often decides whether the code will be merged to the master branch. In this section, we will discuss previous work related to code reviews.

Testing in Code Reviews

For production code, many open source and industrial software projects employ code review, but the question remains whether and how code review is also used for ensuring the quality of test code. Research has been done by Spadini et al. [30] where the researchers conducted quantitative analysis on more than 300,000 code reviews and qualitative analysis by interviewing twelve developers.

There are four main research contributions in this paper. First of all, results show that there is no association between the type of code (production or test) and future defects. Secondly, test files are not discussed as much as production files during code reviews.

Thirdly, developers face a variety of challenges when reviewing test files, including dealing with a lack of testing context, poor navigation support within the review, unrealistic time constraints imposed by management, and poor knowledge of good reviewing and testing practices by novice developers. Finally, a tool called GerritMiner was created to help the researchers collect a dataset of 654,570 code reviews from open source, industry-supported software systems.

Related work showed that both test code and production code suffer from quality issues [3, 23, 38] and more than half of the projects studied had bugs in the test code [36]. Also, current bug detection tools are not tailored to detect test bugs, thus making the role of effective test code review even more critical. The researchers hypothesize that there are substantial differences in how test code and production code are reviewed.

This paper specifies three review scenarios where files are modified: both production and test files, only production files, or only test files. For the semi-structured interview, each interview started with general questions about code reviews.

The results show that test files are almost 2 times less likely to be discussed during code review when reviewed together with production files. Another finding is that some reviewers prefer to inspect test code before production code and vice versa. The main concern of reviewers is understanding whether the test covers all the paths of the production code and ensuring tests' maintainability and readability. A big challenge for reviewers is that there is a lack of test-specific information within code review tools, which forces reviewers to inspect the code in their local IDE to navigate through the dependencies. Also, reviewing test files requires developers to have context about the production code. In addition, test files are often much longer with new additions instead of modifications which makes the review harder. Finally, an overall problem is that test files are considered less important and novice developers and managers are not aware of the impact of poor testing and reviewing in software quality.

Modern Code Review

Another paper describes an exploratory investigation of modern code review at Google by Sadowski et al. [28]. There are three research methods used: semi-structured interviews, a survey, and quantitative analysis of log data.

Participants for the interviews were selected using snowball sampling, starting with developers known to the paper authors. From this pool, participants were selected to ensure a spread of teams, technical areas, job roles, length of time within the company, and role in the code review process. In total, twelve interviews were conducted with each interview taking approximately one hour. To analyze the data, first open coding was used, and later also closed coding was performed by another author. For the quantitative data about the code review process, logs produced by the review tool were used. The final dataset includes nine million changes created by more than 25,000 authors and reviewers, and 13 million comments were collected from all changes between September 2014 and July 2016. An online questionnaire was sent to 98 engineers, which resulted in 44 valid responses (45% response rate). The survey asked respondents about how they perceived the code review for their specific recent change. This strategy allowed to mitigate recall bias.

2. BACKGROUND AND RELATED WORK

The main reason behind the introduction of code review was to force developers to write code that other developers could understand. However, three additional benefits became clear: checking the consistency of style and design, ensuring adequate tests, and improving security by making sure no single developer can commit arbitrary code without oversight. Moreover, four key themes for what Google developers expect from code reviews were identified when coding the interviews: education, maintaining norms, gatekeeping, and accident prevention.

The code review at Google is linked to two concepts: ownership and readability. Any developer can propose a change to any part of the codebase, but an owner of the directory must review and approve the change before it is committed. Developers can gain readability certificates per language to be able to review a code review on readability.

Results show that the median time for a developer to wait on initial feedback is under an hour, 90% of the changes to review modify fewer than ten files, one reviewer is often deemed sufficient, and developers spend around three hours per week on code review. Small changes are always preferred because review quality is higher and review latency decreases. Also, reviewers with knowledge of the code under review give more useful comments. Developers who started within the past year typically have more than twice as many comments. Moreover, then the number of files reviewed increases much more than the number of files edited over time. The majority of changes are small, have one reviewer and no comments other than the authorization to commit. During the week, 70% of changes are committed less than 24 hours after they are mailed out for an initial review. We will compare this result with the review duration at Adyen in section 4.3.

Another paper by Bacchelli and Bird [4] describes an empirical study on modern code reviews across different teams at Microsoft. The study reveals that finding defects remains the main motivation for reviews, but reviews also provide additional benefits such as knowledge transfer, increased team awareness, and the creation of alternative solutions to problems.

The research method consists of six steps, analysis of previous studies, observations and interviews with 17 developers, card sort on interview data, card sort on code review comments, the creation of an affinity diagram, and survey to managers and programmers. The interviews took 40-60 minutes each, with the respondents' time at the company ranging from 18 months to ten years and the card sort was done with open coding. The final surveys were first sent to the managers and the second survey was sent to randomly selected developers.

From the qualitative analysis, the results show that finding defects is the most important reason for code reviews, while the quantitative analysis pointed out that code improvement and understanding are the most occurring categories. Also, code reviews are of higher quality and done faster when the reviewer is familiar with the code under change. In section 4.2, we will compare code comments categories of Bacchelli and Bird with the code comments at Adyen before and after the migration. The limitation of this research is that managers and developers indicated that knowledge transfer is important in code reviews, however, this could not be validated through quantitative analysis, since it is difficult to assess this from the discussions in reviews.

Code Review Factors

Porter et al. examined several empirical studies in 1995 on software inspections and assessed their costs and benefits [24]. Moreover, they reported the effects of factors such as team size, type of review, and the number of sessions on code inspections.

Reviewer Recommendation

Rigby and Storey researched broadcast-based peer reviews on open source projects [27], where code reviews are not assigned to specific individuals but hundreds of potential reviewers. They find that this works well in practice as developers find code changes they are competent to review.

Thongtanunam et al. find that code quality can be improved when the right reviewer is assigned [32], but this is a challenge in modern code reviews. They proposed a recommendation tool that suggests reviewers for code reviews based on their past experience in the project. They also found that this speeds up the code review process.

Balachandran uses static analysis tools in the review process to automatically assign reviewers to reviews [5]. Moreover, this tool places comments in code reviews that developers should fix, which increases the overall code quality.

Code Review Acceptance

Tsay et al. researched the likelihood of contribution acceptance in open source projects [34] and found that both social and technical factors play a role. Merge requests with many comments were less likely to be approved and project managers look at the developer's prior contributions to the project when evaluating merge requests.

Thongtanunam et al. [33] study review participation on modern code reviews in multiple open source projects. They find that the description length of a patch shares a relationship with the likelihood of receiving poor reviewer participation or discussion and the introduction of new features can increase the likelihood of receiving slow initial feedback compared to bug fixes or documentation issues.

Hirao et al. investigate why reviewers do not agree with each other in open source projects and provide suggestions for handling this problem [17]. They find that more experienced reviewers are more likely to agree and reviewers with a lower level of agreement take more time to complete their reviews.

Knowledge Sharing

Rigby and Bird research peer reviews in software projects [26]. They find that peer reviews increase the number of distinct files a developer knows about by 66% to 150% depending on the project. Moreover, they concluded that reviewers prefer discussions and fixing code over reporting defects.

Sutherland and Venolia studied code review practices of software product teams at Microsoft [31]. They found that during the review there is an exchange of knowledge between

the code author and the reviewer, but that retention and recovery of this information is not well supported

Code Quality

McIntosh et al. studied the relationship between code quality and: (1) code review coverage, and (2) code review participation [22]. They did this through a case study and found that both code review coverage and code review participation share a significant link with the code quality as poorly reviewed code harms software quality in large systems using modern code review tools.

Kollanus and Koskinen presents a literature survey on code inspection [20]. There are 153 articles included and the main result includes a description of the research trends during 1980-2008. The researchers found that inspections (code reviews) generally benefit software development and quality assurance.

Kononenko et al. studied code review processes of a large open-source project and investigated how developers perceive code review quality [21]. Data was collected through a survey on 88 developers and they found that review quality is associated with the thoroughness of the feedback, the reviewer's familiarity with the code, and the perceived quality of the code itself. Moreover, reviewers often find it difficult to keep their technical skills up-to-date, manage personal priorities, and mitigate context switching.

Bosu et al. identify what factors contribute to useful code reviews with an empirical study at Microsoft [9]. They also found that code review quality increases dramatically in the first year at the company, after which it stabilizes. Moreover, the quality of code review comments decrease when more files have been changed,

Security

The security perspective on code review is studied by di Biase et al. in a case study [11]. They found that only 1% of the code review comments address security issues. They also point out that reviews conducted by more than two developers are more successful at finding security issues.

Review-then-Commit vs Commit-then-Review

Rigby et al. compare two peer review techniques, review-then-commit (RTC) and commit-then-review (CTR), in open source development [25]. This can be compared with code reviews in the merge request-based model and trunk-based model respectively where the order of reviewing and committing differ. The results show that the main reason to adopt CTR is that there is no review interval and CTR is 2.2 times faster than RTC. They do not find a statistically significant difference in the number of defects found in both techniques and mention that for industrial purposes the CTR could be applied. The researchers conclude that it is unlikely that one technique will be clearly superior in all environments and further experimentation is needed, such as analyzing different variables.

Chapter 3

The Developers' Expectations on the Migration Towards Merge Requests

The goal of this chapter is to understand the benefits and challenges of working with the trunk-based model at Adyen. Also, we explore the expectations developers have on the merge-based development model.

At the start of the research at Adyen, the merge request model was still being developed and therefore not in use yet. This was ideal for us to conduct interviews before the migration to the merge request-based model to capture the experiences of the trunk-based model as well as the expectations of the merge request-based development model. In this chapter we will answer the first research question:

RQ1 What are the expectation of migrating from a trunk-based development model to a merge request-based development model at Adyen?

To answer this research question we define the following subquestions:

- RQ1.1** What benefits do developers see with working with the trunk-based model?
- RQ1.2** What challenges do developers face while working with the trunk-based model?
- RQ1.3** What do developers expect to change when migrating from the trunk-based model to the merge request-based model?
- RQ1.4** How do developers do their code reviews in the trunk-based model?
- RQ1.5** What do developers expect to change in code reviews when switching to the merge request-based model?

We will first provide the methodology, after which will we talk about the results. First we will talk about the code reviews in the trunk-based development model and the expectations of code reviews of the merge request-based development model. After that, we will talk about the development speed, followed by merge conflicts. We will then provide the results regarding security and safety of the codebase. Finally, we will talk about feature branches and conclude with an overall comparison of both development models.

3.1 Methodology

In this section, the methodology will be explained. First, we create and conduct interviews to explore the perceptions and expectations of the developers. After that, we survey developers to generalize our findings. A summary of the methodology is shown in Figure 3.1.



Figure 3.1: Methodology.

3.1.1 Interview Design

The interview was designed as semi-structured and consisted of five main parts, the introduction, participant information, trunk-based experience, merge request expectations, and concluding questions. The full script used for this interview can be found in appendix A.

In the introduction, the interviewer explained the purpose of the study, asked for permission to record the interview, and mentioned the interviewee would remain anonymous during the analysis and publications of the results.

The second part focused on the background of the interviewee. The main goal of this part is to get an overview of the experience in software development, time at the company, and the nature of the development work of the participant. Moreover, we would ask whether the interviewee had previous experience with the merge request-based model.

The next part is about the experience of the trunk-based model at Adyen. We would ask the interviewee about the challenges and benefits of this model and the code reviews that are being done.

If the participant had previous knowledge of the merge request-based model, we would ask what they see as benefits and challenges of that model compared to the trunk-based model. We would ask every developer to express their expectations of the merge request-based model since most developers were aware of the migration.

Finally, we would ask the interviewee which development model they would prefer at Adyen with the knowledge of both models before the migration. And to wrap up the interview, we would provide a summary of the points made by the participant and validate whether it is a fair summary and ask if the interviewee has something additional to add to the interview that was not discussed before.

3.1.2 Participant Selection

Participants for the interviews were randomly selected from a sample of around 515 people including all teams. There is a Mattermost¹ channel at Adyen that is called 'Development'.

¹mattermost.com

This channel includes all developers, team leads, and other employees that are interested in development-related topics. We started with one pilot interview, which we did not discard since the participant provided valuable information. In the first round of participant selection, we invited ten developers for an interview of which six were available and four did not reply. In the second round, we invited fourteen more developers, from which four developers were available and one was out of office, one was not available, and eight did not reply. Therefore, the response rate was 44%. Those eleven interviews were considered enough since the information provided gave theoretical saturation, which means that new information is not likely to emerge from further interviews.

The full list of the participants interviewed before the migration to the merge request-based development model is shown in Table 3.1.

ID	Software Engineering Experience	Time at Adyen	Team	Role	Experience in MR
P1	7y	3y	A	Statistic Analyst	no
P2	20y	7m	B	Java Back-end Engineer	yes
P3	4y	1y 6m	C	Java Software Engineer	yes
P4	12y	4m	D	Front-end Engineer	yes
P5	9y	1y	E	Java Developer	no
P6	1y 6m	1y 6m	F	Data Scientist	no
P7	12y	1y 5m	D	Java Developer	yes
P8	13y	1y 3m	G	Software Engineer	yes
P9	5y	1y 3m	H	Data Scientist	no
P10	8y	1y 6m	I	Security Specialist	yes
P11	14y	8m	D	Back-end Developer	no

Table 3.1: Interview participants.

3.1.3 Data Analysis

All interviews were conducted online over Zoom² since the study was done during the COVID-19 pandemic and everyone was working from home. This made it easy to record the interviews with Zoom. The recordings of the interviews were automatically transcribed with Otter.ai³ and corrected and analyzed with the ATLAS.ti⁴ software. Open coding was used to analyze the interviews. The transcript was broken up into related sentences and codes were applied to each informative group of sentences.

3.1.4 Survey Design

The survey is designed based on the results of the qualitative analysis. From the interviews, seven topics emerged that will be questioned in the survey. The goal is to generalize the

²zoom.us

³otter.ai

⁴atlas.ti

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS

findings of the interviews.

The survey has been created with LimeSurvey⁵, which is an online survey tool. The questionnaire has been divided into nine sections; Background Information, Code Reviews, Development Speed, Merge Conflicts, Code Quality and Safety, Feature Branches, Microservices, Development Models, and Ending Questions. Each section, except for the Background Information and Ending Questions, contains Likert scale questions where we ask the participant how much they agree with the statements.

First, a pilot has been done with one team. The feedback from this pilot has been used to improve the survey. Some questions had to be rephrased to become more understandable and some questions have been removed since the survey took longer than expected.

The survey has been distributed through the official mailing list of all developers within the company. There are approximately 500 developers on this list of which 32 completed the survey. A week later a reminder was sent in the Development channel of the online communication tool used within Adyen which contains around 530 people, from which most of them are developers. The survey was completed by 14 more developers, which sums up to 46 responses in total. This gives a response rate of 8.3%. The survey also got three partial responses, which are not completed for more than 50% but were not finished. These responses will not be used in the results.

There were developers from 35 teams who completed the survey. The number of years experience in software engineering ranged from zero to 21, with an average of nine years and a median of nine years. The time at Adyen ranged from half a month till ten years with an average of two years and three months and a median of one year. In total, 35 developers had previous experience with merge request-based development and eleven developers have only used the trunk-based development model.

The complete survey with all the questions can be found in appendix B. Whenever presenting evidence from the survey data, we show the percentage of participants that selected one of the items in the Likert scale as well as a miniature bar plot representing the entire distribution of answers. Bar plots contain six bars, in the following order: *strongly disagree*, *disagree*, *neutral*, *agree*, *strongly agree*, *does not apply*.

3.2 Code Reviews

In this section, we will first talk about the results of the interviews and survey on code reviews in the trunk-based model. After that, we will discuss the expectations of code reviews in the merge requests. In Table 3.2, we show the survey responses to questions regarding code reviews in the trunk-based model.

3.2.1 Code Reviews in the Trunk-Based Development Model

First of all, we asked the interview participants what they look for when doing code reviews. Developers try to find big mistakes in the code (P4, P5) and check the code style and readability (P3). Moreover, reviewers look at the code complexity to see if the code is

⁵limesurvey.org

Statement	Result
The reviewer has to convince the author to improve their code.	
There is enough time to complete code reviews before the release.	
It is a problem that developers sometimes have to wait too long for a review.	
It is useful to get reviews assigned from other teams.	
It is important for beginner developers to conduct code reviews as they will learn the best practices.	
Less experienced developers will learn more when reviews are mandatory for code to be merged to the master branch.	

Table 3.2: Responses to the survey (part 1).

as efficient as possible (P7). Developers also look at the context around the code changes (P4, P5, P10), for example, they look at which other features does the code change impact and the higher-level functionalities. The interviewees mention putting comments in the review on typos (P5) and looking at variable names to see whether they make sense (P4). Developers are testing the new feature locally and playing with it to see if it works (P3) and checking for null pointers (P3). One developer does not look at check-style (P5), since this is already done by the automatic tools. P4 explains in detail how they do their code reviews in three steps: *“For me, I do sweeps. I check the files first for big mistakes, big code smells, something I go by, like, what is going on here? Why is this? Then in the second sweep, I go a bit deeper. And check the context on what is this file? What is actually the entire file doing? Does this make sense this change in this context? ... And then the third sweep is just pickiness.”*

Moreover, a general remark on code reviews is that the main goal of code reviews is to produce good quality code (P7). This participant also stresses the importance of doing code reviews when working remotely to share each other’s opinions.

Benefits of Reviewing Code in the Trunk-Based Model

In the trunk-based development model, the code reviews are done after the merge to the master branch. The reviewer has to convince the author to improve their code if the reviewer disagrees with the changes being made. This is seen as a benefit of the trunk-based model because the reviews will provide more valuable information (P5). This participant explains

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS

this by the asymmetry in power between author and reviewer when doing reviews. With the merge-request model, the reviewer has all the power on deciding whether a commit will go into master or not and the commit is being gate-kept by the reviewer. Some people with strong opinions might take advantage of this power by placing insignificant comments, which results in changes not being committed for a long time. While in the trunk-based model, the commit is already live and the reviewer has to make a ticket to fix the commit and make effort to convince the author to change it. P6 mentions the comments that they place are not getting addressed by the author of the code. In the survey the participants were asked if the reviewer has to convince the author to improve their code. 56% agrees on this statement, whereas 15% is neutral and 13% disagrees (---■-).

Long Waiting Times for Code Reviews

A big challenge in the trunk-based model is that reviews are not done within the time limit (P1, P2, P3, P4, P5, P6, P8, P11). Developers contact their reviewers directly to ask for a review (P1, P6, P8, P10). As a reviewer, they contact the author for critical problems to understand what they are doing (P6). Developers say to have replaced the assigned reviewer when they do not respond in time (P1, P8). P1 said: *"It is rare to get both reviewers finished in less than let's say three days, it's rare."*

It is hard to always review on time (P2), especially when the reviewer has a day off or when they are busy. Getting a quick review is also a challenge (P8), for example, when this participant makes a bug fix, it is desired to get a quick review, but sometimes it takes days to get this review. P1 mentions that there is a lack of discipline among the developers to review each other's code changes quickly (P1). Developers mention they immediately do their reviews once they have free time (P2, P3, P4, P8, P9), and some always review within one day (P1, P8, P9). The long time waiting for a review is a big challenge (P1, P8, P10). 50% of the survey respondents disagree with the statement that there is enough time to complete the code reviews before the release, whereas 15% agree (-■---). Also, 50% of agree with the statement that it is a problem that developers sometimes have to wait too long for a review. 22% strongly agrees and only 7% disagree (-■■-).

Team Reviews

Each contribution should be reviewed by 2 developers which can be from any other team, but some participants prefer to only review their team members. The number of reviews required depends on each specific code change and developers with expertise in the parts that are changed should be assigned as reviewers (P10). Also, getting assigned to reviews from different teams is not worth it as the reviewer has to take the time to understand the other team's domain (P6). In their words: *"If I have to, for example, review somebody's ML code, understanding why they added this parameter and remove that parameter. It's just not scalable. It's just too much time I have to spend trying to understand the domain. It's not worth it."* 28% of the survey participants agree, 28% filled in neutral, 20% strongly agree and 20% disagree with the statement that it is useful to get reviews assigned from other teams (-■■■-).

Beginner Developers

The interviewees say that the trunk-based development model proposes some challenges for developers that just joined the company. The trunk-based model is not ideal for beginners when their contributions do not get reviewed (P2), because they will not learn from their beginner mistakes. Also, reality does not reflect the code review guidelines (P5) because, for example, the guidelines state that a reviewer should not place a comment without providing an alternative, but in reality, this does happen. One developer (P7) mentions giving constructive comments in code reviews since this participant already knows what the code is about. They also mention expecting good code reviews in return when this participant does good code reviews on others, especially beginners. The same interviewee mentions beginner developers need to conduct code reviews as they will learn the best practices. This participant also mentions that for more experienced developers to be reviewed becomes less relevant and it is better for them to conduct code reviews than to be reviewed. More specifically, P7 said: *“I really enjoy, you know, going to the real depths of what is happening and go to the most efficient things in mathematical terms, you know, big O notation. I love that part of algorithms and I’m always seeking those things. And my team, they’re really thankful, especially new people, that I do those remarks, that I take time to really review. And I expect the same in return and when I get these, it is really fruitful to have them. And I think at Adyen it works really well.”* 46% of the survey participants strongly agree and 41% agree with the statement that it is important for beginner developers to conduct code reviews as they will learn the best practices (---■). When asked whether less experienced developers will learn more when reviews are mandatory for code to be merged to the master branch, 39% agree, 24% strongly agree and 15% disagree. (---■).

Findings

- F1** Reviewers have to convince the author to improve their code.
- F2** Code is not reviewed fast enough.
- F3** Code reviews are considered very important for beginner developers.

3.2.2 Code Reviews Expectations of the Merge Request-Based Model

Since most developers have had previous experience or knowledge on merge requests, they expressed their expectations of this development model for Adyen. In this subsection, we will discuss the expectations of the merge requests on code quality, expected challenges, and expected benefits. In Table 3.3, we show the survey responses.

Code Quality

The largest expected benefit from the merge request-based model is that the code quality will improve and that reviews in general prevent mistakes (P6, P11). The review quality

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS






Statement	Result
Review quality and therefore code quality will improve.	
Merge Requests makes the reviewer see a bigger picture of the review.	
Reviewers will often block the merge by placing insignificant comments.	
It would be useful if automatic reminders are sent to the reviewers.	
Reviewers should face some sort of consequences (no heavy punishment) when they do not review in time.	

Table 3.3: Responses to the survey (part 2).

and therefore the code quality is expected to improve because the reviews will contain all relevant commits to the new feature (P11). Another interviewee (P3) expects that the code quality will depend on the reviewer instead of the author and reviewers should take the responsibility to improve the code quality. This participant also thinks the quality of the review depends on how familiar the reviewer is with the code.

The code quality is expected to improve in the long term (P6), as more experienced developers will review the less experienced developers and share their knowledge on best practices and teach them how to write better code. In their words: *“I hope I am using all the good software engineering practices, but there are quite some more senior people, more experienced people that can foresee certain side effects that your code can have. So in that aspect, I think I do can learn a lot. So I think in terms of quality, long term speaking, it will, for sure, would increase the quality.”* One developer (P7) says to enjoy doing reviews and sees benefits because developers can teach each other and learn from each other when doing reviews.

Some interviewees expect no difference in the quality of the reviews (P2, P10). P2 said: *“So the quality of the reviews, I do not see a difference. I think everyone that is doing reviews is going to go the same way right now. ... And well, code quality. And I think that is the same scenario, I think it is going to be the same because people are doing the review right now.”* One other developer (P6) says the quality of a review will depend on the quality of the commit and the mindset of the reviewer, which also holds for the reviews in the trunk-based model.

However, some participants were less optimistic about the proposed merge request-based model. They dislike the requirements set for doing reviews and that forcing someone to do a review in the merge request model is a bad idea since they can just click on merge (P5).

A trade-off is expected to be made between maintaining the development speed and giving good reviews that will prevent bad code (P8). An alternative is suggested to ensure code reviews, which is to get volunteers to spend a few hours reviewing all the unreviewed code before the release (P1), and having five volunteers would be enough to review everything each week according to this developer. 37% of the survey respondents agree with the statement that the review quality and therefore the code quality will improve in the merge request-based model. Even 22% strongly agree and 15% disagree (■■■■).

Responsibility

Another reason for the migration was to stimulate developers to review faster (P2, P3, P4, P8), which is seen as a benefit because the model forces developers to review (P3, P8). Code reviews are a shared responsibility between the author and the reviewers (P1, P3) and the shared responsibility of the author and reviewer is a benefit of the merge request model (P3). In their words: *“It feels like the reviewers will also have the responsibility to make sure that code quality is good enough. Because after they approve, the code is merged to master and then released. So, it is kind of sharing the responsibility as well.”* One developer (P4) mentions that reviewers should take the time to talk to the authors when doing code reviews.

Better Overview

Developers mention that a merge request helps when reviewing because you see a bigger picture of the feature instead of reviewing every commit as in the trunk-based model (P5, P11). The interviewees expect developers to get more familiar with the code as they will participate more in code reviews (P3, P5), which will benefit the developers to collaborate easier. However, one interviewee (P11) mentions that in the trunk-based model you can tag your commit with the previous review ID to get a better picture of the new feature. When asked in the survey if merge requests would make the reviewer see a bigger picture of the review, 28% responded neutral, 26% agreed and 22% disagreed (■■■■).

Gate-keeping

An expected challenge of the merge request model is the gate-keeping of the reviews (P2, P5, P8) because the reviewer has the power to prevent code from going live. One developer (P5) mentions that this will become a big challenge when reviewers block the merge when there are, for example, only small typos in the commit. In their words: *“But in a review, this is gate kept, right? So in principle, you cannot merge it on the master until somebody approves it and this creates a power asymmetry that a certain type of person takes advantage of. And the type of person that takes advantage of that is somebody with strong opinions. And typically, people have strong opinions about insignificant things.”* This opinion is very divided in the survey as 30% of the survey respondents disagree and 30% agree with the statement that reviewers will often block the merge request by placing insignificant comments (■■■■).

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS

Review Reminders

The interviewees foresee a challenge in the merge request-based approach that reviews will not be done on time and proposed solutions for this, such as specific time frames for developers when they have to do code reviews (P6, P8, P10). This way, developers are forced to review in a structured manner, and will be done in time. Another solution is sending automatic reminders to reviewers when a commit is waiting for a review (P8). When someone accidentally forgets to do a review, they will get a notification and the merge request will not be open for too long. This participant says that the use of tooling will also build discipline. 41% of the survey respondents agree and 35% strongly agree with the statement that it would be useful if automatic reminders are sent to the reviewers. 17% filled in neutral and only 7% disagrees (▬▬▬▬).

Consequences for slow reviews

Since code reviews will become more important in the new development flow, some developers want to see consequences for reviewers that do not review in time. A benefit of the merge-request model is that developers are relieved of the responsibility to review on time (P1). In the trunk-based model you have to review before the release, which happens every week, whereas, in the merge request model, the reviewer can take the time they need.

However, developers (P1, P7) want reviewers to face consequences when they are not reviewing to stimulate faster reviews, for example, a *Hall of Shame* for people who lack the discipline to review (P7). 33% of the survey respondents agree with some sort of consequence (no heavy punishment) when the reviewer does not review in time. 26% is neutral and 15% disagrees (▬▬▬▬).

Findings

- F4** Developers expect that mandatory code reviews will help beginner developers to learn best practices.
- F5** Review quality and therefore code quality are expected to improve.
- F6** Developers would like to have automatic reminders sent to the reviewers.

3.3 Development Speed

This section is dedicated to the results regarding the development speed. We will discuss the overall velocity, development time, and release cycle. In Table 3.4, we show the survey responses.







Statement	Result
With the merge request-based model, the velocity of producing new code will decrease.	
With the merge request-based model, developing time gets wasted by waiting on merge request approvals.	
The current release cycle is too short.	
With the merge request-based model, a feedback cycle of one week will be too short.	
It would be better to extend the release cycle or postpone the release until everything has been reviewed.	

Table 3.4: Responses to the survey (part 3).

3.3.1 Overall Velocity

First of all, a benefit of the trunk-based model is that iterations go fast (P7, P9), which can be especially helpful for emergency patching (P3, P9) because fixes can be pushed immediately to the master branch without waiting for another person to review them. One developer (P1) says that they like the ability to test their contributions quickly in the master branch to see whether all the jobs and integration tests pass. The expectation of the merge request approach is that the velocity of producing new code will decrease (P1, P4, P5, P6, P7, P8, P10). One interviewee (P4) expects that most of the delay will be a result of the lack of reviewers, while another (P1) mentions that it requires more actions for a contribution to be merged to the master branch because the author has to create a merge request, then a reviewer gets assigned and needs to approve the merge request.

At Adyen, it is expected of developers that they expose their code early and to launch fast and iterate, but developers (P1, P8, P11) expect that maintaining this philosophy is going to be a challenge with the new model as code is only merged when a feature is complete and reviewed. P8 said: *“That will be the challenge for us to maintain our speed by not blocking other people committing code, and giving good reviews that prevent us from committing bad code that will break others dependencies. And the challenge will be to maintain the speed that we have right now.”* On the other hand, some developers (P2, P11) do not expect the speed to decrease significantly when migrating to the merge request-based model. In the survey the participants were asked whether with the merge request-based model, the velocity of producing new code will decrease: 34% agree and 31% disagree (.

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS

3.3.2 Development Time

Developers perceive that the time between the creation of the review and the review being done depends on the size of the code changes (P2, P4, P5, P11), where smaller changes are reviewed faster.

Developers expect a decrease in development speed for the merge request-based model because more communication is necessary (P3), developers will be waiting on the merge request to be approved (P7), and reviewers will not mark themselves as *'out of office'* when they are on leave (P2). So, when they get assigned to a review they will not be able to review until they are back and this will delay someone else's work. 28% of the survey respondents disagree with the statement that with the merge request-based model, developing time gets wasted by waiting on merge request approvals. 24% strongly disagree and 20% agree (■■■■).

3.3.3 Release Cycle

At Adyen, there is a release cycle of one week, which means that every piece of code should be reviewed within a week. Some developers (P2, P6) perceive that one week is too short to properly test everything that is in the master branch before the release branch goes to production. In addition, developers (P1, P7) expect challenges with the feedback cycle for the merge requests, as one week will be too short to produce code, give feedback and test everything properly. 43% of the survey participants disagree with the statement that the current release cycle is too short. 24% strongly disagree and only 4% agree (■■■■). Similar results hold for the expectations for the merge request-based model, although it is expected that the merge request-based model will require a slightly longer feedback cycle. 37% disagree with the statement that with the merge request-based model, a feedback cycle of one week will be too short. 15% strongly disagree and 15% agree (■■■■).

Trunk-based development allows developers to commit quickly but does not ensure all code to be reviewed. The merge request-based model guarantees 100% review coverage but generally has longer release cycles. Therefore a trade-off has to be made between speed versus review coverage as mentioned by one developer (P6) and a proposed solution is to extend the release cycle (P1, P2), as they would rather extend the release cycle or postpone a release until all the code has been reviewed than to have each commit waiting to be merged to the master. This means that they prefer the trunk-based model and only plan a release when every contribution has been reviewed instead of using merge requests. 39% of the survey respondents disagree with the statement that it would be better to extend the release cycle or postpone the release until everything has been reviewed. 24% strongly disagree and 17% agree (■■■■).

Findings

- F7** Developers do not expect that time will get wasted by waiting on code reviews.
- F8** The release cycle is not considered too short and is also not expected to be too short when using merge requests.
- F9** Developers do not want to postpone the release until everything is reviewed.

3.4 Merge Conflicts

In this section, we will discuss the perceptions of merge conflicts in the trunk-based model and the expectations of merge conflicts in the merge request-based model. In Table 3.5, we show the survey responses.






Statement	Result
In the trunk-based model, merge conflicts happen often.	
Resolving merge conflicts is a time consuming task.	
Merge conflicts are easy to fix.	
There will be more merge conflicts in the merge request-based model.	
Merge conflicts are meaningful and help the code to get better.	

Table 3.5: Responses to the survey (part 4).

3.4.1 Perceptions of the Trunk-Based Model

In the Adyen repository, code can get mingled which causes merge conflicts (P4). However, the benefit of the trunk-based model is that merge conflicts do not happen that often (P3, P4, P5, P10), which one developer (P4) explains as a result of the configured Git commands by Adyen. The survey participants were asked whether in the trunk-based model, merge conflicts happen often. 48% disagree, 24% strongly disagree and 7% agree (■■■■■).

Merge conflicts are not considered a large problem in the trunk-based model, but it is only a bit annoying because developers need to pull and push again and communicate with

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS

other developers to discuss which changes should persist (P3, P10). The survey participants were asked whether resolving merge conflicts is a time consuming task. The expectations are divided as 33% disagree and 33% agree (■■■).

Also, merge conflicts are considered easy to solve (P2, P3), especially when the merge conflicts are within the same team (P3) because it will be easy to reach out and fix them quickly. But when other teams are involved, this becomes annoying. In their words: *“I think it depends on the code itself if it is a code that is used or shared among all the teams that can be annoying like different people will change the code. But if it is code just about one team, then it is easy to reach out and communicate. It is easy.”* 43% of the survey respondents agree with the statement that merge conflicts are easy to fix. 33% is neutral and 15% disagree (■■■).

3.4.2 Expectations in the Merge Requests

One developer (P4) mentions that at a previous company, working with the merge request model they experienced fewer merge conflicts in 2 years than in one month working with the trunk-based model. Nevertheless, more merge conflicts are expected in the merge request-based approach (P1, P2, P3, P6, P10, P11) and one developer (P1) is especially afraid of merge conflict loops. Since there are hundreds of developers working in the same repository, someone else has likely merged their code before your changes have been reviewed. This will also slow down the development process as the pipeline should be run again every time which can take around half an hour per cycle. In their words: *“Because the code that we push, will be only merged after some time after it is reviewed. And at times it happens that other parts of code that you were using, could have changed, for example. And then, you need to push again. It is like chasing the turtle if you know what I mean. I am a bit afraid of that.”* The same developer suggests implementing an automated merge when the review is approved to avoid longer waiting times and reduce the risk of merge conflicts. Another developer (P10) says that to reduce the number of merge conflicts within the merge request-based model, Adyen needs to implement faster reviews and faster development. 41% of the survey respondents expect there will be more merge conflicts in the merge request-based model. 20 % disagree with this statement (■■■).

One possible expected benefit of the merge request model is that the merge conflicts that occur will be meaningful and help the code to get better (P4). However, this could not be confirmed with the survey as 26% disagree, 22% agree and 20% strongly disagree with the statement that merge conflicts are meaningful and help the code to get better (■■■).

Findings

F10 Merge conflicts do not happen very often.

F11 Developers expect more merge conflicts to occur in the merge request-based development model.

F12 Merge conflicts are considered easy to fix.

3.5 Security & Safety

One of the goals of code reviews is to catch potential bugs and errors. In the trunk-based model, those errors are already in the master branch at the time of the review and should be fixed before the release date. For merge requests, they can be prevented from going to the master branch as the review is done before the merge. In this section, we will discuss the perceptions of the interviewees and survey participants on the security and safety of the codebase at Adyen. In Table 3.6, we show the survey responses.






Statement	Result
Bugs in production could have been prevented when the responsible code was reviewed.	
The cost of fixing the bug is higher than preventing by doing a proper code reviews.	
I have felt scared to break the system when committing code to master .	
Testing code is more reliable than code reviews.	
A benefit of the merge request-based model is that there are less bugs and errors in the production code.	

Table 3.6: Responses to the survey (part 5).

On a small note, a security specialist (P10) compliments the developers for being aware of security issues. This person is surprised by how cooperative developers are as this participant experienced a lot of friction between other teams and security before. The same interviewee also mentions that Adyen does background checks on new hires to reduce the risk of malicious comments.

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS

3.5.1 Error Prevention

Developers perceive multiple benefits of the trunk-based model regarding error prevention because one developer (P7) mentions that it is easier to spot bugs and errors when they are on the master branch. As a consequence, bugs are detected earlier and also fixed sooner with the trunk-based model. Moreover, the trunk-based model works well for small teams because generally with smaller teams you are more exposed to errors and you can fix them quite fast (P2).

On the other hand developers (P1, P11) mention cases in which errors could have been prevented if the code was reviewed before the merge to master, as merge requests will give reviewers the time to properly review and test the code changes (P2). 43% of the survey respondents agree with the statement that bugs could have been prevented when the responsible code was reviewed. 24% strongly agree and 24% disagree (■■■).

Developers (P2, P6, P7, P8, P10) mention that the effort and costs of fixing a bug, error, or vulnerability in the master branch exceed the costs of preventing it in the first place by doing a proper code review, which will also result in fewer bugs and errors in the merge request based model. 35% of the survey respondents agree with the statement that the cost of fixing the bug is higher than preventing it by doing a proper code review. 24% strongly agree and 17% disagree (■■■).

Participants were also concerned about the security of the codebase and maintaining a secure repository is a challenge. Developers (P3, P10) felt scared committing code to the master branch when they started at Adyen because it might break the system and influence every developer within Adyen. Moreover, another developer (P8) mentions how easy it is for one developer to affect all users without meaning to. 48% of the survey respondents agree to have felt scared to break the system when committing code to master. 15% strongly agree, 17% filled in neutral and 15% disagree (■■■).


3.5.2 Reviews Versus Tests

Another debate occurred during the interviews between the benefits of code reviews and tests. Although most errors get caught before the release through testing and doing a beta release first (P9, P10), not all code is covered during testing (P3), so reviews are necessary. Developers (P4, P10) say that a benefit of the merge request model is that humans will check the code rather than only running integration tests. They say that humans are still the best control for code reviews since the tests can be fooled and humans understand the business scenarios, especially for the front-end reviews (P4) since tests cannot cover all front-end code. P10 said: *“A lot of benefits from a security perspective. First of all, you have a good code review from a human being and that is, I think, very valuable. Imagine a senior person reviewing code will criticize it not only from a code quality perspective but also from fundamental flaws, logical issues, and vulnerabilities being introduced. I think that is the biggest value, we get out of a merge request model that another set of human eyes and brain looking at the code you wrote.”* On the contrary, one developer (P9) believes that testing the code is more reliable than someone doing a code review. 33% strongly agree with the statement that testing code is more reliable than code reviews. 28% agree and 13% disagree

with this statement ().

Multiple developers (P6, P7, P8) see a benefit in running all the tests before the review and the merge. This way you will be certain that all tests passed when you start the review and when you merge the code to the master branch. Moreover, having release managers increases the security of the codebase (P9, P10). One feature developers (P1, P9) still miss is the ability to test everything locally as this will ensure that the tests pass before committing your code.

3.5.3 Expectations of the Merge Requests

Merge requests are expected to decrease the number of occurring bugs and errors. The interviewees (P2, P4, P8, P10) expect the merge request-model to increase the safety within the codebase, as it will mitigate the possibility of one person bringing down the whole platform (P10) and the system is expected to be more reliable as it is expected that the code will be better tested (P6). In addition, merge requests are especially useful for code changes that include critical parts of the system or for new independent features as it allows the developer to experiment in its own branch (P7). 39% of the survey respondents agree with the statement that a benefit of the merge request-based model is that there are fewer bugs and errors in the production code. 20% strongly agree and 15% filled in neutral ().

Findings

- F13** Bugs could have been prevented if the responsible code was reviewed before merging.
- F14** The cost and effort of fixing a bug are higher than preventing it.
- F15** Developers feel scared to break the code when pushing directly to master.
- F16** There are fewer bugs expected in the merge request-based development model.
- F17** Tests are considered more reliable than code reviews.

3.6 Feature Branches

In this section, we will discuss the perceptions of the feature branches in the trunk-based model and the expectations of branching in the merge request-based model. In Table 3.7, we show the survey responses.

3.6.1 Lifetime of the Feature Branch

If a developer at Adyen was building a new feature and it was not finished, the feature should be disabled with a feature flag. Developers were not able to maintain a feature branch for

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS








Statement	Result
Feature branches can only be maintained for one week is a disadvantage of the trunk-based model.	
The commit history will improve in the MR model as commits to the same feature will be grouped.	
The trunk-based model simplifies the process of fixing bugs as it is easy to pinpoint where the error is introduced and revert that commit.	
Saving code remotely on a feature branch without influencing other developers is a benefit of the MR model.	
Developers will abuse the feature branches in the merge request-based model by developing for too long in one branch.	
When the merge-request is too large, the review quality will decrease as it is harder to do a proper review.	
With the merge request-based model, the system should give warnings when feature branches are too long lived.	

Table 3.7: Responses to the survey (part 6).

longer than one week. This means that your changes have to be committed and pushed to master and they have to be live within one week. This is something that developers (P2, P3, P5) disliked about the trunk-based model. P2 explains how they can still maintain a branch with some extra work: “*You can just check out a new branch from your old branch that you cannot push to anymore. But it is still like you are kind of doing a trick just to have your code saved in a branch.*” When asked to the survey participants whether it is a disadvantage of the trunk-based model that feature branches can only be maintained for one week, 35% agree, 22% filled in neutral and 20% disagree (■■■■).

3.6.2 Commit History

There are hundreds of developers at Adyen working in the same branch at the same time. This proposes a big challenge in itself and can lead to a cumbersome repository. Developers (P4, P11) perceive commits easily getting mixed as the order of commits to the master branch is not sorted by feature, therefore it is expected that the commit history will be better in the merge request-based model as commits to the same feature will be on one branch and merged together to the master branch. Therefore related commits will show up together in the master branch. However, one developer (P5) foresees a coordination issue on which

branch to merge first with the merge request model. This participant expects a lot of merge requests to come in at the same time and is wondering who is going to decide which branch will be merged to the master first. 46% of the survey respondents agree with the statement that the commit history will improve in the merge request-based model as commits to the same feature will be grouped. 22% filled in neutral and 15% strongly agree (■■■■).

3.6.3 Versions

One developer (P7) mentions that a result of the trunk-based model is that you have fewer versions in comparison with the merge request-based model, where everyone can work on a separate branch. They mention that fewer versions are a benefit of the trunk-based model as it simplifies the process of fixing bugs because it is easy to pinpoint where the error is introduced and revert that commit. 33% of the survey respondents agree with the statement that the trunk-based model simplifies the process of fixing bugs as it is easy to pinpoint where the error is introduced and revert that commit. 30% filled in neutral and 22% disagree (■■■■).

3.6.4 Expected Benefits

Feature branches introduce some expected benefits to the merge request-based development model. Most developers (P3, P4, P5, P7, P8, P11) mention that the ability to use a feature branch is a benefit of the merge request-based model because they are able to test their code in their local branch (P5, P8), code can always be saved remotely on a branch without influencing others (P5), and it allows developers to work at their own pace without influencing others because everyone can work on an independent branch (P4, P7). P5 said: *“I expect people would prefer the merge request model, not because of the actual merge request, but simply because it is on another branch. And that means that you can just push dangerous code into the remote, without regard if it’s right, and then test there. And I think that can make us a little bit more agile because you are not paralyzed by this fear, you can just submit it.”* 43% of the survey respondents agree with the statement that saving code remotely on a feature branch without influencing other developers is a benefit of the merge request-based model. 24% filled in neutral and 17% strongly agree (■■■■).

3.6.5 Expected Challenges

An expected challenge is that feature branches might be too long-lived (P7, P11). One developer (P7) explains that the trunk-based model requires a lot of planning because every week there is a beta release on Monday and you need to ship your code on Friday and live release on Thursday and your fixes need to be committed on Wednesday, whereas the merge request-based model allows feature branches and you can just commit your changes on your feature branch without worrying about the release. Also, this developer expects that with the feature branches in the merge request model, feature switches as implemented in the trunk-based model will not be abused so much anymore, meaning that unfinished code will not be pushed to the master branch. However, developers (P7, P11) fear that developers might abuse the feature branches in the merge request-based model by developing for too long in

3. THE DEVELOPERS' EXPECTATIONS ON THE MIGRATION TOWARDS MERGE REQUESTS

one branch without merging to master. P7 said: *“I would say that the biggest challenge, like always, is not about the machines, it is about people. I think, if people start abusing feature branches that are long-lived, that might be a problem.”* 37% of the survey respondents disagree and 24% agree with the statement that developers will abuse the feature branches in the merge request-based model by developing for too long in one branch (■■■■).

As a result, too long-lived feature branches can result in larger merge requests, which can be a challenge for the review quality (P11) because when the feature is larger or there are more code changes it becomes more difficult and painful to review. 59% of the survey respondents strongly agree and 24% agree with the statement that when the merge request is too large, the review quality will decrease as it is harder to do a proper review (■■■).

One interviewee (P7) provides a solution to those abused feature branches and would like the system to give warnings when feature branches are too long-lived. 41% strongly agree and 37% agree with the statement that with the merge request-based model, the system should give warnings when feature branches are too long-lived (■■■).

Findings

F18 The commit history is expected to improve.

F19 Review quality is expected to decrease if the merge request is too large.

F20 Saving code remotely is expected to be a benefit of the merge request-based model.

F21 Developers would like the system to give warnings when feature branches are too long-lived.

3.7 Development Models

The development model preferences of developers will be discussed in this section. In Table 3.8, we show the survey responses.

3.7.1 Trunk-Based versus Merge Requests at Adyen

For Adyen specifically, challenges are expected when switching context or migrating models (P1, P3, P7, P8) and convincing the developers who are in favor of the trunk-based model (P4). Developers (P1, P2) expect that switching development model, is bad for productivity, however, they expect the migration to be a learning curve and after some time when everyone gets used to the new model, there will not be much delay anymore. P1 said: *“In my opinion, context switching is never good. Like, you can get used to it to some degree, but in my opinion, it will always be the case that the more context switching you are doing, the less productive you are. You can only mitigate it with getting used to it.”* One developer (P6) assumes that everyone has a shared sentiment whether to move to merge request or







Statement	Result
For Adyen, using the MR model has more benefits than the trunk-based model.	
Every team should decide for themselves whether to use to trunk based model or the MR model.	
A hybrid model such as the MR model allowing direct commits would be preferred over using the MR model.	
Senior developers should have the ability to merge directly without a review.	
The trunk-based model is preferred over the MR model for startups and fast developing codebases.	
The MR model should be used for mature codebases and high impact projects.	

Table 3.8: Responses to the survey (part 7).

remain the trunk-based model, while others mention to have been unaware of the reason for the migration (P9) and are skeptical about the reviews and mentions being uncertain about what Adyen is trying to accomplish with reviews (P5).

The interviewees (P2, P3, P4, P6, P8, P9, P10, P11) expect that the merge request development model would be better than the trunk-based model for Adyen at this time. Developers are expected to enjoy and benefit from the merge request-based model (P6, P7) and it is very useful for onboarding new developers to implement the industry standards of software development (P4), which is merge requests. 39% of the survey respondents agree with the statement that for Adyen, using the merge request-based model has more benefits than the trunk-based model. 35% is neutral and 17% strongly agree (-■■■-).

3.7.2 Optional Merge Requests

Not all interviewees are convinced to use the merge request-based model. Developers (P1, P6, P10) suggest making the merge request optional since sometimes a developer wants to quickly fix some small thing and it would not make sense to create a merge request. One developer expects these merge requests to become a sort of spam (P6), therefore it is best to let every team decide for themselves what way of development works best for them (P1). 28% of the survey respondents disagree, 26% strongly disagree and 24% agree with the statement that every team should decide for themselves whether to use the trunk-based model or the merger request-based model (■■■■-).

3.7.3 The Hybrid Model

A third option, the hybrid model is proposed by some participants. One developer (P5) would rather see a hybrid model that would look like the trunk-based model with feature branches and merge requests only as a discussion tool, but without requiring a review to merge the code changes. This can also be described as the merge request model with direct commits. 32% of the survey participants filled in neutral with the statement that a hybrid model such as the merge request-based model allowing direct commits would be preferred over using the merge request-based model. 30% agree and 15% disagree (.-■■-).

Another solution to the hybrid model is to allow only senior reviewers to merge directly. One interviewee (P10) mentions having already switched to the merge request model because this developer is working in the security team, which also has a separate repository. In this team, they use a hybrid model as certain team members are allowed to commit directly to the master branch. This person would rather have only senior folks have the ability to merge directly, however, this proposes an extra risk to security and it would be hard to determine with hundreds of developers which developers would get access to direct commits. For the main repository, this person thinks it is too sensitive to have, but for other repositories, it could benefit the development process. In their words: *“Go with like a hybrid, where you have certain senior folks in your team who can commit directly to the master branch in your merge request model and then have merge request model for other members of the team who are relatively junior.”* This opinion is not shared with the survey respondents as 37% strongly disagree and also 37% disagree with the statement that senior developers should have the ability to merge directly without a review (■■-.-).

3.7.4 Why Trunk-Based or Merge Requests?

The best choice between the trunk-based model and merge request-based model depends on the maturity of the project according to the interviewed developers (P6, P7). The trunk-based model is a good fit for startups and fast-developing codebases and therefore was perfect for Adyen so far. However, as Adyen is reaching some stage of maturity, they believe that the merge request is a better option for the company to make the platform as reliable as possible. 28% of the survey participants disagree and 24% agree with the statement that the trunk-based model is preferred over the merge request-based model for startups and fast developing codebases (■■■■-).

Also, participants (P6, P9, P10) claim that the merge request-based model is preferred for more mature and higher impact projects, where breaking the code has high consequences and costs. P6 said: *“There is no silver bullet to it, I would say there is not always merge request or always trunk-based, I think it should be weight, with considering the different things like the maturity, the type of impact that you can make, the team members, the size of the team, and the speed of development.”* 35% of the survey participants agree with the statement that the merge request-based model should be used for mature codebases and high-impact projects. 22% disagree and 15% strongly agree (.-■■■).

Findings

- F22** Developers do not want senior developers to have the ability to merge directly without a review.
- F23** The trunk-based development model is not considered to always be the best option for startups and fast-developing codebases.
- F24** The merge request-based development model is considered to be preferable for mature codebases and high-impact projects.

Chapter 4

Differences in How Developers Review Code in Trunk-Based and in Merge Requests

The goal of this chapter is to explain with qualitative data about code reviews in the trunk-based development model and merge request-based development what changes when migrating models. Moreover, we will do a quantitative analysis of code reviews of both models where we categorize code comments and compare the categories with a similar study done by Bacchelli and Bird [4] at Microsoft. In this chapter we will answer the second research question:

RQ2 What has changed in code reviews after migration from trunk-based to merge request-based at Adyen?

To answer this research question we define the following subquestions and for each subquestion compare the trunk-development model with the merge request-based model:

RQ2.1 What are the motivations for code review at Adyen?

RQ2.2 What changes in the time for a review to be closed when migration from the trunk-based development model to the merge request-based development model?

RQ2.3 What changes in the number of comments in the reviews?

4.1 Methodology

This chapter consists of two main parts, the qualitative analysis of the code review comments and the quantitative analysis of code reviews.

4. DIFFERENCES IN HOW DEVELOPERS REVIEW CODE IN TRUNK-BASED AND IN MRS

4.1.1 Qualitative Analysis on Code Comments

For the qualitative analysis, we manually categorized code comments into the nine categories described by Bacchelli and Bird [4]. The categories are listed and explained in Table 4.1

Category	Description
Code Improvement	Comments about code in terms of readability, commenting, consistency, dead code removal, etc.
Understanding	Comments where more explanation on the changes is requested.
Social Communication	Social talk between the author and reviewer that has nothing to do with the code changes.
Defects	Comments to find defects.
External Impact	Comments on the code that will impact other systems or other parts of the code.
Testing	Comments about test code or more tests are requested.
Review Tool	Comments regarding Upsource or GitLab, such as links to other reviews and merge requests.
Knowledge Transfer	Comments from a learning perspective for both the author and the reviewer on APIs usage, system design, best practices, etc.
Misc.	Any other comment that could not be categorized in the ones above.

Table 4.1: Code Comment Categories.

First of all, we selected discussions from the review tool that was used in the trunk-based model from the period of 24 January 2018 till 27 July 2021. There were a total of 96,402 discussions at the moment of data collection, from which 94,161 discussions consists of at least two comments. These discussions are included in 43,249 reviews from which 38,589 reviews have at least two comments. The total number of comments is 152,678. For our data analysis, we randomly selected 200 discussions with at least two comments, which resulted in a total of 518 code comments. This means that the confidence level is over 95% with a confidence interval of five.

In the merge request-based model we found 1658 merge requests between 9 March 2021 and 1 October 2021. There were a total of 423 reviews from which 368 reviews contained at least two comments. There are a total of 2662 code comments. We sampled code comments from 75 reviews which resulted in 477 code comments. This also gives a confidence level higher than 95% with a confidence interval of five.

4.1.2 Quantitative Analysis

To understand what the actual impact of the migration is on code reviews, we did a quantitative analysis of the code reviews before and after the migration. We gathered data using the APIs from Upsource and GitLab to request information about the code reviews and merge requests that have been created at Adyen. We then compare the review times and number of code comments per review in both models to see if a significant change occurred during the migration.

4.2 Qualitative Analysis on Code Comments

In this section, we will present the results of the qualitative analysis of code review comments. Figure 4.1 shows the proportions of comments by category of reviews in the trunk-based development model at Adyen, the merge request-based development model at Adyen, and reviews done at Microsoft studied by Bacchelli and Bird [4].

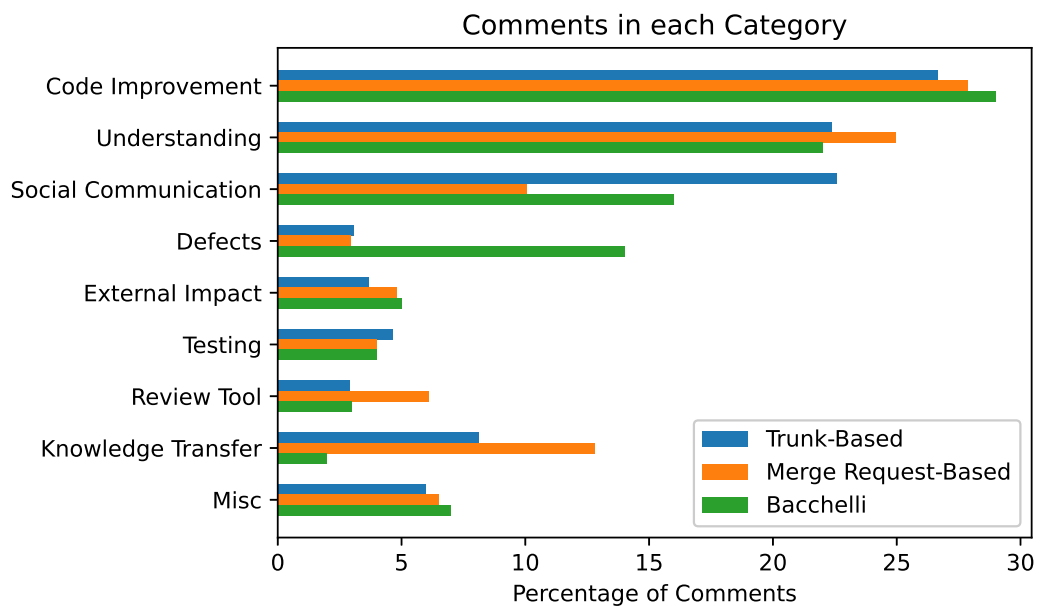


Figure 4.1: Proportion of comments by category.

From the results, we can see that for all models, the largest category of code comments is about code improvements, followed closely by the category of understanding.

4. DIFFERENCES IN HOW DEVELOPERS REVIEW CODE IN TRUNK-BASED AND IN MRS

There is a significant decrease in comments for social communication after the migration from the trunk-based development model to the merge request-based development model. Bacchelli and Bird report a lot more comments on defects, while at Adyen, both models do not contain as much. Compared to Bacchelli and Bird there are much more comments on knowledge transfer, especially in the merge request-based model. Also, in the merge requests, there are more comments placed regarding the review tool compared to both the trunk-based model and Bacchelli and Bird. Most of those comments were related to the migration itself, since a new review tool is used to review the code. The number of comments regarding the external impact and testing is similar across all three models.

Findings

- F25** The most comments in reviews are about code improvement and understanding.
- F26** Social communication has decreased during the migration.
- F27** More comments to transfer knowledge are placed in merge requests.

4.3 Quantitative Analysis: Review Times

First of all, we compared the time it takes for a review to be closed in both models. Table 4.2 shows some statistics on both models. Surprisingly, there was one review in the old model that has been closed after 1034 days. We assume that this is an incident and therefore the more meaningful statistic to look at is the median time it takes to close a review, which is significantly less in the merge request-based model. We filtered out all the reviews and merge requests closed within 10 minutes as it is very unlikely that someone did a proper review here. The results are visualized with a violin plot in Figure 4.2.

	Trunk-Based	Merge Requests
max	1034 days	100 days
mean	19 days 18 hours	2 days 17 hours
median	5 days 1 hour	16 hours
standard deviation	52 days 1 hour	6 days 13 hours

Table 4.2: Times between the creation of the review and the closing of the review.

Since there are many outliers in both models, we show the same violin plot in Figure 4.3. This violin plot is a zoomed-in version up to 33 days and shows that there are more reviews in the trunk-based model that take longer to complete.

To visualize these findings better and show the percentages of reviews closed within a certain timeframe, we created histograms that compare both models. Figure 4.4 shows the

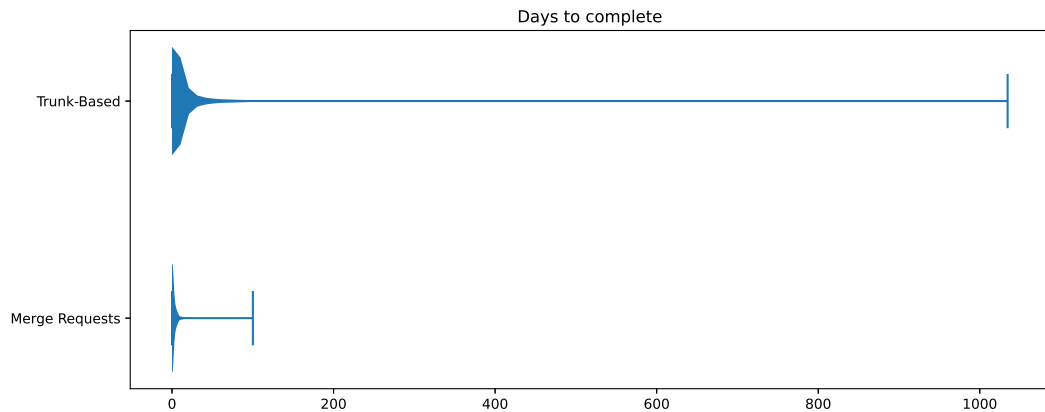


Figure 4.2: Days to complete.

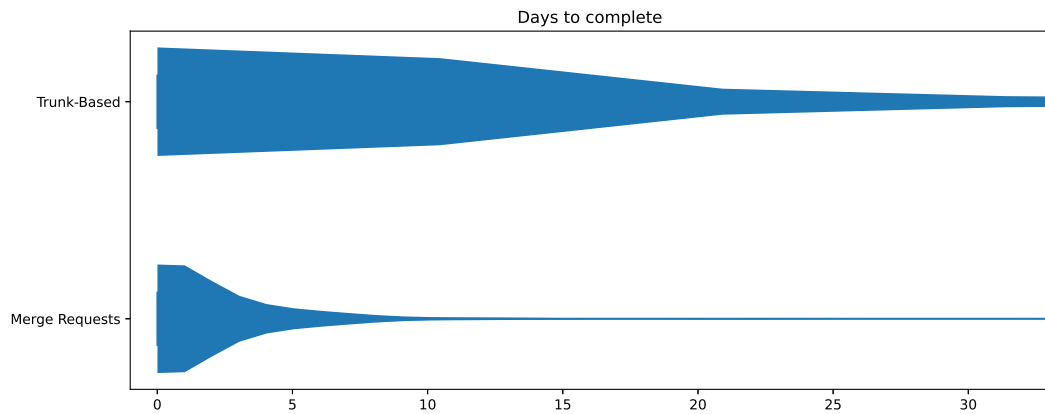


Figure 4.3: Days to complete.

distribution of the completion times of all reviews of both models in percentage. Clear to see is that almost all reviews in the merge request-based model are done within the first bin of the histogram, while there are reviews in the trunk-based model still closed after a few months.

Since there are outliers mainly in the trunk-based model we visualize in Figure 4.5 the completion times of all reviews that have been closed within a month. In the merge request-based model, 62% of the comments are closed on the first day, where this is only 19% for the trunk-based model. For merge requests, similar results were found by Gousios et al. [14], who found that 60% of merge requests on GitHub are merged or closed in less than a day. Sadowski et al. [28] found at Google, 70% of changes are committed less than 24 hours after they are mailed out for an initial review. Moreover, in the trunk-based model, more reviews are closed later in the month, while for the merge request-based model almost all reviews are closed within seven days.

If we look only at the reviews that are closed in the first week, we show the distribution

4. DIFFERENCES IN HOW DEVELOPERS REVIEW CODE IN TRUNK-BASED AND IN MRS

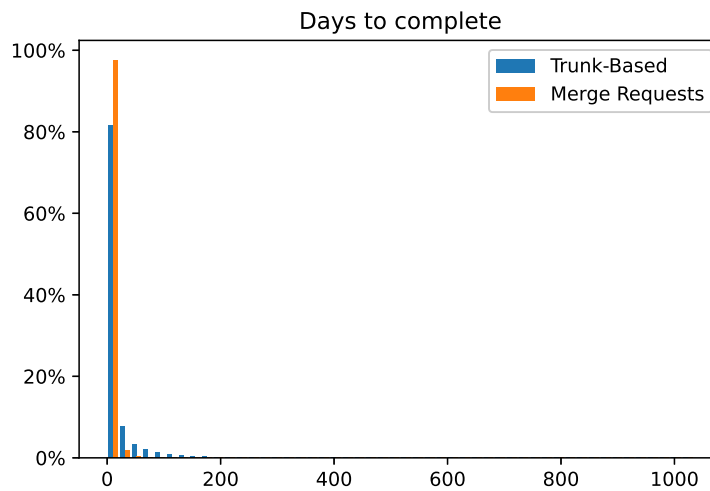


Figure 4.4: Days to complete.

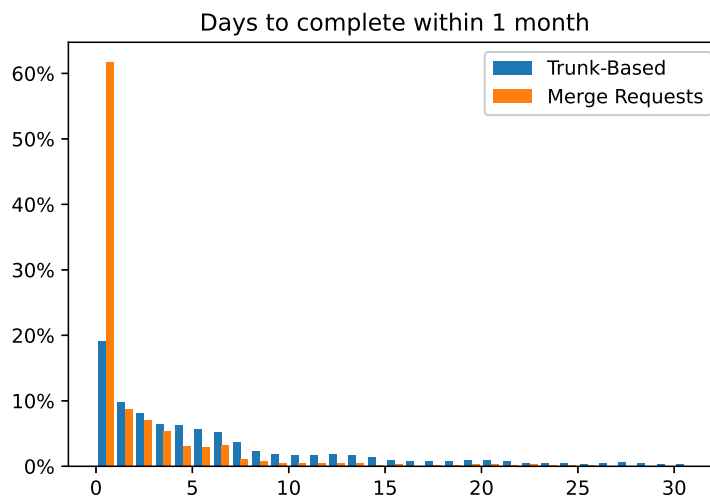


Figure 4.5: Days to complete within one month.

in Figure 4.6. Here we can still see that most reviews of the merge request-based model are closed within the first day, while that time is more spread for the trunk-based model. Interesting here is that we can see a wave pattern, which can be explained by the fact that there are fewer developers active at night.

Figure 4.7 shows the time distribution of all reviews that are closed within one day. This is nicely spread for the trunk-based model, considering the day and night. For the merge request-based model, the reviews are done sooner, with 23% of the reviews done in the first hour, where this is only 3% in the trunk-based model.

If we scope into the time distribution for all reviews that are closed within one hour, we

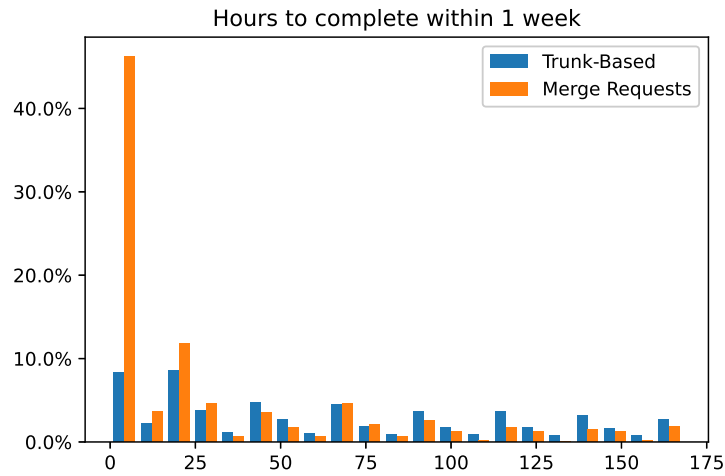


Figure 4.6: Hours to complete within one week.

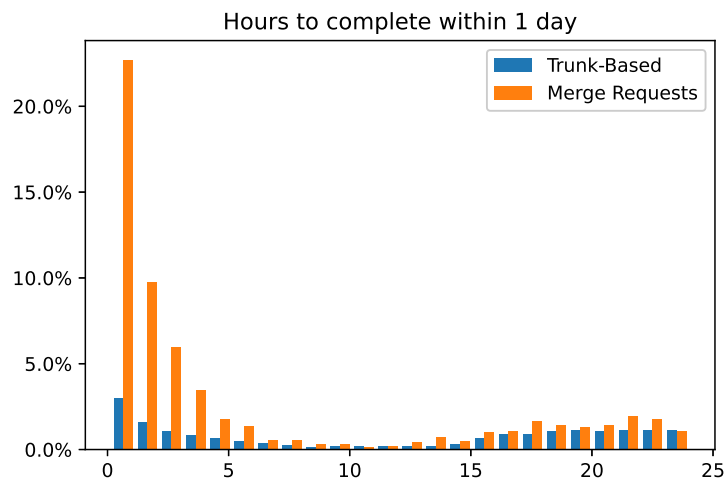


Figure 4.7: Hours to complete within one day.

can look at Figure 4.8. There are many more merge requests closed in the first hour than reviews in the trunk-based model.

We also checked for statistical significance. For this, we used the Mann-Whitney U test, also known as the Mann-Whitney-Wilcoxon or Wilcoxon rank-sum test. This method was used because we have two independent data samples. To perform this test we need to define the null hypothesis, which is H_0 . When the distributions are similar, H_0 holds. The other hypothesis H_1 holds when the distributions are not similar.

When performing this test, we got a p-value of 0.0, which means that the null hypothesis is false. This implies that the distributions are not similar. Also, we got an effect size of 0.58, which implies that there is a large difference between the two datasets. From this, we can

4. DIFFERENCES IN HOW DEVELOPERS REVIEW CODE IN TRUNK-BASED AND IN MRS

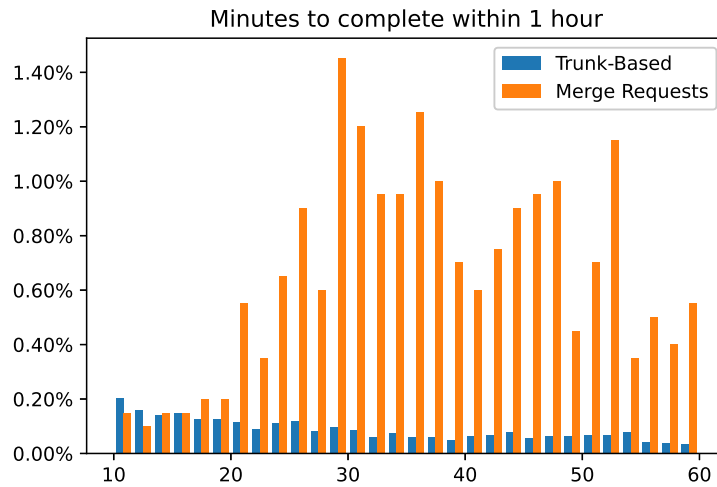


Figure 4.8: Minutes to complete within one hour.

conclude that there is a statistically significant difference in the completion times between the trunk-based development model and the merge request-based development model.

The time distributions for reviews to be completed in form of a boxplot are shown in Figure 4.9. Here we can see the reviews in the merge request-based model are completed faster and the trunk-based model has further outliers. This final finding can be explained by the fact that merge requests are not active for such a long time.

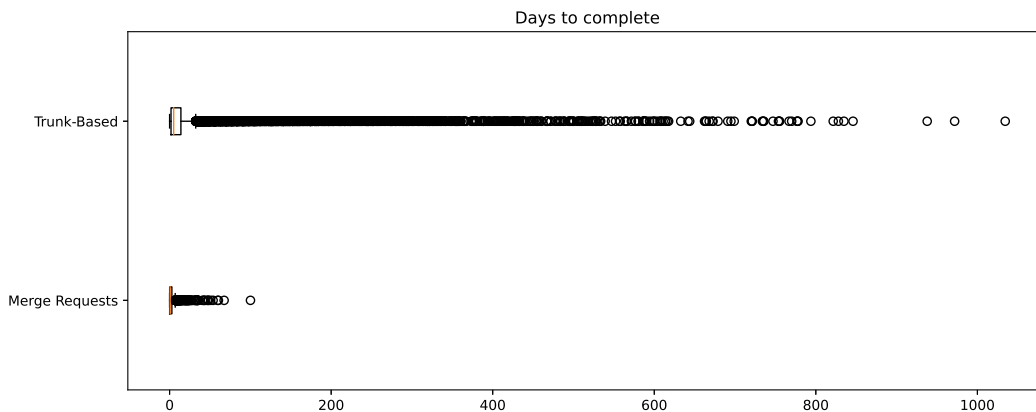


Figure 4.9: Days to complete.

A zoomed-in version of the boxplot is shown in Figure 4.10 up to 33 days. This boxplot clearly shows that the review time for merge requests is faster than the review times in the trunk-based model.

4.4. Quantitative Analysis: Number of Code comments

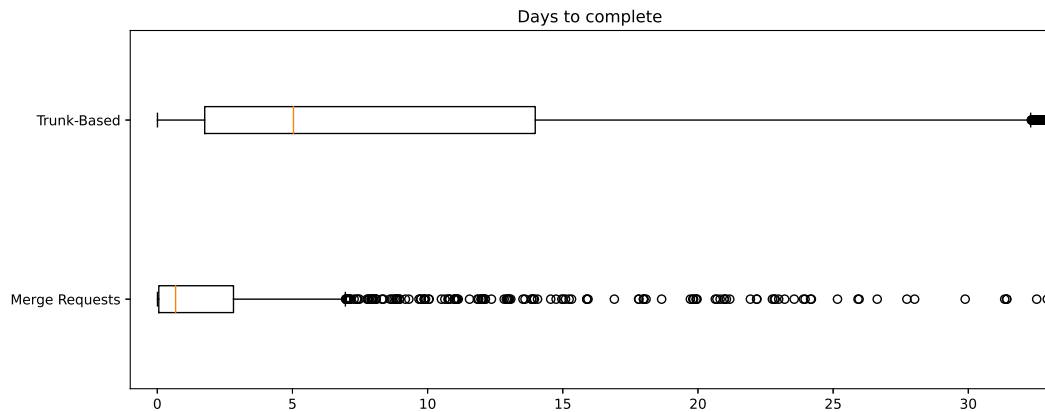


Figure 4.10: Days to complete.

Findings

F28 The time for completing reviews has significantly decreased when migrating from trunk-based development to merge requests.

F29 More than 60% of the merge requests are closed within one day, while this is not even 20% for reviews in the trunk-based model.

4.4 Quantitative Analysis: Number of Code comments

Secondly, we compared the number of comments per review in both models, to see whether there are more or fewer discussions in merge requests. Table 4.3 shows general statistics on the number of comments in reviews of the trunk-based model and the merge request-based model. Although the trunk-based model has the review with the most comments, the mean and median of the merge request-based model lie higher. This means that on average there are more comments in the merge requests. The results are visualized with a violin plot in Figure 4.11.

	Trunk-Based	Merge Requests
min	1	1
max	190	123
mean	3.53	7.55
median	2	4
standard deviation	5.63	10.81

Table 4.3: Comments per review.

4. DIFFERENCES IN HOW DEVELOPERS REVIEW CODE IN TRUNK-BASED AND IN MRS

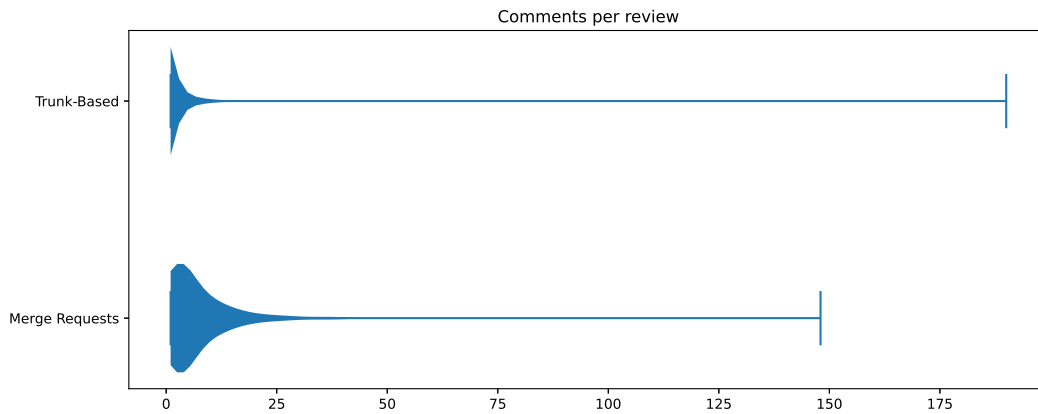


Figure 4.11: Comments per review.

Since there are also many outliers in the number of comments per review, we show a zoomed-in version of the violin plot in Figure 4.12 with a maximum of 20 comments per review. This plot shows that the proportion of merge requests with more comments per review is much higher for merge requests than for reviews in the trunk-based model.

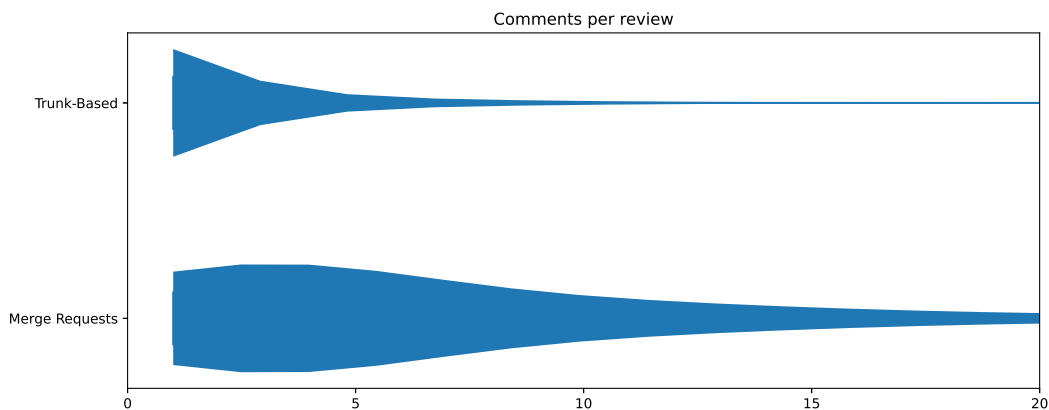


Figure 4.12: Comments per review.

Figure 4.13 shows the distribution of the comments as a percentage of the total amount of comments for the trunk-based and merge request-based development models. Since there are outliers with a lot of comments in one review in the trunk-based model, the bar chart is hard to read.

When we only look at reviews with at most 25 comments, we can see a more readable bar chart in Figure 4.14. We can see that 43% of the reviews in the trunk-based development model have only one comment, while this is 9% for the merge request-based model. When the number of comments increases, both models seem to follow the same trend.

We also checked for statistical significance. For this, we used the Mann-Whitney U

4.4. Quantitative Analysis: Number of Code comments

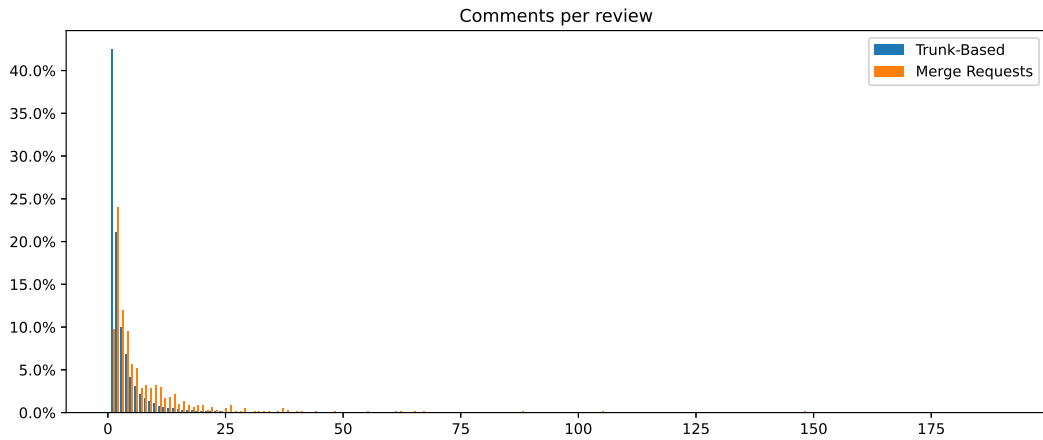


Figure 4.13: Comments per review.

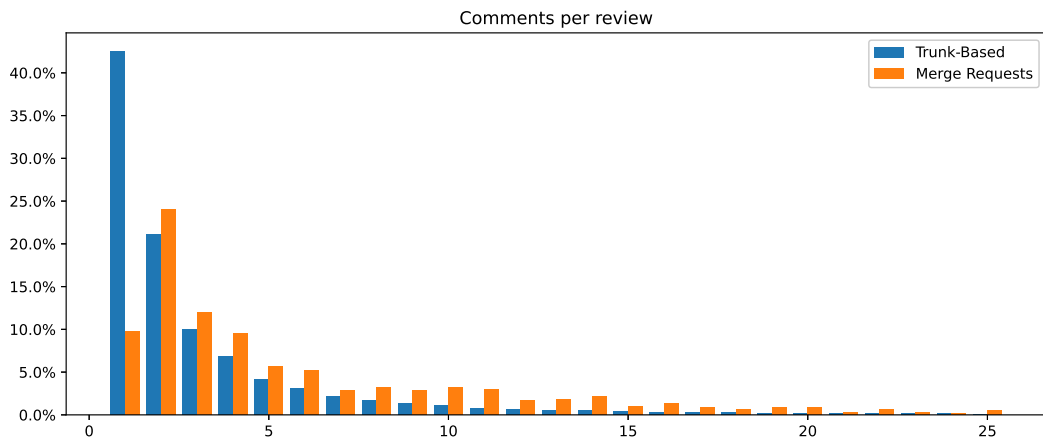


Figure 4.14: Comments per review.

test again because we have two independent data samples. To perform this test we need to define the null hypothesis, which is H_0 . When the distributions are similar, H_0 holds. The other hypothesis H_1 holds when the distributions are not similar.

When performing this test, we got a p-value of $3.35e^{-66}$, which means that the null hypothesis is false. This implies that the distributions are not similar. Also, we got an effect size of 0.43, which implies that there is a medium-size difference between the two datasets. From this, we can conclude that there is a statistically significant difference in the number of code comments between the trunk-based development model and the merge request-based development model.

The boxplots of the distributions are shown in Figure 4.15. We can see that the merge requests on average have more comments in the review.

The zoomed-in version of the boxplot is shown in Figure 4.16 up to 20 comments per

4. DIFFERENCES IN HOW DEVELOPERS REVIEW CODE IN TRUNK-BASED AND IN MRS

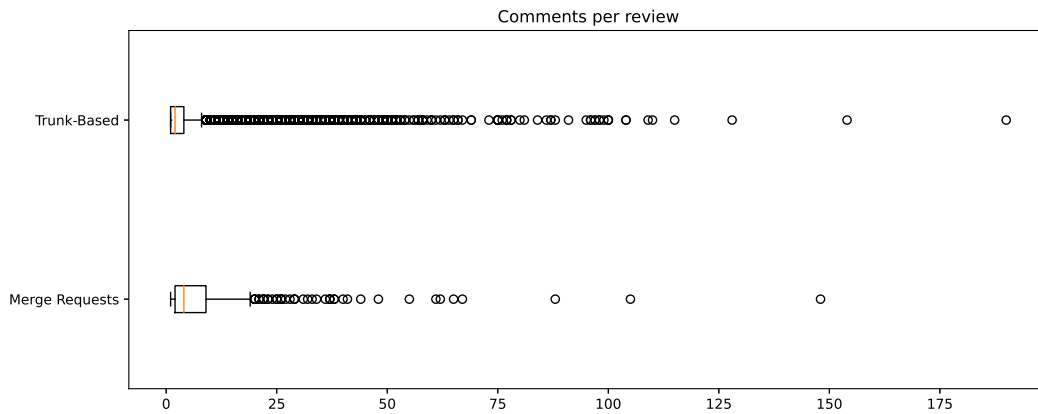


Figure 4.15: Comments per review.

review. It is even better visible that the merge requests contain more comments per review than reviews in the trunk-based development model.

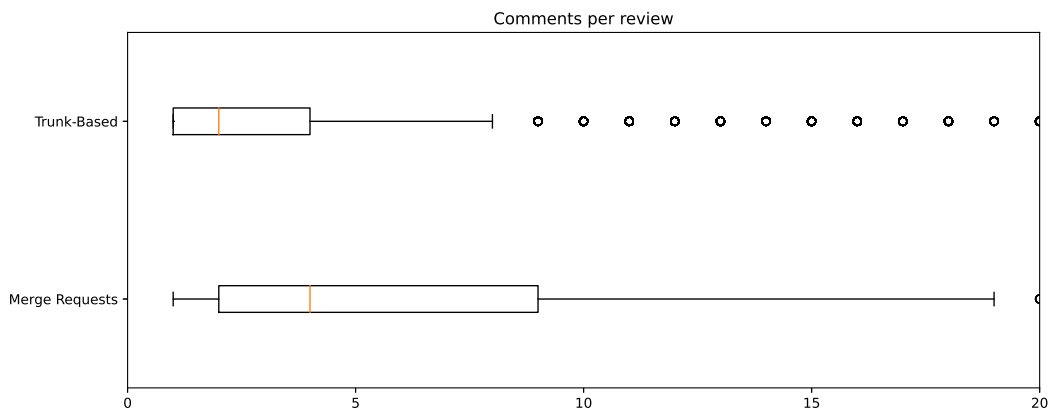


Figure 4.16: Comments per review.

Findings

F30 Significantly more code comments are placed in merge requests.

F31 Merge requests get an average of 7.55 comments per review, whereas this is 3.53 for reviews in the trunk-based model.

Chapter 5

The Developers' Perceptions on the New Merge Request Model

The goal of this chapter is to understand the perceptions of software developers on the migration from trunk-based development to merge request-based development at a large software engineering company. Also, we will compare the perceptions on the benefits and challenges with the expected benefits and challenges described in chapter 3. Unfortunately, not all developers were using the merge request-based model during the period of this study, so only the perception of a subset of the developers' group could be captured and we aim to get the best possible perception of those developers. In this chapter we will answer the third research question:

RQ3 How did the developers perceive the migration from trunk-based development to merge request-based development at Adyen?

To answer this question we define the following subquestions:

- RQ3.1** What benefits do developers see with working with the merge request-based model?
- RQ3.2** What challenges do developers face while working with the merge request-based model?
- RQ3.3** What changes did developers experience when migrating from the trunk-based model to the merge request-based model?
- RQ3.4** How do developers do their code reviews in the merge request-based model?
- RQ3.5** What changes did developers experience in code reviews when switching to the merge request-based model?

5.1 Methodology

In this section, the methodology will be explained for the data collection of the perception of developers on the migration from the trunk-based to the merge request-based development

model. We conducted eight interviews with developers from multiple teams with experience in the merge request-based model at Adyen. A summary of the methodology is shown in Figure 5.1.

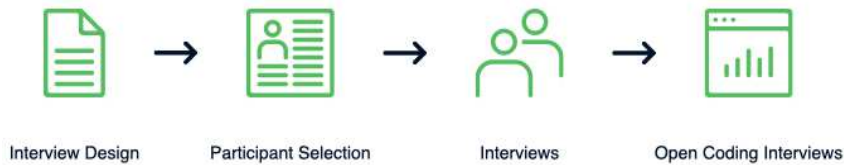


Figure 5.1: Methodology.

5.1.1 Interview Design

Similar to the interview before the migration, this was a semi-structured interview. The interview consisted of eight small parts, the introduction, participant information, general questions, migration, merge conflicts, code review, development speed, and ending questions. The full script used for this interview can be found in appendix C.

In the introduction, the purpose of the study was explained, followed by questions on information about the participant.

The next part was general questions on the perception of this migration. The interviewee was asked about the benefits and challenges of the merge request-based model compared to the trunk-based model.

More specific questions on changes from trunk-based to merge request-based were asked, regarding the migration process, merge conflicts, code review, and development speed. These topics are based on the results of the interviews on the expectations of the migration as discussed in chapter 3.

The interview was concluded with the question about which development model they prefer working with. Again, a summary of the interview was provided by the interviewer to validate the answers of the participant. Finally, the interviewee was asked whether they had additional input on the topic.

5.1.2 Participant Selection

Since Adyen was migrating during this study, not all developers had experienced the merge request-based development model at this company. Therefore, we selected the participants for the interviews based on the activity of developers in merge requests. For this, we used the GitLab API¹. We collected information on the number of times a developer created merge requests, the number of times a developer was assigned as a reviewer to merge requests, the number of times a developer took action in merge requests, and the number of comments a developer placed in merge requests. We invited ten developers through a personal message

¹docs.gitlab.com/ee/api/

on Mattermost² from which eight responded and were available, this gives a response rate of 80%.

The full list of participants interviewed after the migration is shown in Table 5.1.

Participant Information on Merge Requests							
ID	Software Engineering Experience	Time at Adyen	Team	Author	Reviewer	Actions	Comments
P12	6m	6m	J	6	27	67	33
P13	15y	8m	J	22	14	229	52
P14	1y	1y	K	23	15	139	43
P15	7y	3y 6m	K	5	1	49	21
P16	9y	2y	A	68	74	518	99
P17	6y	1y	J	11	20	121	57
P18	1y	9m	A	136	59	353	72
P19	5y	11m	K	36	21	255	70

Table 5.1: Interview participants after migration. Information is based on the date of each interview. The final four columns show the data of each participant, which are the number of times the participant was the author of a merge request, the number of times the participant was the reviewer of a merge request, the number of actions the participant took in merge requests and the number of comments the participant placed in merge requests.

5.1.3 Data Analysis

The data analysis is similar to the first round of interviews. All interviews were conducted online over Zoom³. Six interviews were conducted in English and transcribed with Otter.ai⁴ and two interviews were conducted in Dutch and automatically transcribed with Trint⁵. These transcriptions were corrected and analyzed in ATLAS.ti⁶. Open coding was applied to the relative parts of the interviews.

5.2 Perceptions on Code Reviews

In this section, we will present the results regarding the code reviews of the interviews conducted on developers who have experienced the merge request-based model at Adyen.

²mattermost.com

³zoom.us

⁴otter.ai

⁵trint.com

⁶atlas.ti

5.2.1 Perceived Advantages

Code reviews have changed a lot during the migration in many ways, for example, they only review team members in the new model whereas they had to review developers from other teams before (P15). This participant also mentions that they feel more comfortable reviewing their team members as they know them personally and see this as a benefit of the new model. In their words: *“With Upsource, I was assigned one other person like outside the team, and that person will be like maybe rude or does not know anything about to project and give like a wrong comment. But with this merge request, I feel like home, you know, like, whoever makes a comment is just for good.”*

The participant shared how they tackle code reviews in the merge request-based model. One developer (P16) mentions that it is best to wait with handling the review comments until the reviewer is completely done with the review. This will also save some time as most of the time only one or two iterations are necessary. An advantage of doing code reviews in GitLab is that it is possible to place comments on multiple lines of code instead of one line in Upsource (P12). This makes it clearer to the author what piece of code the comment is about. Nevertheless, some developers (P13, P14, and P15) say they do not approach code reviews differently in the new model.

5.2.2 Knowledge Sharing

A great benefit of the merge request-based model is that more knowledge is shared among the developers because more discussions are happening in the new model (P12, P13) and developers have a better understanding of what others are doing since merge requests are mandatory (P17). In their words: *“I think it also helps in, you know, building your knowledge about the codebase, because you are exposed to more stuff, and you can start learning more what other people are doing.”* One developer (P12) mentions within their team they assign everyone as a reviewer for each merge request to keep everyone up to date. Moreover, developers perceive that the merge requests have resulted in more detailed feedback in the code reviews (P16, P17) and it is easier to get feedback earlier in the new model (P12, P17). One developer (P19) mentions that code reviews are there to learn and teach. In their words: *“I see the code reviews as an opportunity to learn or to teach.”*

5.2.3 Code Quality

Arguably the most important benefit of the merge request-based model perceived by the interviewees (P12, P13, P14, P15, P16, P17, P18, P19) is the increase of code quality and they expect the quality to continue to increase when using the merge request-based development model. Developers mention different reasons for the increase of quality, such as merge requests often bring good discussions (P12, P19), code reviews are much more useful in the merge request as the author has to act upon the comments for their code to be pushed to master (P12, P16, P19), and errors have been caught during merge request reviews that would otherwise have been pushed to master when using the trunk-based model (P13, P16, and P18). P16 said: *“Especially clarity in smaller improvements, renaming a function*

or variables, or even redoing some of the logic. And this kind of thing, in my point of view, improves the quality of the code.”

Moreover, developers feel more stimulated to do a proper code review because they feel more responsible as a reviewer and they know that the author has to act upon the comments they make (P12, P13, P14, P18, P19) and feel more stimulated to look better at code review comments they received (P12, P19) because if they do not improve their code properly, the reviewer will not approve the merge request.

5.2.4 Review Size

The interviewees say that they experience larger reviews. Developers mention that the merge requests have become bigger in the new model because merge requests contain multiple commits instead of one review for each commit (P12, P17, P18). However, one developer (P18) mentions that the Upsource reviews could grow larger if developers reopened old reviews by adding new commits with the same ticket prefix. We have also seen an increase in comments for the merge requests in the quantitative analysis done in section 4.4.

5.2.5 Web Interface for Code Reviews

Developers have mixed opinions on the interface in which the merge request are being done. One developer (P16) mentions preferring the Upsource interface for code reviews because it dedicates more space to the code review. On the other hand, another developer (P12) mentions that the GitLab interface is very useful for code reviews as the web page only shows the changes that you have not seen yet and marks the others as seen. However, the same participant mentions to be missing the integration with IntelliJ for code reviews and would like this plugin to be available for Adyen developers.

Findings

- F32** Developers perceive that more knowledge is shared among the developers.
- F33** Developers perceive that the review quality and code quality increase.
- F34** Developers perceive that better and more discussions occur in the merge request-based model.

5.3 Development Speed

A big concern before the migration was that the development speed might decrease significantly and the interviewees (P12, P13, P14, P15, P16, P18, P19) mention that indeed the development speed has slowed down, except for one interviewee (P17), who mentions that the development speed has not changed when switching models. Nevertheless, one developer (P18) says that the time between creating code and it being in master increases, but the

overall development speed does not necessarily decrease since you can continue on another task when waiting for a review.

5.3.1 Slower Speed

Developers give multiple reasons why the speed has decreased, such as merge conflicts (P15), the continuous integration pipeline takes too long as the development time depends on how many times one has to run the CI pipeline (P12), and IntelliJ is very slow in switching branches because indexing everything takes a lot of time (P15).

Developers suggest solutions to minimize the speed loss, such as reviewing their own changes to make sure that they are delivered faster (P14) and shorter release cycles (P13), which are possible in the merge request-based model as everything in the master has been reviewed and tested by the reviewer. Overall, developers (P14, P15, P16, P19) feel that there is a trade-off between speed and code quality and that it is worth having a slower development speed when the code quality increases. According to P19: *“I think the advantage of the code being a lot better is a lot more important than the speed because usually the speed you get, you just introduce bugs because you are doing everything in one branch.”*

5.3.2 Error Detection and Feedback

Another benefit of the merge request approach is that errors are noticed earlier and can be fixed sooner (P15 and P18) and this is faster when you compare it with the fixes that need to be made in the trunk-based model (P16).

This also holds for receiving feedback since the development speed is faster because developers get feedback earlier in the new model (P16, P17). P16 said: *“Even if you are not done, you might create the merge request and ask for a review. You are aware that it is not done, but at least you get feedback on whatever is done. You keep it as a draft merge request, and you can get a review on it. So that might even speed up development.”* The same developer mentions that there is less context switching in the merge request-based model which results in faster development and the time between submitting and receiving feedback is not that high.

5.3.3 Team

The interviewees also mention that the success of the merge request-based model is dependent on your team and other developers (P12, P13, P15). If your team reviews fast you can have your changes into master very rapidly. More specifically one developer (P13) mentions that the development speed depends on the number of team members who can review your merge request. If the number is too low, those people might be too busy reviewing merge requests, which slows down development speed. Developers mention they already had to chase people for code reviews which result in communication overhead (P12, P13, P18), so the trunk-based approach was faster because they did not have to wait for others to get approval (P13).

Findings

- F35** The overall development speed has decreased.
- F36** Errors are caught earlier and fixed sooner.
- F37** When considering fixing the errors and acting on the feedback, the merge request-based model is faster.
- F38** The development speed is mostly dependent on the team members who will review the code.

5.4 Merge Conflicts

Before the migration, more merge conflicts were expected, however, some developers have not experienced conflicts yet (P14, P16, P17) or experienced fewer conflicts (P13, P19). On the other hand, one developer (P15) says that there are now more merge conflicts because they are waiting on the approval of the reviewer for the code to be merged which can also cause a merge conflict loop and another developer (P17) mentions that the possibility of conflicts is still very high because Adyen is still a mono repository.

5.4.1 Reasons for Merge Conflicts

The interviewees provided multiple reasons for how the merge conflicts occur, such as working in the same files a lot (P12, P14, P15, P17, P19), especially for front-enders (P14). On the contrary, one developer (P18) says that conflicts are not occurring that often because they are not working on the same files within their team.

Moreover, merge conflicts might still occur due to the long waiting times on the pipeline to finish (P13, P18) and some developers mention that the only merge conflicts they had so far were caused by themselves starting on the next feature while the first was still in review (P16, P18). Two developers (P12, P19) mention that the merge conflicts in the merge request-based model look similar to those in the trunk-based model.

5.4.2 Fixing Merge Conflicts

There are some benefits in the merge request-based model concerning the merge conflicts, such as that the merge requests are easy to solve (P15, P17, P19). Moreover, developers (P12, P13) mention that merge conflicts are earlier noticed than in the trunk-based model because merge requests can be opened at the first commit before the merge, while in the trunk-based model developers tend to wait with their commit until the feature is finished. P12 said: *“I got the idea that now you notice merge conflicts sooner, maybe because it is more clear because everyone is working in merge requests. In the merge request branches you can very quickly see, this is going to cause a merge conflict, whereas in the trunk-based approach you would only notice it when you are going to submit.”*

However, there are still some challenges according to the interviewees because fixing a merge conflict can be confusing for the reviewer since it is sometimes not clear what has been changed (P12). Another developer (P18) emphasizes that merge conflicts must be solved carefully because it happened that someone kept the wrong version when solving the merge conflict.

Findings

F39 The number of merge conflicts did not raise as expected.

F40 Merge conflicts are earlier noticed and can therefore be solved sooner.

F41 Solving merge conflicts can be confusing and should be looked at carefully.

5.5 GitLab

One big change during the migration was the switch of the online repository from Gitea to GitLab and the switch from code reviews in Upsource to GitLab. For that reason, the interviewees had some remarks on the online repositories, such as that GitLab is preferred over Gitea since GitLab provides more features and their support is much better (P16), and the standard GitLab features are preferred over the custom Adyen-specific tools (P13).

Merge requests contribute to a better-structured repository and make it easier to see the full picture of a change in the code reviews (P12, P17). However, maintaining a clean Git history might propose a challenge and it requires strict agreements with the team (P12).

5.5.1 Lacking Features

The way GitLab is used at Adyen is not incorporating the complete GitLab flow (P16, P17, P19) as many features are not used yet, but this is planned for the future. Some buttons in the GitLab interface are not working (P12, P16), which is confusing. For example, one developer (P16) tried to cancel a build because the feature was not ready yet, but this was impossible because the cancel button was not working yet. Moreover, it is considered annoying completed merge requests are shown in GitLab as *'closed'* instead of *'merged'* (P16, P19) and the integration of GitLab with the tool for maintaining tickets is missing (P19), which can be useful to switch from the feature description directly to the code.

5.5.2 Custom Scripts

To simplify the merge request flow, a special git command was created, named *git mr submit*, which pushes all your changes to a separate feature branch, automatically creates a merge request, and assigns reviewers. This custom script is not straightforward (P13, P16) as it does help with automation, but rather when things go wrong you are lost (P16). One

developer (P18) mentions they manually open merge requests for big changes instead of using this script.

The standard GitLab features are preferred over the custom Adyen-specific tools (P13, P16) and the team responsible for those scripts underrated the developers' knowledge of git (P16). They received a lot of feedback that developers prefer to stick to normal Git commands instead of the custom scripts. Moreover, Adyen is planning on using the GitLab CI to check for merges and eventually for everything.

5.5.3 Overall Challenges

Moreover, there are some other challenges faced by the developers. One developer (P13) mentions that the name requirement on branches is annoying. Feature branches have to be named with a specific format that includes the name of the author and the ticket number from the issue tracking system. This participant also mentions that it is not easy to switch branches anytime. Also, when the pipeline is not passed, you need to manually retrigger the pipeline by placing a comment, which can be annoying (P17).

Findings

- F42** GitLab is the preferred tool to use because of its many features and the fact that it is widely used in the industry.
- F43** At Adyen, a lot of features from GitLab are lacking and there is still room for improvement.
- F44** Developers prefer the industry-standard features instead of custom scripts.

5.6 Merge Requests

To get a better understanding of the merge requests, we asked the participants about their experience of the merge requests so far. Some developers (P12, P18) open a merge request as soon as a minimum requirement is met, but not when the feature is completely finished. This way the reviewer can already look at the commits that have been made and possible merge conflicts will already show. Another developer (P14) mentions that in their team they keep their merge request small to not make the code reviews too hard.

5.6.1 Amending Commits vs New Commits

One feature of the merge requests is that the author can amend commits or add new commits to the merge request. Amending commits can be confusing for the reviewer (P13, P16) as this rewriting of the history can get messy in the overall commit history. On the other hand, creating new commits to fix an issue is better understandable for the reviewer as they might have already reviewed some commits (P13). The same developer misses the functionality to

squash commits when the merge request is approved because that would keep the git history cleaner.

5.6.2 Communication

One major benefit from the merge requests is that there are more discussions taking place as there is better communication in the new model (P14, P15) and the threshold for discussions and making changes has decreased since in the new model the code is not in master when it is being reviewed and therefore is more open for changes (P12, P15). Moreover, merge requests help to build knowledge about the codebase as developers teach and learn from each other (P17, P19) and the author and reviewer are aligned on the version of the code that goes to master (P16, P19), which is beneficial for the codebase.

Findings

- F45** Merge requests are opened as soon as there is code to share with the reviewer without it being completely ready.
- F46** Amending commits can be confusing for the reviewer while adding new commits results in a messy commit history.
- F47** Merge requests provide better discussions and help to build knowledge.

5.7 Migration

In this section, we will discuss the perceptions of developers on the whole migration process at Adyen.

5.7.1 Overall Experience

Developers experienced the migration as easy and not painful at all (P17, P18) because they were already familiar with the merge request-based model (P12, P17, P18, P19). Moreover, some interviewees mention that everyone in their team had previous merge request experience (P12, P17), which had helped to migrate. P17 said: *“Overall, it is what I said, I like the approach. And I think we are doing a very nice migration. I mean, it was not painful. And I think we managed to run both of them in parallel, which is also super nice. So yeah, so far, I am happy with the merge request. I am looking forward to the next step.”*

Nevertheless, one developer (P15) mentions having asked their team for help on how to use the merge request-based approach since this person had no previous knowledge of the merge request-based model. This participant also mentioned that it took some time to get used to the new flow. Even when following the instructions they faced problems pushing code to their branch. Eventually, they understand how to properly use the merge request-based approach.

The instructions for creating merge requests on the company's internal website have helped with getting up to speed with the new model (P12). This website provides documentation on the merge request flow and explains the custom `git mr submit` script. One developer (P18) mentions that the migration was made easy due to the automated scripts that were provided by Adyen, while others (P12, P17) mention that they needed to get used to the custom automation scripts.

5.7.2 Benefits of the Merge Request-Based Model

There are multiple benefits of the migration that the interviewees mentioned, such as reviewers are not doing the minimal required effort anymore when doing code reviews (P12, P16, P18, P19). They said that reviewers used to approve reviews too quickly without sufficiently looking at the code changes because the code did not break the integration tests and was working and running on the master. Also, developers' happiness has increased when using the merge request-based model as perceived by one developer (P16) and another developer (P12) is happy that their user branch does not expire anymore.

5.7.3 Challenges of the Migration

However, the migration also proposed some challenges because the large size of the company and given that Adyen is a mono repository makes it very difficult to migrate all developing teams to the merge request-based model (P16). According to the same participant, it was especially a struggle for the people working on the migration.

Moreover, small changes or urgent fixes cannot be pushed directly when using the merge request-based model. Some participants see this as a downside of the new model, as urgent fixes get blocked from merging to master because they need to be approved by a reviewer first (P15, P17). Moreover, as an author, it can be annoying to get comments on their code for small changes with low impact (P16) and one developer (P14) even mentions to have used the trunk-based approach for small changes to avoid having to wait for a review.

5.7.4 Trunk-Based or Merge Requests

To conclude the interview we asked the participants which development model they preferred and why. Most developers (P12, P13, P15, P16, P17, P18, P19) prefer the merge request-based model over the trunk-based model as it is always better to review code before merging (P13) and in addition to the code review, you can also review the commit title and text (P16, P19). Adyen is using prefixes and ticket tags in their titles, which should be properly used. P16 said: *“Not considering GitLab, the application itself because I think GitLab brings lots of improvements over Gitea as well. But I do like merge requests and one thing that merge requests enable that trunk-based development will never enable is reviewing even the commits, the titles, and the texts. That is not something that we do currently. But merge requests enable this.”* Moreover, a benefit of the merge request-based model is that it is the current industry standard of software development (P15, P17).

Others argue that the best development model depends on the situation because they prefer what they are familiar with (P12, P13), which is the merge request-based model for

5. THE DEVELOPERS' PERCEPTIONS ON THE NEW MERGE REQUEST MODEL

both of them. One developer (P18) mentions that the trunk-based approach is preferable in some situations, such as for start-ups and small development teams.

As also suggested in the interviews before the migration, a hybrid model might be a good solution as well. Only one developer (P14) mentions preferring the hybrid model where most contributions are merged after a code review, but where there is a possibility to merge directly for small non-functional changes. In their words: *“I like the old way, it has a flexible way. Also, I think the quality is also important. So I think in the future, like for the small changes, no functional changes, I would prefer the old way, but for like, really like big changes, the functional changes, I prefer merge requests.”*

Findings

- F48** The migration process went smooth for developers with the help of documentation and team members.
- F49** Developers are overall happier with the merge request-based model.
- F50** The large size of the company makes it difficult for the responsible developers to migrate all development teams.
- F51** Not being able to push small changes or urgent fixes to the master branch is seen as a challenge.
- F52** The merge request-based model is preferred by the majority of developers in the situation of Adyen.

Chapter 6

Discussion

In this chapter, we will discuss the implications of this research and provide recommendations to Adyen and other companies in the software engineering industry. Also, we will discuss the threats of validity that this research might include.

6.1 Implications

In this section we will summarize all the findings in this thesis, which are stated in chapters 3, 4, and 5. The (F x) in parenthesis point to the findings presented in those chapters.

6.1.1 Development Speed

First of all, before the migration, code reviews in the trunk-based model were perceived as slow (F2). However, quantitative analysis showed that the time for completing reviews has significantly decreased when migrating from trunk-based development to merge requests (F28). More than 60% of the merge requests are closed within one day, while this is not even 20% for reviews in the trunk-based model (F29).

In the trunk-based model, the release cycle was not considered too short and was also not expected to be too short when using merge requests (F8). Developers did not expect that time will get wasted by waiting on code reviews (F7). After migrating, developers perceive that the overall development speed has decreased (F35) as errors are caught earlier and fixed sooner (F36). Moreover, developers feel that when considering fixing the errors and acting on the feedback, the merge request-based model is faster (F37). Nevertheless, they mention that development speed is mostly dependent on the team members who will review the code (F38).

6.1.2 Code Quality

In the trunk-based model, developers felt that they had to convince the author to improve the code (F1) while bugs could have been prevented if the responsible code was reviewed before merging (F13). Also, tests were considered more reliable than code reviews (F17), and code reviews were considered very important for beginner developers (F3). Developers

do not want senior developers to have the ability to merge directly without a review in the merge request-based model (F22). Moreover, developers felt scared to break the code when pushing directly to master (F15) and mention that the cost and effort of fixing a bug are higher than preventing it (F14). However, developers did not want to postpone the release until everything was reviewed (F9).

Before the migration, developers expected that mandatory code reviews would help beginner developers to learn best practices (F4), increase review quality and therefore code quality (F5), and fewer bugs would occur (F16). Moreover, the commit history was expected to improve (F18) and saving code remotely was expected to be a benefit of the merge request-based model (F20). However, review quality was expected to decrease if the merge request are too large (F19).

After the migration, developers perceive that more knowledge is shared among the developers (F32), the review quality and code quality increased (F33), and better and more discussions occur in the merge request-based model (F34, F47). In the quantitative analysis, we found that indeed significantly more discussions occur in merge requests (F30) as they get an average of 7.55 comments per review, whereas this is 3.53 for reviews in the trunk-based model (F31). Quantitative analysis of the code reviews show that most comments in reviews are about code improvement and understanding (F25), social communication has decreased during the migration (F26), and more comments to transfer knowledge are placed in merge requests (F27).

6.1.3 Merge Conflicts

In the trunk-based model, merge conflicts did not happen very often (F10) and were considered easy to fix (F12). Developers expected more merge conflicts to occur in the merge request-based development model (F11), however, the number of merge conflicts did not raise as expected (F39). On the contrary, merge conflicts are earlier noticed and can therefore be solved sooner in merge requests (F40) as merge requests are opened as soon as there is code to share with the reviewer without it being completely ready (F45). Nevertheless, solving merge conflicts can be confusing and should be looked at carefully (F41).

6.1.4 Development Models

Before migrating, the trunk-based development model was not considered to always be the best option for startups and fast-developing codebases (F23), and the merge request-based development model was considered to be preferable for mature codebases and high-impact projects (F24). However, having experienced the merge requests, developers are overall happier with the merge request-based model (F49) and the merge request-based model is preferred by the majority of developers in the situation of Adyen (F52). The only challenge is that developers are not able to push small changes or urgent fixes to the master branch (F51).

In addition, the migration process at Adyen was perceived as a smooth process for developers with the help of documentation and team members (F48). However, the large size

of the company makes it difficult for the responsible developers to migrate all development teams (F50).

6.2 Recommendations

Developers also had some complaints or remarks on the migration, which they expressed during the interviews and survey.

First of all, the custom scripts that were created to simplify the merge request flow was sometimes confusing since it was not clear what they were doing. Developers would like to use the industry standards (F42) as most of the developers are already familiar with merge requests. Moreover, a lot of features from GitLab are lacking and there is still room for improvement (F43). Also, recent joiners of Adyen said that the onboarding can be very complicated with all those extra tools. So, we recommend maintaining the regular git commands as much as possible and keeping up with the standards in the industry, and making the process of onboarding of new developers smoother (F44). In addition, Amending commits can be confusing for the reviewer while adding new commits results in a messy commit history (F46), so we recommend creating new commits and squashing them during the merge if necessary.

Also, the pipeline that runs all the checks takes a very long time. Currently, the pipeline runs in Jenkins, which is an external continuous integration tool. Developers would like to have a lightweight CI version that can quickly check for each commit if it is likely to fail tests. And only when merging to master, the complete CI should be performed. I would recommend this flow. Moreover, to maintain the fast development speed, we recommend having automatic reminders sent to the reviewers for merge requests (F6) and having the system give warnings when feature branches are too long-lived (F21).

6.3 How to expand to the community

From the results shown in the previous chapters, a few observations became clear. One big observation from the interviews and the survey is that the merge request-based approach is preferred over the trunk-based approach for the codebase of Adyen. The main reason for this is the increasing code quality, as more discussions take place in code reviews. Also, the guarantee of code always being reviewed before being merged to master is considered a major benefit of the merge request-based model.

Although developers mentioned before the migration that they prefer the fast development speed of the trunk-based model, from the interviews after the migration this was not considered a large problem as reviews were done very quickly. This is also shown in the quantitative analysis of the code reviews, where we have shown that code reviews are significantly completed faster. Therefore, migration seems to have been a good choice for the development at Adyen.

To the software engineering community that is considering developing in either the trunk-based or merge request-based approach, I would suggest looking at the following aspects of the project:

1. **The size of the development team:** For smaller teams it might make more sense to develop with the trunk-based approach as it might take a lot of time to wait for a reviewer to be available.
2. **The impact when live breaks:** When the costs of breaking live are high, it might be better to use the merge request-based approach as this introduces an extra security check.
3. **The maturity of the project:** For startups it can be more beneficial to develop trunk-based as you are building an application that has not been released or you need to launch and iterate very rapidly. In contrary to mature projects, where the development speed decreases and code quality becomes relatively more important than fast development.

6.4 Threats to Validity

In this section, we will discuss internal and external threats to validity.

6.4.1 Internal validity

We believe that this research proposes several internal threats to validity. We can categorize two types of internal threats to validity: credibility and confirmability.

Credibility

First of all, the response rate of the interview participants for the first round was only 44%. This means that only a selection of developers had input in the interviews. We found that especially people who recently joined the company had time for interviews while senior developers were busier. This could propose a bias to the results. The same holds for the responses to the survey. With a response rate of 8.3%, only a portion of the opinions is captured.

The interviews were only conducted by one interviewer and analyzed by the same person. Although approached objectively, this could still propose a researcher bias. During the qualitative analysis of code reviews, codes were assigned to comments on the opinion of one researcher. Also, the results were compared to the findings of Bacchelli and Bird [4], which were known before the codes were assigned. This could propose a regression bias towards the distribution of Bacchelli and Bird.

Also, the quantitative analysis of the code reviews is not done on all code reviews, which might not perfectly reflect the development models. For example, code reviews that were completed within ten minutes were discarded, since this did not seem a realistic time frame to properly complete a code review. But a proper limit to determine whether code has been reviewed or whether it was forced into master is discussable.

One of the largest threats to validity is the interview participant selection after the migration. Since not the whole company was migrated to the merge request-based development

model, only developers from a few teams were qualified to interview. Although the response rate was 80%, the interviewees do not give a complete representation of the entire company.

Confirmability

This research also has one high threat to validity concerning confirmability. Since this research was started when the whole company was using the trunk-based development model and ended when the company was partly migrated to the merge request-based development model, it is almost impossible to replicate the study without bias at this company. Especially the developers that have experienced the migration will have a biased opinion on the new model and are therefore unsuitable to interview on their expectations of the merge request-based model.

6.4.2 External validity

There are also two types of external threats to validity: generalizability across populations and generalizability across situations.

Population

For this study, only a small subgroup of the developers was interviewed. Although selected randomly, only 19 of more than 500 developers participated in the interviews, which proposes a threat to the external validity as it is hard to say whether this subgroup is generalizable to the whole developers' group.

The same holds for the subgroup of developers that responded to the survey, but here the participants were not selected randomly as everyone was invited. However, the response rate was also much lower, which could mean that the subgroup that filled in the survey is not a good average of the whole development group.

Situation

It is possible to replicate this study at other companies when they are going to switch from the trunk-based development model to the merge request-based development model, however, there are some threats to external validity when trying to replicate this study. The migration is dependent on a lot of variables that might be different at other companies.

First of all, the environment in which developers are doing their code review is tailored to Adyen specifically. Before the merge, code was pushed to the repository on Gitea, and code reviews were performed in Upsource. After the merge code was pushed to GitLab and reviews were done in GitLab as well. This could propose bias towards one or the other model because of preference of the tool instead of preference of the development model. Also, the tools used can be different for other companies.

Moreover, Adyen is a unique company in the sense that it is a relatively young company considering the large size and the high number of employees, especially important for this research is the high number of developers. There are not many companies in the software engineering field that match these criteria. Either a company is more mature and more

6. DISCUSSION

likely already using the merge request-based model for a very long time. Or the company is young and counts only a few developers, in which case it might not make sense to use the merge request-based approach as there is very low impact or it would slow down the development speed significantly. Either way, the situation differs from Adyen and these threats to generalizability should be considered.

Chapter 7

Conclusion

In this chapter we will repeat the research questions posed in the introductions and the subquestions posed in chapter 3, 4, and 5. We will answer these questions by summarizing our results. Finally, we will provide some food for thought for any future work.

RQ1 What are the expectation of migrating from a trunk-based development model to a merge request-based development model at Adyen?

RQ1.1 What benefits do developers see with working with the trunk-based model?

RQ1.2 What challenges do developers face while working with the trunk-based model?

RQ1.3 What do developers expect to change when migrating from the trunk-based model to the merge request-based model?

RQ1.4 How do developers do their code reviews in the trunk-based model?

RQ1.5 What do developers expect to change in code reviews when switching to the merge request-based model?

To answer the main research question, the expectations of migrating from trunk-based development to merge request-based development varied a lot. Whereas some developers expected an increase in code quality and decrease of errors, others expected a lot of merge conflicts and slower development speed. This answers **RQ1.1**, as the main benefit of the trunk-based model is the development speed because code is instantly in the master when pushed. Answering **RQ1.2**, the main challenge is the unreviewed code in the master which proposes danger to the codebase. To answer **RQ1.3**, we see that developers expect many different processes to change, such as a decrease of development speed, increase of merge conflicts, and increase of code quality. Answering **RQ1.4**, the way developers did their code reviews in the trunk-based model is by reviewing the code after the merge. While most reviewers do the code reviews properly, it occurs that some developers approve code reviews when the code has been in the master for some time and did not break anything. To answer the final subquestion **RQ1.5**, when switching to the merge request-based model, developers expect code reviews to play a more important role because they are mandatory

7. CONCLUSION

for your code to be merged. Reviewers will be more responsible to ensure the good quality of the code.

RQ2 What has changed in code reviews after migration from trunk-based to merge request-based at Adyen?

RQ2.1 What are the motivations for code review at Adyen?

RQ2.2 What changes in the time for a review to be closed when migration from the trunk-based development model to the merge request-based development model?

RQ2.3 What changes in the number of comments in the reviews?

For the second main research question, we used the data from Upsource and GitLab to visualize the differences between the trunk-based model and the merge request-based model. We have shown that there are differences in motivations for code reviews, the time for reviews to be completed, and the number of comments in code reviews. To answer **RQ2.1**, the main motivations for code reviews for both models are code improvement and understanding. The motivations for code reviews have shifted from social communication to review tool and knowledge transfer. Compared to Bacchelli and Bird [4], the main difference is that there are much more code reviews on knowledge transfer for both models at Adyen. Answering **RQ2.2**, the time for a review to be closed has significantly decreased when migrating from trunk-based to merge request-based. To answer **RQ2.3**, the amount of code comments per review has on average increased after the migration. So to summarize, while the code reviews contain more comments, they are closed faster in the merge request-based model compared to the trunk-based model.

RQ3 How did the developers perceive the migration from trunk-based development to merge request-based development at Adyen?

RQ3.1 What benefits do developers see with working with the merge request-based model?

RQ3.2 What challenges do developers face while working with the merge request-based model?

RQ3.3 What changes did developers experience when migrating from the trunk-based model to the merge request-based model?

RQ3.4 How do developers do their code reviews in the merge request-based model?

RQ3.5 What changes did developers experience in code reviews when switching to the merge request-based model?

The third main research question on how developers perceived the migration will be answered by information retrieved through interviews. To answer **RQ3.1**, the main benefit of the merge request-based model is the increase of code quality. There are more and better discussions in merge requests and the commit history in the merge request-based model is cleaner. Answering **RQ3.2**, the only challenge in the new

model is that the development slightly decreases when considering the time between the commit and code being in master. Although considering the time it takes to fix errors that are introduced in the trunk-based model that could have been prevented using a merge request, some developers argue that the merge request-based model is faster. Nevertheless, code quality versus speed is the most important trade-off that can be made between the two models. To answer **RQ3.3**, the main change that developers experienced from the migration is that code reviews play a more important role in the development flow. Developers spent more time doing code reviews and acting upon comments they receive. Also, developers have a better understanding of what team members are doing and can therefore also provide better feedback. Regarding **RQ3.4**, developers approach the code reviews in a similar way as in the trunk-based model, only they block more time to do their code reviews. To answer the final subquestion **RQ3.5**, instead of reviewing random commits done by anyone in the company, reviewers get only assigned to merge requests within their team. Most developers prefer this way of reviewing.

The goal of this thesis was to observe the migration from trunk-based development to merge request-based development. We gathered a lot of data, mostly through the opinions of developers. The reason for the migration reduce the number of errors on the master branch and increase code quality. Since not the whole company migrated to the new model, it is not yet possible to show whether this goal has been achieved. Future work could show the number of errors before and after the migration at Adyen to see if the desired outcome will be reached.

Also, more metrics could be used to quantitatively analyze the code reviews. In this work, we used the time to review and the number of code comments, but more information on code reviews could be analyzed to compare the models and interdependencies can be explored.

Bibliography

- [1] Maurício Aniche, Christoph Treude, and Andy Zaidman. How developers engineer test cases: An observational study. *arXiv preprint arXiv:2103.01783*, 2021.
- [2] Maurício Aniche, Christoph Treude, Igor Steinmacher, Igor Wiese, Gustavo Henrique Lima Pinto, Margaret-Anne Storey, and Marco Aurélio Gerosa. How modern news aggregators help development communities shape and share knowledge. In *Proceedings of 40th International Conference on Software Engineering (ICSE)*, 2018. doi: 10.1145/3180155.3180180.
- [3] Dimitrios Athanasiou, Ariadi Nugroho, Joost Visser, and Andy Zaidman. Test code quality and its relation to issue handling performance. *IEEE Transactions on Software Engineering*, 40(11):1100–1125, 2014.
- [4] Alberto Bacchelli and Christian Bird. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 712–721. IEEE, 2013.
- [5] Vipin Balachandran. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 931–940. IEEE, 2013.
- [6] Earl T Barr, Christian Bird, Peter C Rigby, Abram Hindle, Daniel M German, and Premkumar Devanbu. Cohesive and isolated development with branches. In *International Conference on Fundamental Approaches to Software Engineering*, pages 316–331. Springer, 2012.
- [7] Christian Bird and Thomas Zimmermann. Assessing the value of branches with what-if analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, pages 1–11, 2012.
- [8] Christian Bird, Peter C Rigby, Earl T Barr, David J Hamilton, Daniel M German, and Prem Devanbu. The promises and perils of mining git. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 1–10. IEEE, 2009.

- [9] Amiangshu Bosu, Michaela Greiler, and Christian Bird. Characteristics of useful code reviews: An empirical study at microsoft. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, pages 146–156. IEEE, 2015.
- [10] Scott Chacon and Ben Straub. *Pro git*. Springer Nature, 2014.
- [11] Marco di Biase, Magiel Bruntink, and Alberto Bacchelli. A security perspective on code review: The case of chromium. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 21–30. IEEE, 2016.
- [12] Alessio Ferrari. Empirical methods in software engineering, 2020. URL <https://www.youtube.com/playlist?list=PLSKM4VZcJjV-P3fFJYMu2Oh1TjEr9Bj10>.
- [13] Georgios Gousios. The ghtorrent dataset and tool suite. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 233–236. IEEE, 2013.
- [14] Georgios Gousios, Martin Pinzger, and Arie van Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, pages 345–355, 2014.
- [15] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. Work practices and challenges in pull-based development: The integrator’s perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 358–368. IEEE, 2015.
- [16] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. Work practices and challenges in pull-based development: the contributor’s perspective. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 285–296. IEEE, 2016.
- [17] Toshiki Hirao, Akinori Ihara, Yuki Ueda, Passakorn Phannachitta, and Ken-ichi Matsumoto. The impact of a low level of agreement among reviewers in a code review process. In *IFIP International Conference on Open Source Systems*, pages 97–110. Springer, 2016.
- [18] Rashina Hoda and James Noble. Becoming agile: a grounded theory of agile transitions in practice. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 141–151. IEEE, 2017.
- [19] Yujuan Jiang, Bram Adams, and Daniel M German. Will my patch make it? and how fast? case study on the linux kernel. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 101–110. IEEE, 2013.
- [20] Sami Kollanus and Jussi Koskinen. Survey of software inspection research. *The Open Software Engineering Journal*, 3(1), 2009.

- [21] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. Code review quality: How developers see it. In *Proceedings of the 38th international conference on software engineering*, pages 1028–1038, 2016.
- [22] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 192–201, 2014.
- [23] Helmut Neukirchen and Martin Bisanz. Utilising code smells to detect quality problems in ttcn-3 test suites. In *Testing of Software and Communicating Systems*, pages 228–243. Springer, 2007.
- [24] Adam Porter, Harvey Siy, and Lawrence Votta. A review of software inspections. *Advances in Computers*, 42:39–76, 1996.
- [25] Peter Rigby, Daniel German, and Margaret-Anne Storey. Open source software peer review practices. In *2008 ACM/IEEE 30th International Conference on Software Engineering*, pages 541–550. IEEE, 2008.
- [26] Peter C Rigby and Christian Bird. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 202–212, 2013.
- [27] Peter C Rigby and Margaret-Anne Storey. Understanding broadcast based peer review on open source software projects. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 541–550. IEEE, 2011.
- [28] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. Modern code review: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 181–190, 2018.
- [29] Todd Sedano, Paul Ralph, and Cécile Péraire. Software development waste. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 130–140. IEEE, 2017.
- [30] Davide Spadini, Maurício Aniche, Margaret-Anne Storey, Magiel Bruntink, and Alberto Bacchelli. When testing meets code review: Why and how developers review tests. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 677–687. IEEE, 2018.
- [31] Andrew Sutherland and Gina Venolia. Can peer code reviews be exploited for later information needs? In *2009 31st International Conference on Software Engineering-Companion Volume*, pages 259–262. IEEE, 2009.

- [32] Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 141–150. IEEE, 2015.
- [33] Patanamon Thongtanunam, Shane McIntosh, Ahmed E Hassan, and Hajimu Iida. Review participation in modern code review. *Empirical Software Engineering*, 22(2): 768–817, 2017.
- [34] Jason Tsay, Laura Dabbish, and James Herbsleb. Influence of social and technical factors for evaluating contribution in github. In *Proceedings of the 36th international conference on Software engineering*, pages 356–366, 2014.
- [35] Kristín Fjólá Tómasdóttir, Maurício Aniche, and Arie van Deursen. The adoption of javascript linters in practice: A case study on eslint. *Transactions on Software Engineering (TSE)*, 2018. doi: 10.1109/TSE.2018.2871058.
- [36] Arash Vahabzadeh, Amin Milani Fard, and Ali Mesbah. An empirical study of bugs in test code. In *2015 IEEE international conference on software maintenance and evolution (ICSME)*, pages 101–110. IEEE, 2015.
- [37] Enrique Larios Vargas, Maurício Aniche, Christoph Treude, Magiel Bruntink, and Georgios Gousios. Selecting third-party libraries: The practitioners’ perspective. In *Proceedings of the ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2020. doi: 10.1145/3368089.3409711.
- [38] Andy Zaidman, Bart Van Rompaey, Serge Demeyer, and Arie Van Deursen. Mining software repositories to study co-evolution of production & test code. In *2008 1st international conference on software testing, verification, and validation*, pages 220–229. IEEE, 2008.

Appendix A

Interview Before the Migration

This appendix contains the interview script used for the interviews before the migration.

Interview Questions (Before Migration) [25 min]

- Introduction [2 min]
 - Explain the purpose of this study
 - Mention that the interviewee will remain anonymous
 - Ask for permission to record the interview
- Participant Information [5 min]
 - What is your professional working experience in software development?
 - How long have you been working for Adyen?
 - In which teams within Adyen have you worked so far?
 - How long have you been working in your current team?
 - What is the nature of your development work in your current team?
- Trunk-Based [10 min]
 - Can you tell me something about the current project you are working on?
 - What are the challenges you face working with the trunk-based model in your current project?
 - Are your contributions being reviewed? How?
 - What challenges do you face with the way code is reviewed in the current model?
 - Do you review other developers' contributions? How?
 - How long does it take for you to review someone else's changes? Why?
 - How long does it take for someone else to review your changes? Why?

A. INTERVIEW BEFORE THE MIGRATION

- MR knowledge [5 min]
 - Do you have experience working with a merge request-based development model? (Have you participated in the pilot for merge requests?)
 - If the answer is yes:
 - * What are the benefits of using the merge request-based model over the trunk-based model? Why?
 - * What are the challenges of using the merge request-based model?
 - * You (probably) have heard that Adyen is about to move towards a merge request-based model. What are your expectations?
- Ending questions [3 min]
 - What do you expect to be the most preferable way of developing and why?
 - (Summarize interview) Is this a fair summary of your point?
 - Anything else you would like to add?

Appendix B

Survey

This appendix contains the survey questions as provided to the participants. Only the participants filled out the survey online.



Dear participant,

Thank you for taking this survey.

This questionnaire is a part of my master thesis project where I am analyzing the benefits and challenges of the migration from the trunk-based development model to the merge request-based development model at Adyen. By collecting enough answers, the published results will include advice for Adyen and other companies in the software engineering industry on what development model to choose and how to benefit most from those models.

This survey is aimed at all developers at Adyen, both with and without prior experience in the merge request-based development model as the goal of this survey is to generalize the expectations of moving to the new model.

This survey takes around 5-10 minutes to complete and the answers will solely be used in aggregated and anonymized form.

Thank you,

Toon de Boer

Delft University of Technology

Section A: Background Information

A1. How many years of professional experience do you have in software engineering?

A2. In which team are you currently working?



A3. Do you have any previous professional experience working with the merge request-based model?

Yes

No

A4. How many years have you been working for Adyen?
Please use decimals if necessary (e.g. 1 year 6 months is 1.5 years).

--	--	--	--	--	--	--	--	--	--	--

Section B: Code Reviews

B1. Regarding the trunk-based model, how much do you agree with the sentences below?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The reviewer has to convince the author to improve their code.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
There is enough time to complete code reviews before the release.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It is a problem that developers sometimes have to wait too long for a review.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The senior reviews add value to the code reviews.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It would be better to have 2 team reviews instead of 1 team review and 1 senior review.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It is important for beginner developers to conduct code reviews as they will learn the best practices.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It is useful to get reviews assigned from other teams.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

B2. Regarding your expectations about the merge request-based model, how much do you agree with the sentences below?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Reviewers will often block the merge by placing insignificant comments.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Merge Requests makes the reviewer see a bigger picture of the review.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Review quality and therefore code quality will improve.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Less experienced developers will learn more when reviews are mandatory for code to be merged to the master branch.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It would be useful if automatic reminders are sent to the reviewers.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Reviewers should face some sort of consequences (no heavy punishment) when they do not review in time.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



Section C: Development Speed

C1. How much do you agree with the sentences below considering the development speed?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
The current release cycle is too short.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The build time of the pipeline takes too long.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It would be better to extend the release cycle or postpone the release until everything has been reviewed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
With the merge request-based model, the velocity of producing new code will decrease.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
With the merge request-based model, a feedback cycle of one week will be too short.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
With the merge request-based model, developing time gets wasted by waiting on merge request approvals.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
With the merge request-based model, some checks will have to be skipped in feature branches and only run in the master branch to increase development speed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Section D: Merge Conflicts

D1. How much do you agree with the sentences below considering merge conflicts?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
In the trunk-based model, merge conflicts happen often.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
There will be more merge conflicts in the merge request-based model.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Resolving merge conflicts is a time consuming task.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Merge conflicts are meaningful and help the code to get better.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Merge conflicts are easy to fix.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Section E: Code Quality and Safety

E1. How much do you agree with the sentences below considering code quality and safety?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
That fact that not all code in production has been reviewed is a problem.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Bugs in production could have been prevented when the responsible code was reviewed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The cost of fixing the bug is higher than preventing by doing a proper code reviews.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I have felt scared to break the system when committing code to master .	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
It is a problem that code goes live even if the reviewer disagrees with the changes being made.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A benefit of the merge request-based model is that there are less bugs and errors in the production code.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Testing code is more reliable than code reviews.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Section F : Feature Branches

F.1. How much do you agree with the sentences below considering feature branches?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Feature branches can only be maintained for one week is a disadvantage of the trunk-based model.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The trunk-based model simplifies the process of fixing bugs as it is easy to pinpoint where the error is introduced and revert that commit.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Developers will abuse the feature branches in the merge request-based model by developing for too long in one branch.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
When the merge-request is too large, the review quality will decrease as it is harder to do a proper review.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Saving code remotely on a feature branch without influencing other developers is a benefit of the MR model.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The commit history will improve in the MR model as commits to the same feature will be grouped.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
With the merge request-based model, the system should give warnings when feature branches are too long lived.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Section G : Microservices

G.1. How much do you agree with the sentences below considering microservices?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Microservices will help the codebase to scale.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Microservices will make the life of developers better.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Microservices add a lot of complexity since multiple services that are dependent on each other need to work together.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Deploying different applications independently would benefit the entire platform.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The release cycle of microservices is faster than that of a mono repository.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Working with smaller repositories would make it easier for beginners during the onboarding.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
There is more value in a mono repository as there are currently a lot of interdependencies.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Adyen should break up the codebase and use containerisation and microservices.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Section H: Development Models

H1. How much do you agree with the sentences below considering the development models?

	Does not apply	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Every team should decide for themselves whether to use to trunk based model or the MR model.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Senior developers should have the ability to merge directly without a review.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The trunk-based model is preferred over the MR model for startups and fast developing codebases.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
The MR model should be used for mature codebases and high impact projects.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
For Adyen, using the MR model has more benefits than the trunk-based model.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A hybrid model such as the MR model allowing direct commits would be preferred over using the MR model.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Section I: End

I1. If you have any suggestions on how to improve the merge request model for Adyen, please add them here



12. If you have any comments on the survey or on the topic, please add them here.

Thank you for participating in this survey!

Appendix C

Interview After the Migration

This appendix contains the interview script used for the interviews after the migration.

Interview Questions (After Migration) [25 min]

- Introduction [2 min]
 - Explain the purpose of this study
 - Mention that the interviewee will remain anonymous
 - Ask for permission to record the interview
- Participant Information [5 min]
 - What is your professional working experience in software development? (years)
 - How long have you been working for Adyen?
 - In which teams within Adyen have you worked so far?
 - How long have you been working in your current team?
 - What is the nature of your development work in your current team?
 - Did you have experience with merge requests before you started at Adyen?
- General Questions [4 min]
 - What are the benefits you experience from using MR?
 - What are the challenges you experience from using MR?
- Migration [2 min]
 - How did you experience the migration from trunk based to merge request based?
- Merge Conflicts [2 min]
 - Do you experience more merge conflicts and how do they compare to the conflicts in trunk-based development?

C. INTERVIEW AFTER THE MIGRATION

- Code Review [5 min]
 - What has changed in code reviews when comparing the merge requests with the code reviews in Upsource? (quality, content, size, speed)
 - Could you give an example of how the code review has changed?
- Development Speed [2 min]
 - How does the new model compare to the old model in terms of development speed?
- Ending questions [3 min]
 - Having experienced both models at Adyen, what do you think is the most preferable way of developing and why?
 - (Summarize interview) Is this a fair summary of your point?
 - Anything else you would like to add?