UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering Curriculum

Kiryl Lashkevich

Improving Agile Processes with Customized Mission-based Practices. Case Study

Master's Thesis (30 ECTS)

Supervisors: Fredrik Payman Milani

Mario Ezequiel Scott

Tartu 2020

Improving Agile Processes with Customized Mission-based Practices. Case Study

Abstract:

There is no silver bullet when it comes to software development approaches, methodologies and practices, since every IT business is unique. Selecting a suitable software development model is one of the recipes for effective IT company functioning and growth. However, quite often in order to keep up with company scaling the initially chosen model has to be revisited.

In this paper, a case study about a new Agile software development framework adoption for scaling engineering at Pipedrive will be presented. The main research goal is to find out what the impact of such software development process redesign is.

Keywords:

Agile Software Development, Software Product Management, Scaling Engineering, Scaling Agile, Software Development Redesign, Scrum, Tribes, Missions

CERCS: P170 Computer science, numerical analysis, systems, control

Väleda tarkvara arendusprotsessi parandamine kohandatud Missiooni-põhiste võtetega. Juhtumiuuring

Lühikokkuvõte:

Kuna iga infotehnoloogia (IT) ettevõte on unikaalne, ei leidu hõbekuuli tarkvara arendusprotsesside, -meetodite ja -tavade valdkonnas. Sobiva tarkvaraarenduse metoodika leidmine on efektiivselt töötava ja kasvava IT ettevõtte üks aluseid. Siiski, ettevõtte suurenedes tuleb sageli parema skaleeruvuse nimel esialgselt valitud arendusmetoodika üle vaadata. Käesolev töö tutvustab juhtumiuuringut, mis käsitleb uue väleda tarkvaraarendamise protsessi juurutamist Pipedrive tarkvaraarenduse skaleerimiseks. Peamiseks uurimiseesmärgiks on tuvastada tarkvara arendusprotsessi ümberkujundamise mõjusid ettevõttele.

Võtmesõnad:

Väle tarkvaraarendus, tarkvara tootejuhtimine, tarkvaraarenduse skaleerimine, väle suures mastaabis, tarkvaraarenduse ümberkujundamine, scrum, hõimud, missioonid

CERCS: P170 Arvutiteadus, arvanalüüs, süsteemid, kontroll

Table of Contents

1 Introduction

Agile software development principles are frequently adopted nowadays by IT companies around the globe. Compared to the classical waterfall development model, where all engineering activities are organized as a sequence of phases (requirements, design. implementation, verification, maintenance) and require stable product definition and the detailed set of product requirements before the implementation starts, the agile software development introduces more flexibility for the development process by enabling frequent delivery of valuable software by the means of development iterations. By embracing frequent changes instead of following a specific long-term development plan and collaborating with the customer throughout the whole development process, the adepts of the agile software development can react to the actual needs of the customer as early as possible and deliver the functionality that is in high demand today.

Even if the general agile principles are adopted, in order to be competitive, successfully maintain the growing number of employees and satisfy the need for greater and faster value delivery, modern companies must be able to scale their software development processes. However, classical agile frameworks such as Scrum prescribe relatively small teams (from 5 to 9 members). When companies scale up, such methodologies can present to be limiting. There are several popular alternatives for implementing agile at scale, such as Scaled Agile Framework (SAFe) and Large Scale Scrum (LeSS). Recently, a new tribe-based approach, as used by Spotify, has gained popularity. However, it is not clear if replacing for instance Scrum with tribe-based methodology will improve the software development process. In light of the above context, this thesis investigates the software development process performance of one fast-growing IT company, Pipedrive, that replaced Scrum-based agile software development methodology with the tribe- and mission-based one.

The thesis presents a study of the redesign of the software development process. The **research objective** is to investigate **what the impact of software development process redesign at Pipedrive (Pipedrive Agile Framework) is**.

In order to fulfill this objective, both qualitative analysis (interviews, questionnaires) and quantitative analysis (e.g. measuring Agile software development metrics) will be conducted.

1.1 Problem Statement

Pipedrive is a fast-growing IT company based in Estonia. The core company's product is the Pipedrive software as a service (SaaS) customer relationship management (CRM) tool for driving sales growth in sales teams of small and medium scale companies.

During the period 2013-2018, Pipedrive had multiple independent teams organized according to different product areas (i.e. web, mobile, support, etc.). Engineering teams were responsible for both new feature development and maintenance of existing functionalities. In addition, teams were static, specialized units, and migration of engineers between teams were rare.

At the beginning of 2018, the Pipedrive management found out that the company business and engineering growth became harder due to many issues related to the current approach. There was a clear need for important changes in internal structure and processes. After the Pipedrive management board investigated the existing issues and analyzed software

development approaches of other companies such as Spotify, Nokia HERE Maps, and Facebook, among others, the teams were replaced with more flexible tribes and the feature development was handled by dynamically formed mission teams.

Although these changes were successfully implemented at Pipedrive, there was a lack of empirical evidence of the benefits introduced. For this reason, the following research questions will be covered:

- **RQ1:** *What are the main differences between the Scrum- and Mission-based software development processes?*
- **RQ2:** *What is the impact of the changes on the software development process?*

2 Background

In this section, the background related to the field of study is presented. It includes the definition of the key concepts mentioned in this thesis, including Agile software development and scaling Agile methods.

2.1 Agile Software Development

Agile Software Development is a general umbrella term for software development frameworks and practices based on the values and principles expressed in the **Manifesto for Agile Software Development** and the **Principles behind the Agile Manifesto** [1]. In Agile software development, software requirements and solutions evolve through the collaborative effort of self-organizing and cross-functional teams and their customers [2]. This means that it is focused on the frequent and continuous delivery of high-quality software, based on regular cooperation with the customer (end-users).

There are several agile frameworks that are popular nowadays. One of the most widely used is **Scrum**, which is based on incremental working software delivery within defined iterations, called **sprints** [3]. It is focused on managing software development projects by means of strictly defined roles, meetings and process artifacts. Scrum framework has three artifacts: the product backlog, the sprint backlog and the increment.

The **product backlog** (or simply **backlog**) is one of the most popular software product development artifacts, representing an ordered list of issues (tasks, bugs, user stories, etc.) to be done in the product [4]. The **sprint backlog** is, in its turn, a list of issues (formed from the backlog) that are selected to be addressed during the sprint. The **increment** is a list of completed sprint backlog items [3], integrated with the product.

In addition to sprints, there are 4 other Scrum events:

- **Sprint planning** - the process of defining (planning) the content of the increment (spring backlog items) for the upcoming sprint;
- **Daily scrum** - a timeboxed (up to 15 minutes) daily event to identify the progress of tasks completion towards the sprint goal, appearing impediments and plans for the next 24 hours;
- **Sprint review** - the event held at the end of a sprint to review the increment and discuss the next things to be done;
- **Sprint retrospective** - the event that goes after the sprint review, but before the sprint planning to discuss what went well during the latest sprint, what could be improved, and what actions should be done to improve the process.

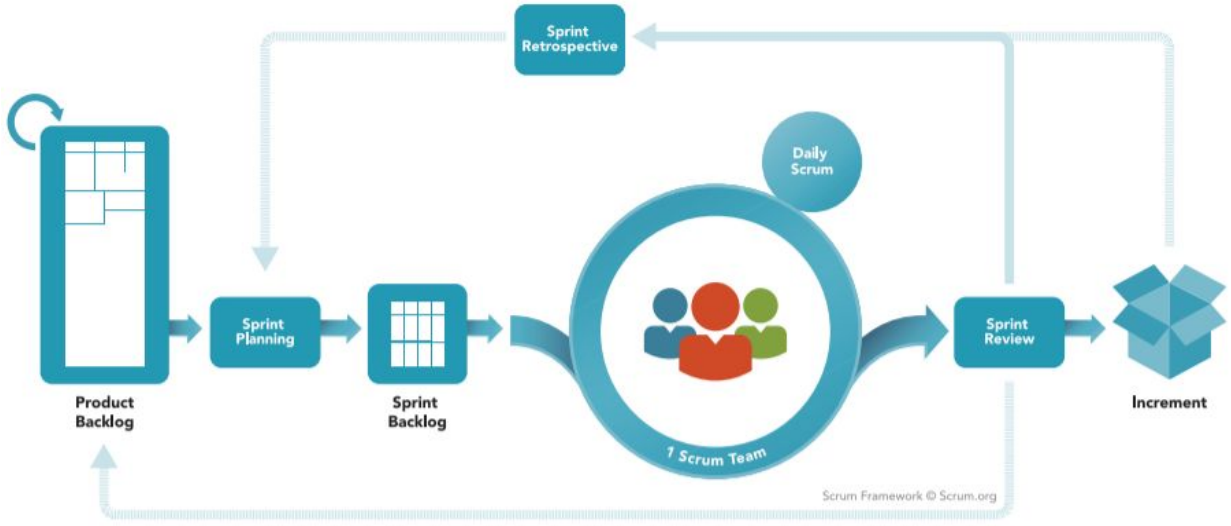

Figure 1. Scrum framework[1].

Lean Software Development is a set of software development principles and practices originated and adapted from lean manufacturing principles in Toyota corporation [5]. It is focused on creating value for customers, development time and resource optimizations, and elimination of waste. **Kanban** [6] is a Lean Software Development methodology focused primarily on the visualization of the workflow and limiting work in progress.

Scrumban is another agile framework that can be considered a hybrid methodology combining elements of Scrum (daily scrum meetings, iterations, iteration planning, etc.) and Kanban (visualization of the development flow using the kanban board, limiting work in progress).

2.2 Scaling Agile Methods

Scrum is a lightweight framework that prescribes a relatively small development team (from 5 to 9 members), which can be a limiting factor when companies scale up. In order to deal with Scrum scaling constraints, several scaling Agile methods were introduced. For instance, **Scaled Agile Framework (SAFe)** – is a set of organizing principles, allowing to apply agile methodologies to the whole organization by going beyond single-team methods like Scrum [7] and promoting alignment, collaboration, and delivery across large numbers of agile teams. There are different configurations of SAFe. For instance, SAFe 4.0 introduces 3 organizational levels [8]: Team, Program, Value Stream and Portfolio, which contain their inner activities. All levels are connected together and operate according to Agile and Lean practices. **SAFe teams** are agile, self-organizing and cross-functional. They consist of developers, the product owner and the scrum master.

Another well known **Large Scale Scrum (LeSS)** framework also specifies cross-functional teams and is suggested to be used for up to 70 people [9]. According to [9], LeSS is supposed to include a **single Product Owner** for all teams. To manage sprint planning, two representatives from every Scrum team gather together with the Product Owner. From an organizational point of view, Inter-team coordination meetings can be used to facilitate knowledge sharing between teams; Joint Light Produce Backlog Refinements are aimed to

[1] https://www.scrum.org/resources/scrum-framework-poster

refine product backlog items together; Joint Retrospective Meetings are used to plan the improvements for the product or organization as a whole.

One of the software development frameworks that was an inspiration for the software process redesign in Pipedrive was introduced by the **Spotify** company. The **tribe-based** software development model was described in [10].

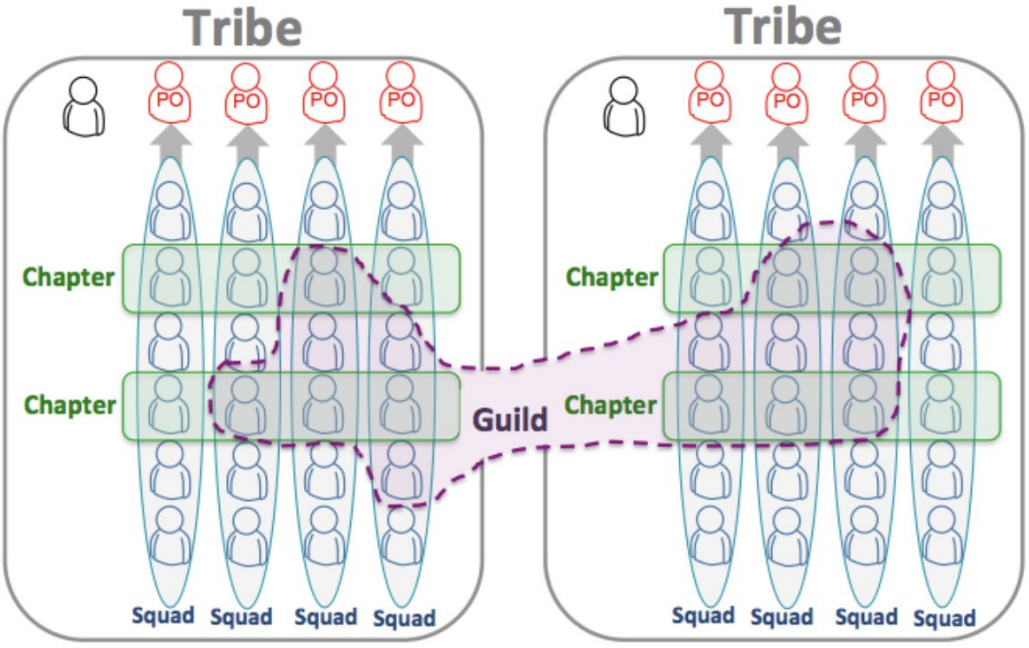

Figure 2. Spotify Framework[2].

According to [10], Spotify **squads** are self-organizing teams that are built around long-term goals related to product parts (i.e. Android client development and improvements, payment components development, etc.). Squads are able to choose their own software development methods and practices.

Each squad has a **Product Owner**, which is responsible for squad task prioritization. In addition, every squad can contact an **agile coach** to get help in improving the inner processes and the organization of meetings.

Spotify **tribes** are collections of squads working on specific large product areas (infrastructure, back-end, web application, DevOps, etc.), containing 30-200 people each and physically located in the same office. Tribes have a clear mission, a set of principles, and a senior experienced **leader** [11].

Chapters are groups of engineers with similar skill sets and competency coming from different squads (e.g. testing chapter, backend chapter, etc.). Chapters serve the goal of knowledge sharing between squads and creating tools that can be useful for all squads within a tribe. Every chapter has the **chapter lead** role for line management of chapter members regarding questions of technical competency, personal growth, etc. The chapter lead role is assigned to one of the tribe members.

[2] https://creativeheldstab.com/wp-content/uploads/2014/09/scaling-agile-spotify-11.pdf

Finally, **Guilds** are groups of people connecting different tribes to share knowledge, tools and practices at the company level.

Thus, the Spotify model can be interpreted as a sort of a matrix organizational structure with two dimensions: squad members get tasks from squad product owners and also receive guidance on how to develop these tasks from the chapter leads (see Figure 3).

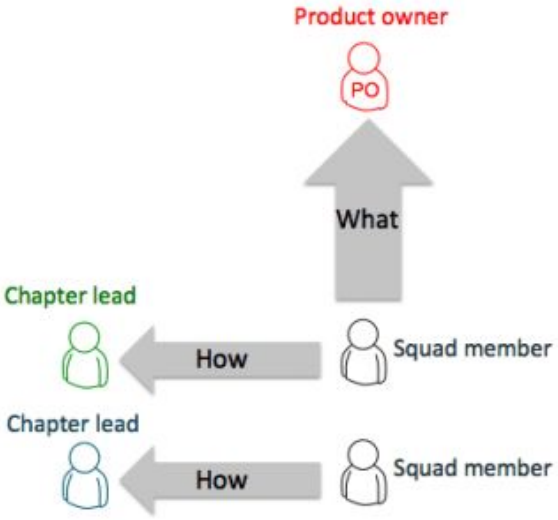

Figure 3. Two-dimensional organization of squads in Spotify Framework[3].

[3] https://creativeheldstab.com/wp-content/uploads/2014/09/scaling-agile-spotify-11.pdf

3 Related Work

In this section, the review of several papers related to the agile methods scaling is presented. There are no research papers related to the software product development in Pipedrive found in the public domain. All sources presented in this section are related to software process redesign and scaling agile software development in different companies.

Trends in scaling agile methods were presented in [12]. The covered topics include agile transformations, knowledge sharing, inter-team management in terms of scaling agile development in large multi-site organizations. The paper describes the workshop proceedings covering various topics of agile at scale: challenges of SAFe and LeSS adoption, agile transformations and business agility and inter-team communication.

The case study of tailoring the Spotify practices of autonomous teams (squads) in the outsourced large-scale project with 6 squads was reported by [13]. The study conducts 14 semi-structured interviews and direct observation practices to identify the practices to promote the effectiveness of autonomous squads. In addition, it provides the results of the adoption of Spotify framework practices in the specific project. Although chapters, squads and a two-dimensional matrix structure were successfully tailored, Spotify guilds and tribes concepts were not adopted. The study methodology used is based on the pure qualitative analysis of the conducted interviews and observations and does not cover metrics to assess the redesign quantitatively.

Another case study of large-scale agile transformation at the Ericsson company is covered in [14]. The study is based on 45 semi-structured interviews and 5 observation sessions aiming to investigate the process of agile transformation in the globally distributed organization. Starting with a plan-driven software development process, Ericsson began the transformation to agile software development in 2012. The research covers the detailed description of the company-specific agile transformation process, the challenges discovered during the transformation and the ways to mitigate them. The used methodology also does not contain quantitative measurements of software process improvements with agile development introduction.

One more case of the transition from the plan-driven development to the agile software development in the medium-sized software company is presented in [15]. The results of this case study research showed the satisfactory adoption of SAFe; the major difficulties were met in Release health and Technical health areas throughout the adoption process. The study addresses the question of SAFe adoption by using the qualitative and quantitative periodic team self-assessment. Selected research methods are surveys and observations.

The agile at scale delivery in the banking field is described in detail in [16] in the case study example of one large bank organization. The primary objective of an agile adoption was to achieve a significant improvement in the software development efficiency by increasing delivery flexibility, introducing shorter release cycles and improving the quality of products. The company was able to organize agile software development on the basis of Scrum, OpenUP and XP agile practices and concepts, as well as introduced its own business-specific practices. The agile delivery process started with 8 pilot projects involving over 100 participants. These pilot projects showed improvements in productivity and quality of the software development, and the lessons retrieved allowed to initiate further transformation

procedures in the organization. The results are presented in a qualitative form, the qualitative measurements of software process redesign are not presented.

All the cases mentioned above note that applying software process redesign is complicated and not straightforward. Every business is unique and requires a lot of research and company-specific adjustments in order to successfully redesign software development.

In contrast to this thesis, the mentioned papers present mainly qualitative outcomes of different software process redesign cases and do not show the quantifiable impact of the redesign compared to initial models. The hybrid research methodology based on both qualitative and quantitative analysis, as used in this thesis, was mentioned only in [15].

4 Research Method

This section covers the research method background and clarifies the research questions. Based on the research questions, the study design is formulated.

4.1 Background

Pipedrive is an international sales software development company with offices in Estonia (Tallinn, Tartu), USA (New York, St. Petersburg), Portugal (Lisbon), UK (London) and Czech Republic (Prague). The core company's product is the Pipedrive software as a service (SaaS) customer relationship management (CRM) tool. Pipedrive has the web application and mobile clients (iOS and Android). According to [17], **CRM** is the process of managing all aspects of the interactions a company has with its customers, including prospecting, sales and service. **CRM tools** are used to manage customer relationships in an organized way [17] and drive sales growth. **Software as a service (SaaS)** licensing and delivery model means that users pay for the software on a subscription basis [18]. In contrast to traditional software, SaaS publishers usually tend to frequent value delivery (new feature deployments) as soon as they are ready [18]. Pipedrive was founded as a startup in 2010 by 5 experienced salesmen and developers. The innovative characteristics and the simplicity of the product led Pipedrive to wide recognition and popularity among the target customers - salespeople. The Pipedrive SaaS software is currently used by more than 90000 companies worldwide. Currently, there are more than 600 employees at Pipedrive around the globe, and most of the workers are IT specialists.

The business scaling of Pipedrive has been a continuous process right from the start. It appeared that together with the growth of the company and the increasing amount of development areas, software product management and development approaches were changing and evolving as well. Development processes that worked very well at the beginning, when Pipedrive was a small startup, could not work that well after a certain stage of the company's growth. Starting from a small self-organized team, Pipedrive implemented agile practices with multiple independently working teams after 3 years. This approach was used effectively during the next 5 years, and many of the new engineering and management practices were successfully adopted and are still in use in the company today. In 2018, the company faced several issues that hindered further company scaling. As a result, the Pipedrive management board decided to introduce the substantial changes to the company-wide Agile software development process by introducing customized mission-based practices.

4.2 Research Questions

As mentioned in the introduction, the research objective of the studies is to investigate what impact of the redesign is on the software development process at Pipedrive. To address this objective, two research questions are formulated as follows:

RQ1 (*What are the main differences between the Scrum- and Mission-based software development processes?*) covers the qualitative analysis of software product development in Pipedrive. Answering this question will help to get a more precise understanding of how the software development in Pipedrive worked before implementing the mission-based practices, what problems were identified, and what changes came with the new framework.

To address **RQ2** (*What is the impact of the changes on the software development process?*), a quantitative analysis was conducted to find out if the process redesign is able to work successfully with the engineering scaling and solve the issues the initial model had. If the transition helped, we can also assess how much.

4.3 Study Design

In this section, a high-level description of the research procedures is presented. Figure 4 shows a conceptual diagram of the steps applied.

In order to address RQ1, and get an understanding of the software development process used before the mission-based practices were implemented, different sources were analyzed, including:

- Internal company documents and articles (related to the software development process instructions, migration from teams to tribes, mission- and tribe-based practices, the issue tracking guidelines, etc.); two articles in public domain written by employees [19][20].
- Internal surveys (two rounds) to understand the specifics of software development in Scrum-based and mission-based frameworks. The surveys are organized on the basis of another survey designed to understand the software development approaches in companies [21], and contain mostly multiple-choice questions about the software development framework and practices in Pipedrive.
- Internal unstructured interviews (organized as meetings and on-the-go questions discussions).

The analysis of these sources helps to answer **RQ1** and identify the problems that Pipedrive had before the changes were implemented.

Based on the answer to RQ1, the **Goal Question Metric (GQM)** [22] approach was applied to quantitatively analyze the impact of the changes applied and address **RQ2**. According to the GQM approach, the first step to determine software development metrics is to specify high-level business goals. Secondarily, the questions to assess the achievement of the specified goals should be formulated. Lastly, the metric or set of metrics is connected with every question in order to find the quantitative answers. The metrics were chosen by taking into account that they should be applicable to the two stages of the software development evolution in Pipedrive: the stage where the software development was based on few teams (**Scrum-based framework** period) and the stage where the mission-based practices were implemented (**Mission-based framework** period).

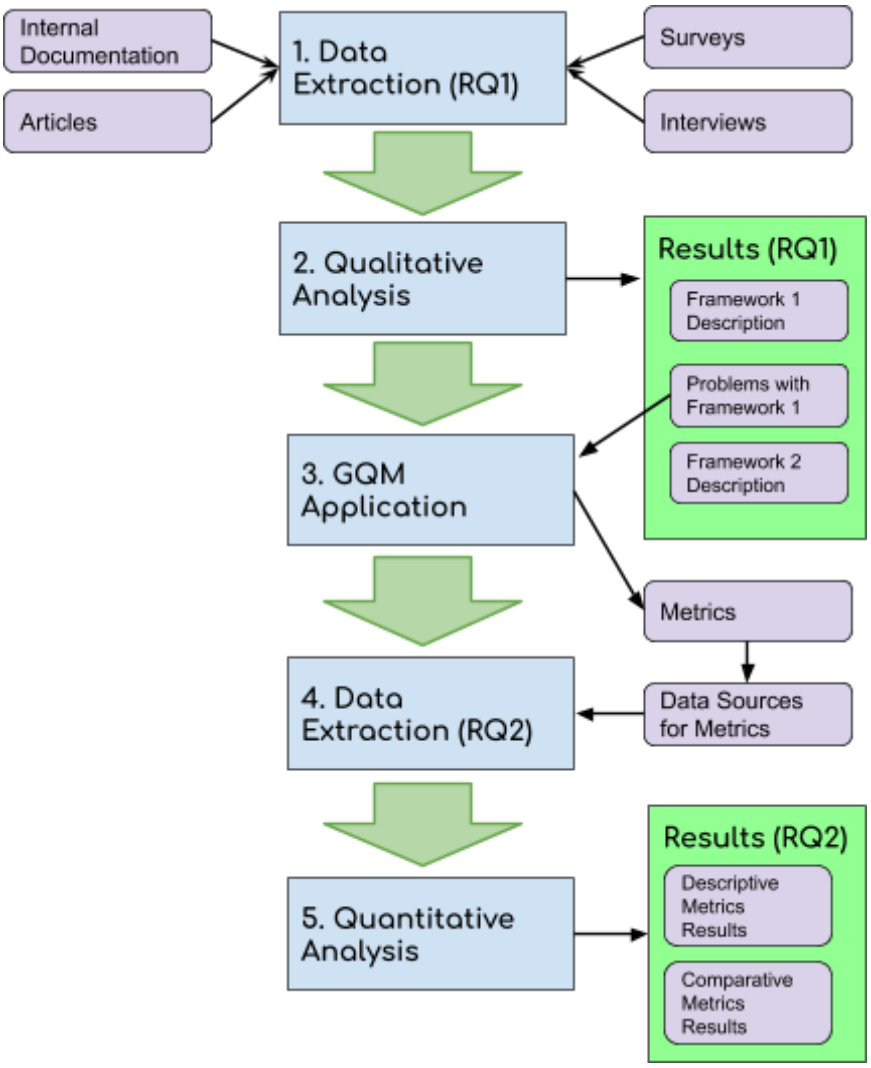

Figure 4. High-level study design.

The results for RQ1 and RQ2 are presented in Chapters 5 and 6 respectively.

5 Qualitative Analysis Results (RQ1)

In this section, the outcomes of the research activities related to RQ1 "*How was the software development process redesigned?*" are presented.

5.1 Qualitative Analysis Execution

The overall picture of the Scrum-based framework and Mission-based framework was retrieved from the internal Pipedrive documentation hosted on the Atlassian Confluence[4] platform. During the data extraction, 19 Confluence articles were read and analyzed. In addition, 2 published articles ([19][20]) were reviewed. The covered topics include Scrum-based and Mission-based framework concepts and processes, teams/tribes product areas, product management retros and discussions, internal software development process improvements, issue management and tracking guidelines, Pipedrive history and product strategy, software development metrics.

To support the statements found during the articles reviews, two rounds of surveys were conducted. Both rounds were based on the survey designed to understand the development approaches used in software organizations [21]:

1) The first round of the survey[5] was conducted in October 2019 to get more information about how the actual process worked before the process redesign (Scrum-based approach). The respondents of this survey are five experienced product managers and technical leads who worked at Pipedrive before the mission-based practices were introduced. In order to get a bigger picture, the respondents were selected from different teams.
2) The second round of the survey[6] was conducted in October-November 2019 to find out how the development process changed after moving to the tribe- and mission-based organization. The respondents were product managers from 5 different tribes (5 respondents in total) to see what approaches are common or unique on tribe scale. The questions mainly remained the same as in Survey 1, but all of them referred to the state-of-the-art agile framework.

The surveys mostly contain multiple-choice questions to assess to which extent are various software development frameworks, methods and practices presented in both agile frameworks, which of them are company-wide, and how successfully they are combined.

Finally, to cover the missing information and enrich the qualitative data with more details, unstructured interviews were conducted with Pipedrive employees. The unstructured interviews were held in the form of 1-on-1 meetings and on-the-go questions discussions. The interview meetings were conducted with 2 Pipedrive employees from engineering and product management. Since the interviews were unstructured, there were no predefined questions, though the examples of the guideline topics are the following:

[4] https://www.atlassian.com/software/confluence
[5] https://bit.ly/2Wg63F2
[6] https://bit.ly/35P4iBR

- How did the Scrum-based teams work? What deviations were met compared to the classical Scrum?
- What issues were met in the Scrum-based approach?
- How was the transition to the Mission-based software development organized?
- What changes came with the new framework? How is the software development organized in tribes?
- What improvements are gained with the Mission-based framework?
- What software development metrics were/are in use in Scrum-based and Mission-based development? Where can we find the relevant data to quantitatively assess the software process redesign in Pipedrive?

On-the-go discussions were conducted with several Pipedrive employees (more than 10) to address the issues that were raised during the thesis development (e.g. incomplete information, limited access to the data sources, etc.).

5.2 Scrum-based Practices at Pipedrive

The results from the analysis of the internal documents and articles show that, at the beginning of Pipedrive (2010-2013), when the team was relatively small (up to 10 engineers), the development process was straightforward: the team was using Asana[7] platform for work management [19]. These days, product features were being developed iteratively, but they were integrated into the product itself and released to the customers only when they were ready as a whole. The feature delivery process was very unstable because of the frequently changing priorities: one large feature release could be postponed for a long time, even in the last stage of development, if more urgent tasks appeared. In addition, frequent switching between unfinished tasks also led to a loss of focus, which could badly affect the overall productivity and quality of the product.

Clearly, that approach was not scalable for the developing company in the long run. Thus, for such purposes, **Scrum** Agile framework [3] was adopted as a base for software product development, which allowed more development flexibility by introducing multiple small development teams. In addition, quick product development iterations provided frequent value delivery for customers and, henceforth, up-to-date feedback. By this time, Pipedrive moved all the work management and tasks tracking to the **Atlassian Jira**[8] software development tool. This time, the development resources were separated according to platforms: 3 teams were initially organized: Web, Android and iOS development teams. According to the internal documents, until the end of 2014, all development teams reported directly to the chief technology officer (CTO).

Later, in 2015, the Web team was divided into four smaller independent teams. From an architectural perspective, the Pipedrive product codebase started migration from monolithic to microservice architecture [23], which helped to deliver features more granularly and independently. All the teams were supposed to be relatively small (as prescribed by Scrum), full-stack and cross-functional, and have developers, a QA specialist and a product manager. From the engineering management perspective, **the team manager** role was introduced as an additional layer of management between teams and CTO. In addition to coding, team

[7] https://asana.com/
[8] https://www.atlassian.com/software/jira

managers were responsible for team-level technical decisions, hiring, mentoring and promotion.

According to the internal documentation and articles, the general approach to all Pipedrive projects was to use the **Scrum** framework [3] elements for the product development. Teams used Backlog Management and iteration planning procedures to define a prioritized list of tasks for development and delivery within each sprint. For planning, Expert/Team based estimation (e.g. Planning Poker) was often used. Tasks progress, appearing issues and plans for the working day were discussed on a daily basis during Daily Standups (Daily Scrum meetings). The project features acceptance was often organized in the form of **Iteration/Sprint Reviews.** Sprint results were discussed during **Retrospective** meetings to find out what worked well, what could be improved and create a plan for further development and process improvements. In order to track iteration progress and see work left to do versus time, Burn-Down Charts were often used.

The results of the conducted survey (Survey 1) supports the document analysis. The responses show that the following practices were used in different projects: Design Reviews, Limit Work-in-Progress (e.g., using Kanban board), Prototyping, Scrum of Scrums, User Stories (as Requirements Engineering Practice), Velocity-based planning. From a development perspective, the following practices were commonly used: Automated Unit Testing, Code review, Coding standards, Collective code ownership, Code Refactoring, Continuous deployment. Less frequently used practices included: End-to-End (System) Testing, Pair Programming and Security Testing. The respondents also mentioned that the software development process was defined company-wide, although there could be some rare deviations in specific projects. For instance, one of the projects launched to develop multi-datacenter infrastructure to serve Pipedrive's customers was 3 years in development, which could not be managed properly with existing Scrum practices. For this purpose, this project team had to follow customized management approaches.

While Scrum methodology usage was a core approach to all Pipedrive projects, it could sometimes be combined with elements of different methodologies and frameworks (i.e. Lean, Kanban, etc.). Most of the survey respondents agreed that they actually combined different development approaches in the development of one project. For instance, in addition to Scrum core, Lean Software Development and DevOps were very often used frameworks/methods these days; Domain-Driven Design, Feature Driven Development (FDD), Kanban, ScrumBan were also sometimes used in various teams.

As seen from the survey, all these chosen combinations of development frameworks, methods, and practices evolved as learning from past projects over time, and overall goals that the management aimed to address with this selection and combination of development approaches included: improved planning and estimation (cost, time, etc.); improved frequency of delivery; improved external product quality (perceived by customers); improved productivity (reduction of effort, cost, etc.) and time-to-market.

According to the developers' feedback [19], the usage of Scrum practices improved product development significantly, since it allowed the better vision of the goals, easier tasks prioritization and much more focus on the specific tasks.

Scrum-based teams used story points for software development estimation. In order to measure team productivity and effectiveness of process changes, several metrics were used,

such as **sprint velocity, commitment and delivery** [19]. For instance, Figure 5 shows how the number of delivered story points comparing to the committed (planned) ones were progressing and stabilizing during 7 sprints in one of the teams:

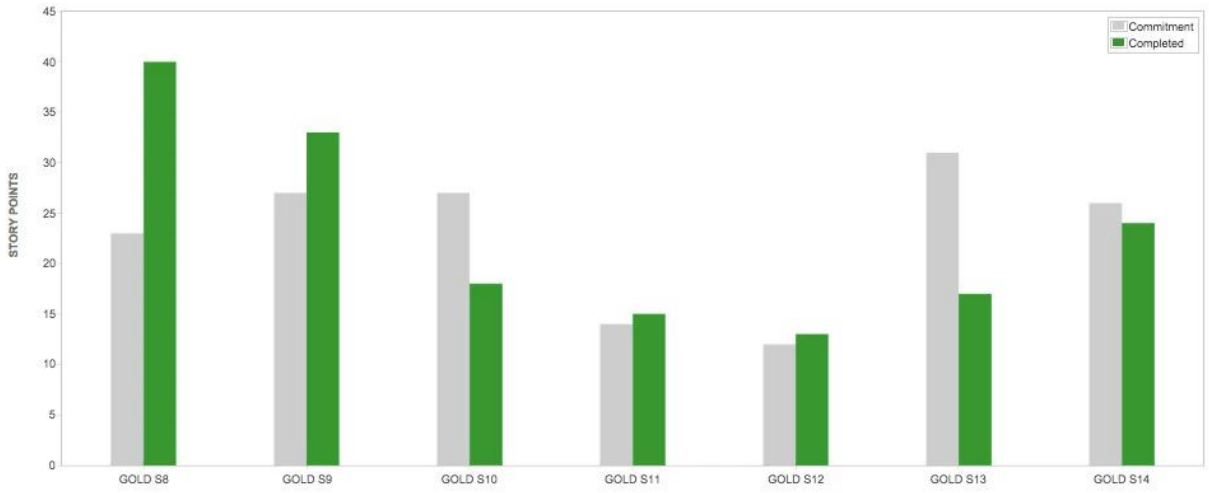

Figure 5. Story points commitment/delivery example.

Another metric was the **average cycle time** for tasks, which allowed the company to find specific places where issues in the development or task management occur. Since the sprint length was only one week, the target was to keep the average cycle time below 3 days to be able to deliver tasks within one sprint. One more metric to measure team productivity and quality of the product is the **net bug score** (the difference between defects resolved and defects registered). The bigger this value is the better is the product quality. Figure 6 shows the example of net bug score measurement in Pipedrive using Jira software.

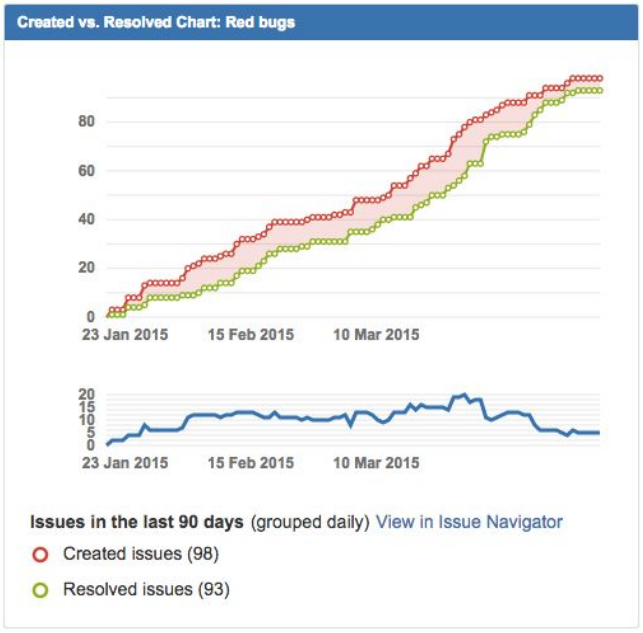

Figure 6. Net bug score graph example (extracted from internal documents).

By the beginning of 2018, the company growth required the restructuring of the engineering management: the team manager role was split into two roles - the team lead (guiding the team

development process) and the engineering manager (responsible for a bigger product area and the outcome of several teams). Moreover, engineering managers started to report to the head of engineering, responsible for even higher-level technical decisions.

From a technical point of view, Pipedrive improved development and deployment routine significantly by enforcing **continuous integration** and **continuous deployment** practices. In order to write high-quality code, developers became responsible for writing unit and integration tests for new code, which are executed every time before merging and delivery. As a result, also thanks to microservice software architecture, Pipedrive was able to build a fully **automated release process** for all of its components, allowing it to easily develop product components independently within teams, automatically test and deliver them to the live environment. In addition to mentioned full-stack teams, additional **supporting teams** (DevOps tooling team, support engineering team, etc.) were established to allow development teams mainly focus on software development.

5.3 Problems with Scrum-based Framework

According to [19] and interviews, the original software development model worked really well in some aspects: small independent development teams were able to work in parallel on separate product areas (microservices) and deliver product updates on the Scrum sprints basis; deployments and testing became really fast and fault-tolerant (Pipedrive engineers were able to do up to 60 live deployments a day); development teams efforts were covered by efficient support teams activities. However, with the further company growth there appeared the following core problems (elicited from the internal documents, interviews and surveys):

Problem 1: **Poor focus on product development** and **poor motivation**. Teams were responsible not for feature deliveries, but also for the maintenance of previously released features. This problem became even worse if a team had a lot of different features delivery [20].

Problem 2: Mobility. In small development teams, everybody had specific responsibilities. Teams back then were relatively static structures, and moving of engineers between teams and projects was complicated and often required lots of agreements and re-estimation.

Problem 3: Poor results. While the number of new employees kept growing, the value delivery and monthly customer growth started decreasing. It became clear that Pipedrive should consider significant changes in the product development model.

Additionally, the surveys and internal articles mention the problems of complicated management and silo mentality in engineering. From the management perspective, by 2018 it became hard to make fast development decisions due to many stakeholders (management layers). Moreover, since teams were closed and static formations, product ownership became not flexible. All the planning was made inside specific teams, and planning of global changes in large areas of the product became hard. This problem also led to bad knowledge sharing.

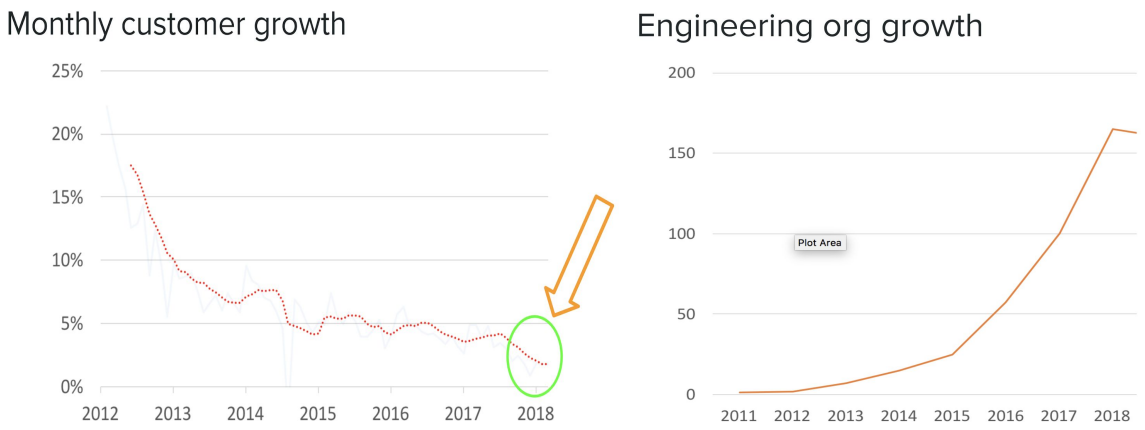

Figure 7. Pipedrive monthly customer growth vs. engineering growth.

The internal documentation shows that in April 2018, the plan of product development framework change was initiated. The management board spent some time analyzing how other software product companies dealt with processes scaling. The resulting model was inspired by engineering practices from Spotify, Facebook, Nokia HERE Maps division, Spotify, Facebook, Airbnb and other companies, and adapted for specifics of the company.

5.4 Mision-based Practices at Pipedrive

5.4.1 Basic Concepts

The information about the basic Mission-based framework concepts was compiled from the internal documentation and articles [19][20] and supported by interviewing the employees.

Mission-based framework (also known as Pipedrive Agile Framework) - is an internal **tribe- and mission-based** software development framework established in Pipedrive in 2018. The foundation concept of the framework is the tribe. **tribes** are the biggest engineering units responsible for certain parts of the product. In addition to product ownership, a tribe holds the knowledge of its product area. For instance, the Tartu tribe owns product components such as billing, global search, security and permission sets, etc. Each Tribe is led by an **engineering manager** and has product managers. Engineering managers report to the relevant head of engineering.

Every tribe organizes itself into smaller teams called **mission teams** that serve single business goals. The **mission** is a set of software product management and development activities aimed to fulfil a specific business goal (e.g. the development of a new product component). Missions are short-term engagements (usually 1 - 3 months), and the mission teams are dynamic structures that in addition to software engineers contain also a PM and a Designer and exist until the goal is met (the mission is landed) or the estimated time is over. After that, engineers return to another team called **launchpad**. A mission is led by a **mission lead**, which is a role for an engineer in a tribe. Engineers can choose the missions to participate in and **even work in other tribes** in case their competency is required. Thus, missions can be cross-tribe. When working on the mission, developers are focused only on the mission goal, there are no tasks being done in parallel.

The mission process is organized into 3 phases:

- **Preparation phase**, that includes the problem identification (based on the customer's problems and opportunities), validation (to understand how many customers are affected by the problem) and the ideation (to find out the possible solutions of the problem, features to be implemented, mock-ups, etc.). The preparation phase ends with the mission pitch to make Pipedrive employees aware of the mission and invite missing mission team members to join.
- **Execution phase**, including gathering the mission team (using the internal **mission tracking tool**), the validation of the solutions by the team according to customer's feedback (optional), building solutions (solution design and development), and the mission landing (finalization of the development, including bug fixing, additional test coverage, documentation, etc.).
- **Learning phase**, that includes the tech and process retro (the mission team discussion of how well the mission went in terms of team efforts and what are the takeaways and action items for further missions), the data gathering (optional stage organized by the product manager used to retrieve and check the usage of the new functionality), and the biz retro (the event conducted by the product manager to discuss the outcomes of the functionality delivery based on its usage information).

Figure 8. Pipedrive Mission Framework[9].

A **launchpad** is a separate team in each engineering tribe which includes engineers that are not on a mission. Its purpose is mainly to support missions (help landing, fix problems after mission landing, etc.), handle smaller product improvements, tasks and defects. Every Launchpad team has its **launchpad lead**, which is responsible for keeping launchpad processes running, for organizational aspects (meetings and discussions), backlog management and prioritization and negotiations with the tribe's engineering manager regarding mission resources. This role is not constantly assigned to a specific person, because the launchpad lead is free to temporarily leave the launchpad and go on a mission. The Launchpad lead is usually chosen by launchpad teams.

[9] https://medium.com/pipedrive-developers/scaling-pipedrive-engineering-from-teams-to-tribes-8f14fd92df8c

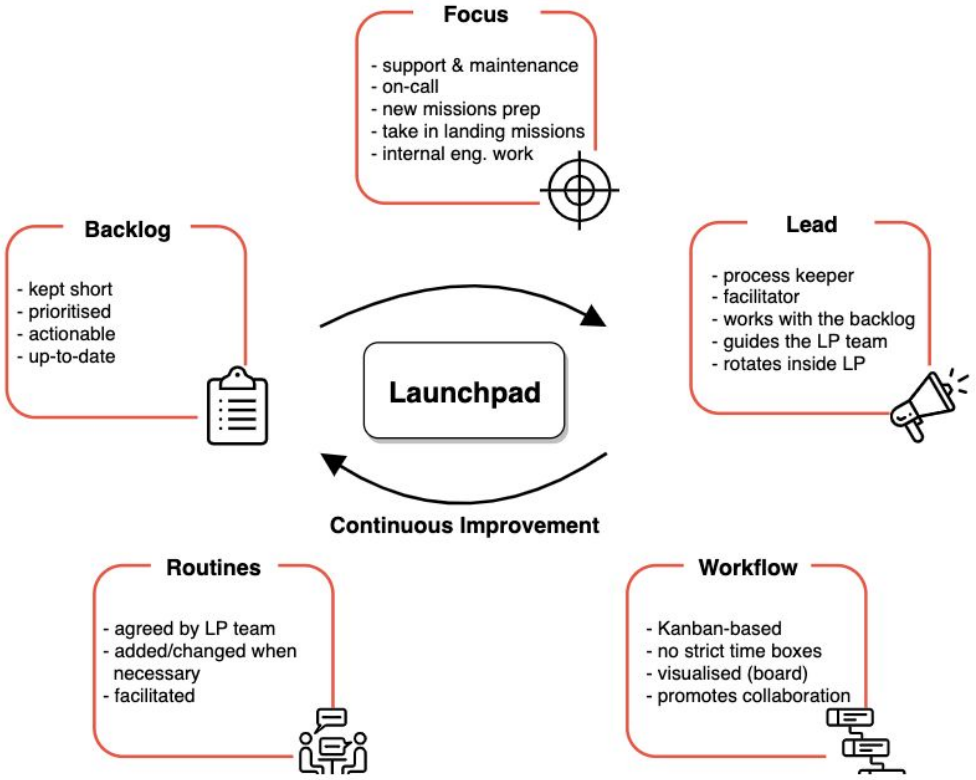

Figure 9. Launchpad concepts.

The core recommendation for launchpad teams inside tribes is to use **Kanban** principles (manage tasks, spot weaknesses using the Kanban Board, limit work in progress, etc.) based to ensure flexibility and focus. However, Launchpad teams are free to decide their own specific practices for the development process. **Kanban** does not prescribe any particular organizational routines, but Pipedrive Launchpad teams usually use the following Agile practices for process organization:

- **Plannings** - to set goals and plan how to reach them;
- **Grooming meetings** - to work with the backlog and prioritize it;
- **Daily standup meetings** - to support constant team alignment (conducted in the morning);
- **Retrospectives** - to reflect on the current state of things and discuss further improvements.

According to the interviews, there is no prescribed development estimation procedure for tribes: T-shirt sizing is sometimes used for the overall product roadmap and mission estimations, story points used less often in several tribes.

The knowledge transfer between tribes is organized via different events: mission pitchings, monthly company-wide update meetings, product demos, and internal technical education events organized by employees. In addition, there are also **guilds** that have a goal to contribute to cross-tribe cooperation on specific areas, such as Architecture, Front-end, Quality, Monitoring, Hiring & Onboarding, etc. A **guild** in Pipedrive is a public and open group of people interested in improving specific areas of the Pipedrive product. The core goal of **guilds** is to maintain a vision of what Pipedrive needs to get in a certain area. They can

also serve for promoting technology/process adoption in tribes, raising developers' skill levels as well as providing support for engineers.

In addition, there are some support teams that do not work on a specific product domain but maintain tribes' activities: Infrastructure Engineering and Operations, Quality Assurance, Agile and Personal coaching.

It is important to mention that although the tribe-based software development at Pipedrive was inspired by some concepts of the Spotify framework, it is significantly different. It is true that both Spotify and Pipedrive have guilds that work similarly. In addition, tribes concepts in Pipedrive are also similar to the ones introduced by Spotify: they both represent the highest level of the organizational structure, related to a certain large area of product development, and are able to establish local development methods and practices.

Other than that, these two frameworks have a lot of **conceptual differences**:

- Spotify squads are long-term and stable teams created around a specific product component. In contrast to squads, Pipedrive mission teams are temporary, usually short-term (1-2 months) teams aimed to achieve a specific goal and deliver a specified value. Once the mission is landed, the relevant functionality is maintained by the launchpad. There are no squad-like structures in Pipedrive.
- Pipedrive tribes do not have the matrix structure. There are no chapter-like units within the tribes. The knowledge sharing is provided by guilds, tribe-wide and company-wide events (e.g. mission pitching, demos, etc.).
- Spotify squads are responsible for both new features development and the maintenance of the existing (there is no equivalent to Pipedrive mission teams). These two types of development activities are conceptually and structurally separated in Pipedrive tribes with the help of mission teams and the launchpad.

The high-level representation of the Pipedrive tribes- and mission-based is shown in Figure 10.

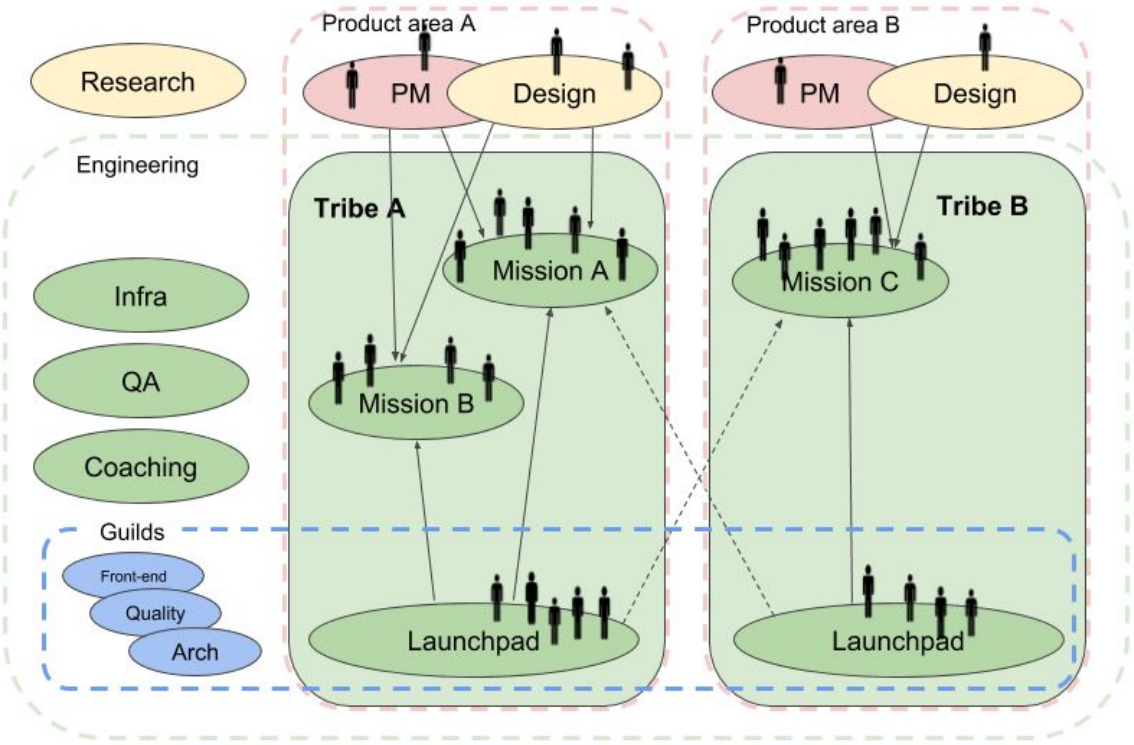

Figure 10. Pipedrive Agile Framework. Overall Picture.

5.4.2 Practices in Tribes

According to the second round of the survey, the variety of practices used in the Mission-based framework (29 reported practices) is wider than in the Scrum-based framework (23 reported practices). All these chosen combinations of development frameworks, methods, and practices evolve as learning from past missions over time, and overall goals that the management aimed to address with this selection and combination of development approaches included: Improved client involvement, Improved project monitoring and control, Improved reuse of project artifacts, Improved frequency of delivery (rapid deliveries); Improved external product quality (perceived by customers); Improved adaptability and flexibility of the process to react to change, Improved productivity (reduction of effort, cost, etc.); Improved time-to-market, Improved staff education and development (personal skill development).

As was mentioned before, although core Pipedrive organizational concepts (tribes, missions, launchpad, etc.) stay the same in all departments, the specific software development practices, methods and approaches can be selected locally by tribes. As in the case of the original model, every respondent agreed that they combine different development approaches in the development of one project.

The survey also shows that the following practices are commonly used in different tribes: Backlog Management, Daily Standup, Definition of done / ready, Design Reviews, Iteration/Sprint Reviews, Prototyping, User Stories (as Requirements Engineering Practice). More rarely used practices include: Burn-Down Charts (as Progress Monitoring Practice), Expert/Team based estimation (e.g. Planning Poker), Iteration Planning, Limit Work-in-Progress (e.g., using Kanban board), On-Site Customer, Velocity-based planning.

From a development perspective, the following practices are commonly used: Automated Unit Testing, Code review, Coding standards, Collective code ownership, Continuous deployment, Continuous integration, Code Refactoring, Release planning, Retrospectives, Security Testing. More rarely used practices include: Architecture Specifications, End-to-End (System) Testing, Model Checking, Pair Programming, Test-driven Development (TDD).

5.5 Key Framework Differences

To sum up the qualitative information and get an understanding of the key differences between the Scrum-based and Mission-based approaches (address RQ1), Table 1 is created.

Table 1. Differences between the Scrum- and Mission-based software development.

Aspect	Scrum-based Framework	Mission-based Framework
Basic organizational units	The only basic organizational unit is a **team**. Teams are autonomous and cross-functional units up to 9 people. Team composition (members, size) rarely changes.	The highest level unit is a **tribe.** Tribes size is bigger than Scrum-based teams, the number of members is not constrained explicitly. Tribes organize themselves into the **mission teams** (temporary development teams existing until a certain business goal is fulfilled) and the **launchpad** (for engineers that are not in the mission). Launchpad composition changes frequently.
Value delivery	The product increment is delivered within timeboxed **sprints** (1 week long).	**Missions** are used to deliver value. The duration of missions is not fixed and depends on the relevant functionality and product area. Launchpad activities are usually not limited to strictly defined iterations.
Development estimation	Story points are used as estimation units. Estimation techniques, such as Planning poker, are used in teams.	There is no prescribed estimation procedure. T-shirt sizing is popular for the overall product roadmap and mission estimations. Story points used less often in several tribes.
Internal agile practices	The software development in teams is company-wise based on **Scrum** concepts, with rare deviations. In addition to Scrum core, the elements of other frameworks/methods can be used (Lean Software Development, Feature Driven	Tribes are free to choose their own specific practices for the development process on the launchpad level, although there is a recommendation to use **Kanban** principles (manage tasks, spot weaknesses using the Kanban Board, limit work in progress, etc.) based to ensure flexibility and focus.

	Development, Kanban, ScrumBan, etc.).	
Management	Engineering management organized as follows: 1. CTO; 2. Head of engineering; 3. Engineering manager (responsible for several teams); 4. Team lead; 5. Team engineers.	There are the following engineering management layers: 1. CTO; 2. Head of engineering; 3. Engineering manager (head of tribes); 4. Tribe engineers (with rotating mission lead and launchpad lead roles).
Organizational events	Scrum events are prescribed: sprint planning, daily scrum meetings, sprint reviews, retrospective meetings.	Recommended events for tribes are plannings, grooming meetings, daily standup meetings, retrospectives.
Collaboration of teams/tribes	Teams are narrow-focused, the complex cross-team solutions are hard to deliver.	Cross-tribe missions help to deliver complex solutions and resolve the problem of a missing skill set required by the mission.
Knowledge sharing	The knowledge sharing events are conducted mainly in the form of technical exchange events organized by employees. There are no specific units (guilds) for cross-team knowledge sharing and cooperation.	The knowledge sharing events are conducted in the form of mission pitchings, monthly company-wide update meetings, product demos, and internal technical exchange events organized by employees. In addition, Pipedrive **guilds** help to prevent knowledge silos by contributing to cross-tribe cooperation on specific knowledge areas.

6 Quantitative Analysis Results (RQ2)

In this section, the outcomes of the research activities related to RQ2 "*What is the impact of the redesigned software development process?*" are presented. First, the execution of the qualitative analysis based on GQM is presented. Then, the data extraction and analysis procedures are described. Finally, the results are presented.

6.1 Quantitative Analysis Execution: Applying GQM Approach

The quantitative analysis conducted to answer RQ2 is based on the GQM approach. This approach proposes deriving metrics after specifying high-level business goals and relevant questions helping to assess the achievement of these goals. In this thesis, the goals are defined based on the 3 of the problems that were identified for the Scrum-based framework (see Section 5.3). The remaining problems were not considered since they are difficult to quantify with the available data. Table 2 shows the defined metrics.

Table 2. Using the GQM approach to define metrics.

Goal	Question	Metric	Metric Definition
(Problem 1) Team is well focused on defined product development tasks.	How long does it take to deliver a new feature?	**Lead Time**	The time it makes for one issue to go through the entire process from start (issue creation) to finish [24]. In terms of Jira issues, by the lead time we mean the time elapsed between the issue creation date/time and date/time when the issue is resolved.
		Cycle Time	The time it takes for the story (issue) to be completed [24]. Specifically, the cycle time is the time elapsed between the moment an issue was taken in progress and the moment it was completed (closed). In terms of Jira issues, we describe the cycle time as the time elapsed between the first moment (date/time) an issue is taken in progress and date/time when the issue is resolved.
	How often do employees switch between (unfinished) tasks?	**Work in Progress (WIP)**	The number of issues the team (tribe) is working on [24]. In the context of Jira issues, We assume an issue is in progress if it is being developed or reviewed.

(Problem 1) Engineers are well motivated and engaged to do their tasks fast and qualitatively.	Are the employees satisfied with the company and its processes?	**Employee retention rate**	The percentage of employees that stayed in a company in a given period. The formula is: $ERR = \frac{E-N}{S} * 100$ (1) Here, E - number of engineers at the end of the period, N - number of new engineers hired during the period, S - number of engineers at the beginning of the period. We will only consider engineers for this measurement.
(Problem 2) Teams are flexible; moving between teams and projects is an easy process.	How often do engineers change their team/project?	**Turnover rate (newcomers)**	The number of engineers who were active in this iteration (period), and not active in the previous iteration divided by the average number of developers in these iterations [25]: $T_{newcomers} = \frac{\lvert E_i - E_{i-1}\rvert}{\frac{\lvert E_{i-1}\rvert + \lvert E_i\rvert}{2}}$ (2) Here, E_i is a **set** of engineers that were assigned to at least one development issue it this iteration (i), E_{i-1} is similar but for the previous iteration (period), $\lvert E_i\rvert$ means the **cardinality** (number of unique set elements) of the set E_i, and $\lvert E_i - E_{i-1}\rvert$ means the number of newcomers, which is the cardinality of the **difference of sets** E_i and E_{i-1}.
		Turnover rate (leavers)	The number of engineers who were not active since the previous iteration (period) divided by the average number of developers [25]: $T_{leavers} = \frac{\lvert E_{i-1} - E_i\rvert}{\frac{\lvert E_{i-1}\rvert + \lvert E_i\rvert}{2}}$ (3) Here, the equation notation is the same as in Eq. 2.

		Team Stability Index (TSI)	This measure shows how stable is the team by iterations with regards to the initial set of engineers. The metric is inspired by the requirements stability index described in [26]. $TSI_i = \frac{\lvert E_0 \rvert + \lvert E_i - E_{i-1} \rvert + \lvert E_{i-1} - E_i \rvert}{\lvert E_0 \rvert}$ (4) Here, TSI_i is the team stability index for iteration i, E_0, E_i and $E_{i-1}$are the sets of developers in a team of the first iteration, the current iteration and the previous iteration respectively. $\lvert E_i - E_{i-1} \rvert$ is the number of newcomers (Eq. 2), $\lvert E_{i-1} - E_i \rvert$ is the number of leavers (Eq. 3). In the research, we will mainly use the **inverse form of TSI**: $TSI_i^{-1} = \frac{1}{TSI_i}$ (5) TSI^{-1} can take values between 0 and 1. The higher the value of TSI^{-1}, the more stable the team is (value of 1 means that the team stayed the same).
(Problem 3) Value delivery is correspondent to the company growth.	What is the quality of new functionality?	**Number of defects**	The number of defect (bug) issues created in a specified period.
		Number of features	The number of issues related to new functionality development, created in a specified period.
		Net bug score	Number (ratio) of defect issues resolved and created within a specified period [19].
		Number of back in development cases	Number of cases a task is sent back in progress after going to further stages (code review, deployment) within a specified period.
		Number of back from	Number of cases when a task is sent back from a backlog to in progress (in progress

		backlog cases	→ backlog → in progress) within a specified period.
	How long does it take to deliver a new feature?	**Lead Time**	The time it makes for one issue to go through the entire process from start (issue creation) to finish [24].

The difference between the lead time and cycle time of the issue (task) can be seen in Figure 11.

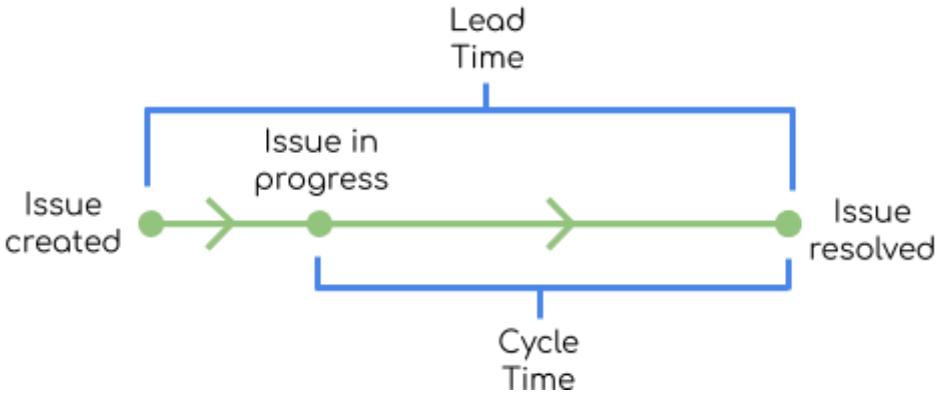

Figure 11. Issue lead time and cycle time illustration.

In terms of the issue tracking system, the **lead time** is the time elapsed between the time an issue was created and the time when it was resolved. **Cycle time** is a part of lead time, and it is calculated as the time elapsed between the moment an issue was taken in progress (development) and the moment it was resolved.

In addition to the metrics identified using GQM, several other metrics were calculated to describe the contexts of the software development approaches. Table 3 shows the metrics and their definitions.

Table 3. Additional descriptive metrics.

Metric	**Description**
Number of all employees	The descriptive measure representing a big picture of all Pipedrive employees in 1 January 2017 - 31 December 2019 period. The aim of it is to show the growing number of company employees and the number of engineers compared to employees from other departments.
Number of issue assignees	This metric represents the number of development issue assignees. Due to the data cleaning, the number of assignees is expected to be slightly smaller for every month than the number of all employees of the Engineering department, calculated as part of the previous metric.
Total number	Shows the total number of issues and issue types ratio in Scrum-based

of issues	and Mission-based frameworks, grouped by issue types.
Number of issues by months	The number of issues **created** every month. It can be grouped by issue types and frameworks (Scrum- or Mission-based).
Teams/tribes size	The number of engineers working in teams or tribes. Since the number of engineers is not a constant value, we will use mean and median values counted by months.

6.2 Data Extraction

In order to describe and compare the original Pipedrive software development framework (**Scrum-based framework**) with the redesigned Pipedrive Agile Framework (**Mission-based framework**), we retrieved the data from various sources:

- Jira[10] issue tracking software API (development projects, issues, issue changelogs, assigned engineers);
- Internal Mission Tracking Tool database (tribes, missions and their members);
- Internal BambooHR[11] human resources API (some missing data about tribes and mission members).

Most of the data analysis and CSV data source files manipulations were implemented using Python programming language and pandas[12] library in JupyterLab[13] interactive development environment (IDE). Pandas library operates with data frames, which are two-dimensional tabular data structures with rows and columns.

When collecting the data, the following assumptions and constraints were taken into account:

1) The data analysis period is **1 January 2017 - 31 December 2019**. Since it is known from the internal Pipedrive documentation that in fact the transition to the Mission-based framework was started in April 2018 and finished by the end of 2018, we consider the year 2017 as a pure Scrum-based framework year and the year 2019 as a pure Mission-based framework year.
2) We consider analyzing **only the data related to Pipedrive engineering** and software development. Data of company departments that are not related to engineering (Marketing, Sales, Management, etc.) is not analyzed in this study.
3) For comparative metrics, we analyze Scrum-based framework data within the year 2017 and Mission-based framework data within the year 2019. Since 2018 is a transition period, we decided to compare Scrum-based framework and Mission-based framework in their pure forms by **excluding the year 2018 from the analysis**;

[10] https://www.atlassian.com/software/jira
[11] https://www.bamboohr.com/
[12] https://pandas.pydata.org/
[13] https://jupyter.org/

4) In order to make the data related to both frameworks comparable, we assume that **Scrum-based teams are conceptually equivalent to Mission-based tribes**.

6.2.1 Issues Data

Since Jira issue tracking software was selected as the company-wise tool for managing software development projects and issues before 2017, all Pipedrive development projects and issues (epics, development tasks, bugs) information was retrieved from Jira REST APIs[14] by using the official Python Jira[15] package.

For such purpose, the open-source Jiraextractor[16] tool was used, which required using an internal Pipedrive Jira API username and access token. In order to properly use the Jiraextractor, some code changes were required:

- Adding the issues block size argument the relevant logic. According to the internally configured limitations of Pipedrive Jira API, the number of issues allowed to be retrieved at once (in a single block) was 100, but the tool was processing 1000 issues at once, which caused data errors.
- Adding additional logging and data processing progress information to keep track of data extraction in case of large amounts of data.

The relevant code commits were pushed and merged into the master branch of Jiraextractor. The output dataset of Jiraextractor tool consisted of 2 CSV files: 'issues.csv' (61330 rows, one per issue), including all Jira issues and with their attributes; and 'changelog.csv' (739740 rows for 60997 issues) containing all historical changes of issues (e.g. status, description, assignee, etc.) as well as relevant timestamps.

The most important values of 'changelog.csv' that we used are:

- 'key' (issue key, same as used in 'issues.csv'),
- 'date' (timestamp of issue change),
- 'author' (initiator of the change),
- 'field' (name of the issue field that was changed),
- 'fromString' (previous value of the changed field),
- 'toString' (new value of the changed field).

6.2.2 Missions and Tribes Data

Pipedrive uses the internal **Mission Tracking Tool (MTT)** to see the list of available missions, their stages and apply to specific missions. This tool has its own database containing Pipedrive employees' personal data (name, email, hire date, contract termination date, office location), all tribes and their members, and missions and their participants. The mission and tribes data was retrieved directly from the MTT MySQL database by using SQL queries, the output was saved as 'people_missions.csv' and 'people_tribes.csv' files respectively.

In order to join Jira issues data with missions and tribes information, we had to figure out a property (column) that can be used as an identity for an employee and is represented in both data sources (Jira issues dataset and MTT database). Since the Mission Tracking Tool

[14] https://developer.atlassian.com/server/jira/platform/rest-apis/
[15] https://jira.readthedocs.io/en/master/index.html
[16] https://github.com/ezequielscott/jiraextractor

database does not store Jira usernames or IDs, we decided to join datasets on employees' full names.

For missions data a set of additional constraints and modifications was applied:

- We removed the data related to missions that did not start and finish between 1 January 2019 and 31 December 2019.
- Due to the specific of the MTT personal data storage, it clears full names of employees with terminated contracts, but preserves their original BambooHR ids, so we had to get the original full names by using BambooHR API directly for such cases and replace them in MTT source.
- In addition to the missing full names issue, we found 20 inconsistencies in presented full names (missing middle names, usage of diacritical marks (e.g. 'á', 'š', 'ã', 'é' etc.)) that required unification in both sources.
- Some people could have multiple roles (dataset rows) in the same mission, so we dropped the role column and removed the duplicate rows to have a 1-to-1 issue-mission relation.

For tribes data we applied the following changes:

- We filtered out only tribe memberships that started before 1 January 2020.
- We removed all columns except 'full name', 'tribe name', 'added date' and 'end date'.

6.2.3 Final dataset

By this time, we had 4 data frames: **'issues'** (extracted from 'issues.csv'), '**changelog'** ('changelog.csv'), '**missions'** ('people_missions.csv') and '**tribes'** ('people_tribes.csv'). In order to make a unified data source for further data analysis, we decided to merge (join) issues, missions and tribes data frames into a single data frame.

Pipedrive Jira does not contain specific custom fields for issues to explicitly specify what mission and tribe the issue belongs to. Hence, we did not have any explicit issue-tribe and issue-mission relations in the issues data frame. An alternative way to define such relations requires finding intersections of date period when the issue was assigned to the engineer, dates when this engineer was on missions and dates of tribe membership for this engineer. Due to the lack of assignment date and last status update date fields in the issues data frame, additional data manipulations were required:

- Create a new 'assign_date' column in the 'issues' data frame and fill it with the latest assignee update date from the changelog data frame for every issue.
- Create a new 'last_issue_status_date' column in the 'issues' data frame and fill it with the latest status update date from the changelog data frame for every issue.

After applying these changes, we merged '**issues**', '**mission**' and '**tribes**' data frames into a single '**issues_full**' data frame. We used sqlite3[17] python library to merge 3 data frames using the following SQL join rules:

- For missions: '*issues.assignee_name = missions.full_name AND issues.assign_date >= missions.launch_date AND issues.last_issue_status_date <= missions.end_date*';
- For tribes: '*issues.assignee_name = tribes.full_name AND issues.assign_date >= tribes.added_date AND issues.assign_date <= tribes.end_date*'.

[17] https://docs.python.org/2/library/sqlite3.html

The merged '**issues_full**' contained 61752 rows, which is more than in the original 'issues' data frame (61330). The root cause of it is that in rare cases developers could be assigned to missions with intersecting dates (e.g. a developer worked only at the beginning of one mission, and then moved to another mission before the first one was landed). For these duplicate rows, we had to manually check every affected Jira issue and leave only rows that correspond to correct missions. For such purpose, a new spreadsheet 'issues-mission-duplicates_final.xlsx' with rows to delete was created. This file was later read by JupyterLab python code and used to remove all duplicates (422 rows) from 'issues_full'.

6.3 Data cleaning

As mentioned above, in this research we will analyze only the data related to Pipedrive engineering and software development. In order to find issues related to the development, we had to manually check all Pipedrive Jira projects assigned to issues in the 'issues_full' data frame.

We created a separate spreadsheet 'projects.xlsx' to manually analyze all projects mentioned in 'issues_full' with the following columns:

- project_key - a unique project key defined in Jira;
- project_name - a project name defined in Jira;
- notes - contains text comments about the projects (for internal purpose);
- is_dev - a column to specify whether the project is related to development or not; can have values TRUE or FALSE respectively;
- is_tribes - a column to specify whether the project is related to tribes (Mission-based framework) or not; can have values TRUE (the project is related to Mission-based framework), FALSE (the project is related to Scrum-based framework), or empty value (for non-development projects or development projects that were used in both frameworks);
- team_name - Scrum-based framework team name.

All mentioned columns, except project_key and project_name were, completed manually after we checked the relevant projects one by one.

In total, 193 projects were analyzed, 96 of them are related to engineering (is_dev is TRUE), 17 belong to Scrum-based framework (is_tribes is FALSE) and 69 belong to Mission-based framework (is_tribes is TRUE). 10 projects marked as engineering are ongoing projects that have been in use in both frameworks. All of the Scrum-based framework projects were named after the relevant team's name, so team_name column values are the same as project_name with some minor changes.

After 'projects.xlsx' spreadsheet was finished, we were able to read it in Python and join it with 'issues_full' data frame based on issues' project keys. Thus, 3 new columns were added to 'issues_full': 'team_name', 'is_dev', and 'is_tribes'. The next step was to remove all non-development issues (those with is_dev equal to FALSE). This procedure decreased the amount of 'issues_full' rows from 61330 to 43240.

Since Pipedrive employees used to create a lot of custom fields specific for different projects, 'issues_full' dataset contained 154 custom fields that were named as "customfield_<number>" according to Jira custom fields specifics. To get an understanding of the data inside, all these fields were checked manually. As a result, 139 custom columns

were dropped from 'issues_full', because they were without any data or the data was not useful for the research.

Finally, the prepared 'issues_full' data for further metrics calculation was saved as a CSV file 'issues_ready.csv'.

6.4 Data Pre-processing

Before the actual metrics calculation, several additional manipulations were required:

1. The source data with issues, tribes, teams and missions information was read from 'issues_ready.csv' file into 'source_data' data frame (43240 rows).
2. All date-like fields in 'source_data' data frame were converted into the python DateTime format for easier dates processing.
3. To make the dataset easier for further analysis, the type of all issues with type 'Sub-task' was updated as follows: if the parent task is of type 'Story', the task of sub-tasks were set to 'Task', otherwise, types of sub-tasks were set to parents' types.
4. Since there is no common company-wise understanding of 'Story' Jira issue type usage, we decided to remove all issues with 'Story' type from the 'source_data' data frame (1318 rows).
5. Since the execution of sub-tasks is included in parent issues' execution (and for the sake of data granularity) all parent tasks were also removed from the 'source_data' data frame (513 rows).
6. 85 Mission-based framework issues appeared to be related to non-development tribes, and though were removed from 'source_data' as well.
7. The 'source_data' dataset (containing 41324 rows after previous steps) was further separated into two datasets:
   - '**teams_data**', containing all 'source_data' rows with 'is_tribes' property equal to FALSE (11488 rows);
   - '**tribes_data**', containing all 'source_data' rows with 'is_tribes' property equal to TRUE (23321 rows).

   The remaining 6515 rows from 'source_data' were related to ongoing projects that have been in use in both frameworks. We decided to exclude them from further data analysis because it was not possible to differentiate Scrum-based and Mission-based issues.
8. For the comparative metrics, we analyze Scrum-based framework data within the year 2017 and Mission-based framework data within the year 2019. Thus, two more datasets were created:
   - '**teams_data_cut**', containing all 'teams_data' issues (rows) created before 2018 (6636 rows);
   - '**tribes_data_cut**', containing all 'tribes_data' issues (rows) created after 2018 (17522 rows).

6.5 Descriptive Statistics Results

This section presents the descriptive statistics of the calculated metrics. Plotly Python open-source graphing library[18] was used for the configuration and visualization of the graphs.

[18] https://plotly.com/python/

The detailed procedure of the descriptive metrics data extraction is presented in Appendix I, Table 12.

6.5.1 Number of Employees

The total **number of employees** by departments was retrieved from the Mission Tracking Tool and is shown in Figure 12. The graph shows the number of employees in 8 departments in the date period between 1 January 2017 and 31 December 2019.

Figure 12. Number of employees by departments.

As seen in the graph above, the engineering department is the biggest in the company. In general, the number of engineers grew gradually, starting with 90 engineers and ending with 272.

Although Figure 12 and the relevant data show the precise general picture, we will use the **number of issue assignees** (in the software development projects) for the sake of uniformity of comparative metrics calculation when counting the number of engineers in Scrum-based framework and Mission-based framework. The relevant graph is shown in Figure 13.

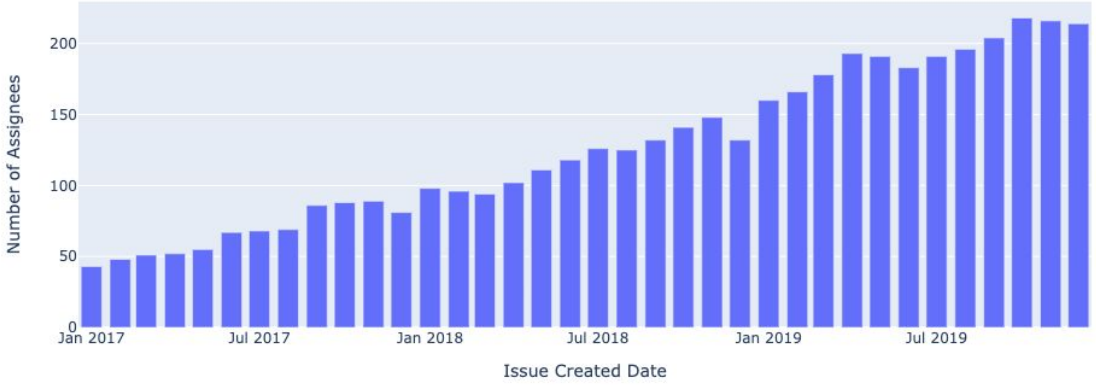

Figure 13. Number of development issues assignees by months.

Although the graph shape of Figure 13 is similar to the graph shape of Figure 12, the number of assignees is lower than the engineering headcount for every month, because 10 ongoing development projects were excluded from the data analysis according to Section 6.4.

As seen in the graph, the number of development issues assignees grew gradually with the engineering headcount growth, starting with 42 assignees and ending with 214. There are several assignee count drops around December 2017, 2018 and 2019 related to the seasonal factors (end of the year and holidays).

6.5.2 Number of Issues

Figure 14 shows 4 pie charts representing the **percentage of issues** aggregated by issue type (task, bug, epic):

1. Issues from all development projects in the research dataset, including the transition year ('teams_data' and 'tribes_data' data frames);
2. Issues from all development projects in the research dataset, excluding the transition year ('teams_data_cut' and 'tribes_data_cut' data frames);
3. Issues from all Scrum-based framework development projects, excluding the transition year ('teams_data_cut');
4. Issues from all Mission-based framework development projects, excluding the transition year ('tribes_data_cut').

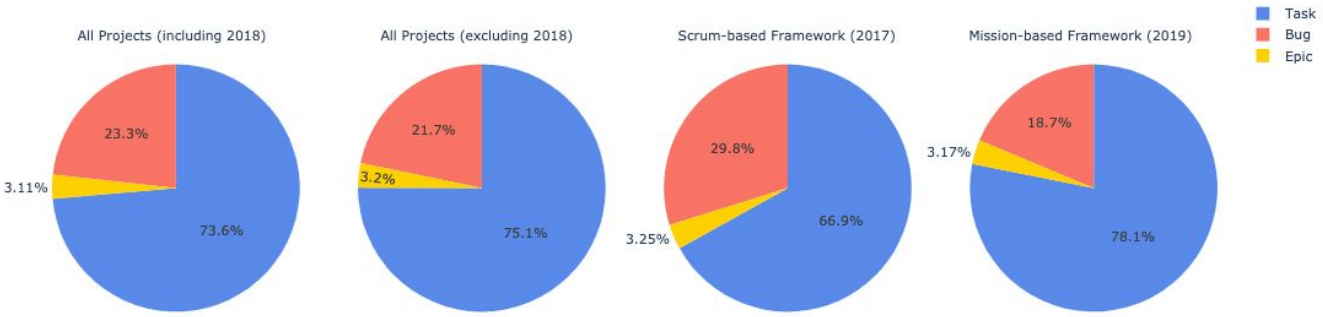

Figure 14. Issue types ratio.

The detailed information about the number and percentage of issues by types according to Figure 14 is represented in Table 4.

Table 4. Number of issues by issue types.

Issue Type	All Data (2017-2019)		Data Excluding Transition Period (2018)					
	All Projects		All Projects		Scrum-based framework (2017)		Mission-based framework (2019)	
	Count	%	Count	%	Count	%	Count	%
Task	25630	73.6	18132	75.1	4441	66.9	13691	78.1
Bug	8097	23.3	5254	21.7	1979	29.8	3275	18.7
Epic	1082	3.11	772	3.2	216	3.25	556	3.17

The stacked bar chart representing the **number of issues created by months** during the whole period of observation (2017 - 2019) is shown in Figure 15.

Figure 15. Total number of issues created.

Although the number of issues created is growing in general, the seasonal decrease cases can be observed at the end of each year. The lowest number (329) was registered in January 2017, the highest value (1943) - in October 2019.

Figure 16 represents the number of created issues from the agile framework perspective.

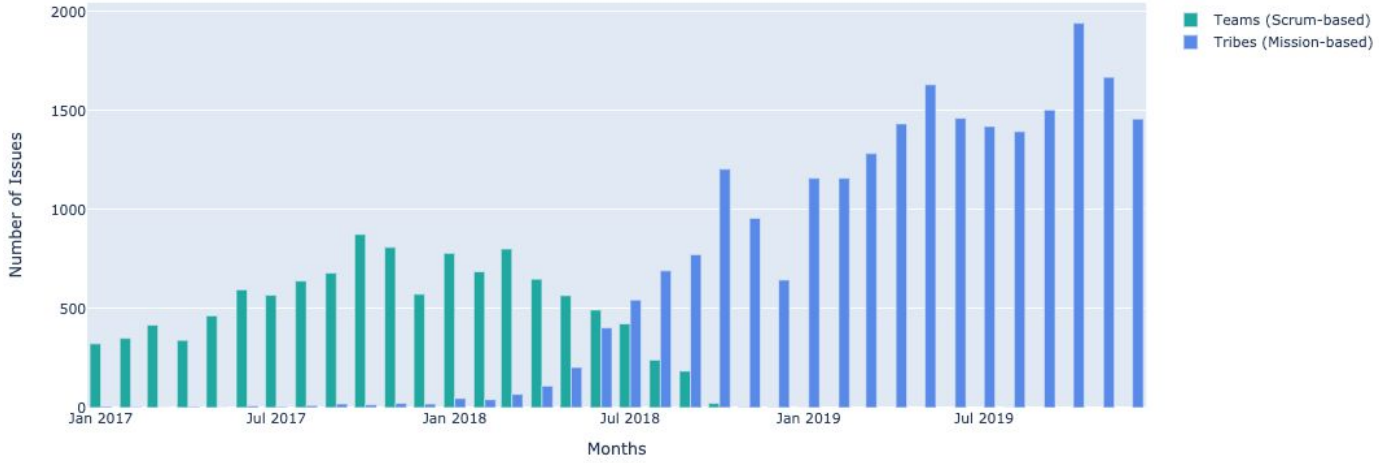

Figure 16. Total number of issues created in Scrum-based and Mission-based projects.

The intersection of bar charts in 2018 shows the transition period from the Scrum-based framework to the Mission-based framework.

6.5.3 Teams/Tribes Size

By **teams/tribes size** we mean the number of monthly employees in Scrum-based framework ('teams_data_cut') and Mission-based framework ('tribes_data_cut'). Although the team/tribe size is not a constant value, they can still be compared by analyzing median, mean and standard deviation measures (see Figure 17).

Figure 17. Monthly number of assignees per team/tribe.

The specific values for the box plots are mentioned in the table below.

Table 5. Monthly number of assignees per team/tribe.

	Teams (Scrum-based framework)	**Tribes (Mission-based framework)**
Minimum	0	1
Maximum	11	45
Median	5	14
Mean	5.28	13.95
Standard Deviation	2.92	8.24

As we can see, the results of tribes (Mission-based framework) contain 3 outliers (41, 44, 45) which show the number for the biggest tribe in Pipedrive that was initially merged from 3 others.

6.6 Comparative Metrics Results

This section presents the results of the comparison of the aforementioned defined metrics for both of the approaches implemented at Pipedrive: Scrum-based and Mission-based frameworks. The detailed procedure of the comparative metrics data extraction is presented in Appendix I, Table 13.

6.6.1 Issue Lead Time and Cycle Time

The **lead time** was measured for all issue types together and separately. Figure 18 shows the lead time calculated for all issue types (tasks, bugs and epics). Figures 19, 20, 21 show the lead time calculated for tasks, bugs and epics only, respectively. Table 6 summarizes the results.

Figure 18. Issues lead time (all issue types).

Figure 19. Issues lead time (tasks).

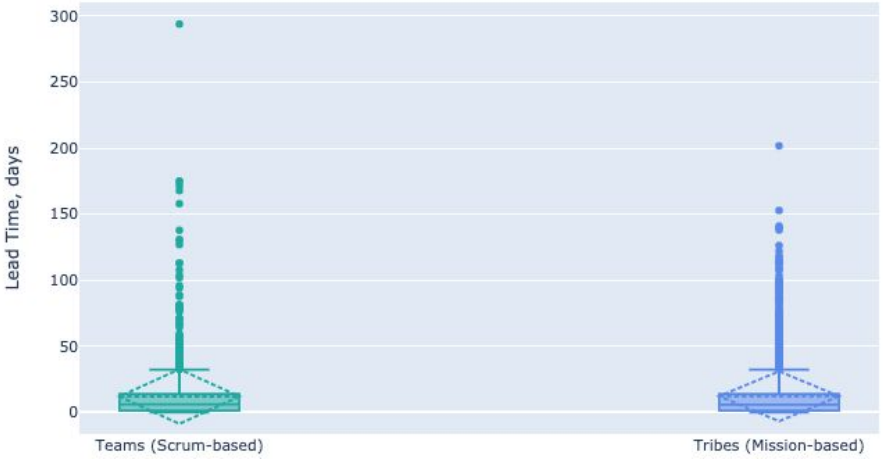

Figure 20. Issues lead time (bugs).

Figure 21. Issues lead time (epics).

Table 6. Issues lead time in both frameworks.

	Teams (Scrum-based framework)				**Tribes (Mission-based framework)**			
	All	**Tasks**	**Bugs**	**Epics**	**All**	**Tasks**	**Bugs**	**Epics**
Min.	≅ 0	≅ 0	≅ 0	0.005	≅ 0	≅ 0	≅ 0	0.93
Max.	300.99	247.23	294.04	300.99	321.96	321.96	201.79	255.02
Median	7.76	10.05	6.00	65.47	7.88	8.03	5.88	65.08
Mean	15.82	18.38	11.98	87.58	16.97	17.22	12.18	72.11
SD	24.91	25.20	20.69	71.62	26.18	26.11	18.91	49.37

The **cycle time** was measured for all issue types together and separately. Figure 22 shows the cycle time calculated for all issue types (tasks, bugs and epics). Figures 23, 24, 25 show the cycle time calculated for tasks, bugs and epics only, respectively. Table 7 represents a summary of the results.

Figure 22. Issues cycle time (all issue types).

Figure 23. Issues cycle time (tasks).

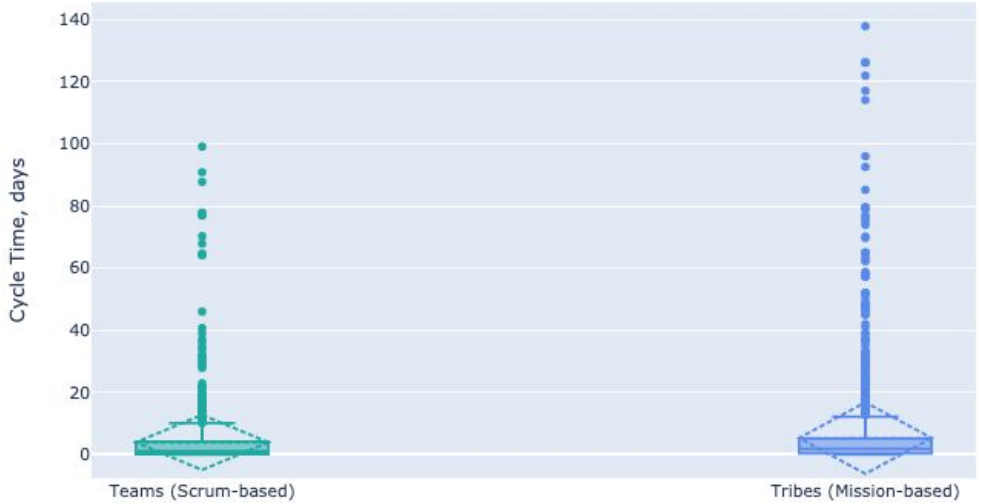

Figure 24. Issues cycle time (bugs).

Figure 25. Issues cycle time (epics).

Table 7. Issues cycle time in both frameworks.

	Teams (Scrum-based framework)				**Tribes (Mission-based framework)**			
	All	**Tasks**	**Bugs**	**Epics**	**All**	**Tasks**	**Bugs**	**Epics**
Min.	≅ 0	≅ 0	≅ 0	3.99	≅ 0	≅ 0	≅ 0	≅ 0
Max.	120.07	120.07	99.11	120.07	316.20	316.20	137.90	156.96
Median	2.18	3.04	1.11	45.85	3.01	3.13	1.90	21.11
Mean	4.50	4.60	3.93	53.00	6.33	6.37	5.31	29.86
SD	8.15	6.28	8.85	36.19	11.57	11.19	11.49	30.23

According to the data extraction procedure (see Appendix I, Table 13), the **cycle time** for the resolved issues without 'back from backlog' rework cases was also calculated for all issue types together and separately. Figure 26 shows the appropriate cycle time calculated for all issue types (tasks, bugs and epics). Figures 27, 28, 29 show the cycle time without rework cases calculated for tasks, bugs and epics only, respectively. Table 8 represents a summary of the results.

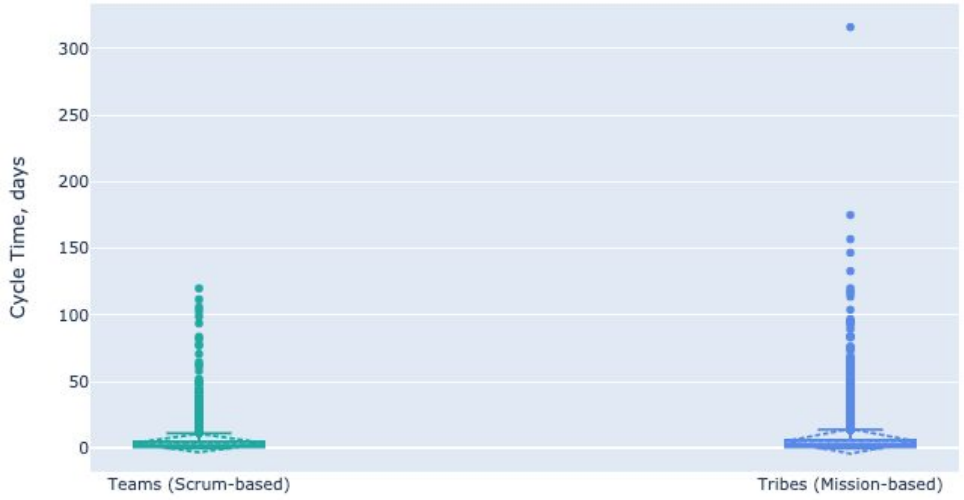

Figure 26. Issues cycle time (all issue types), no ‘back from backlog’ cases.

Figure 27. Issues cycle time (tasks), no ‘back from backlog’ cases.

Figure 28. Issues cycle time (bugs), no ‘back from backlog’ cases.

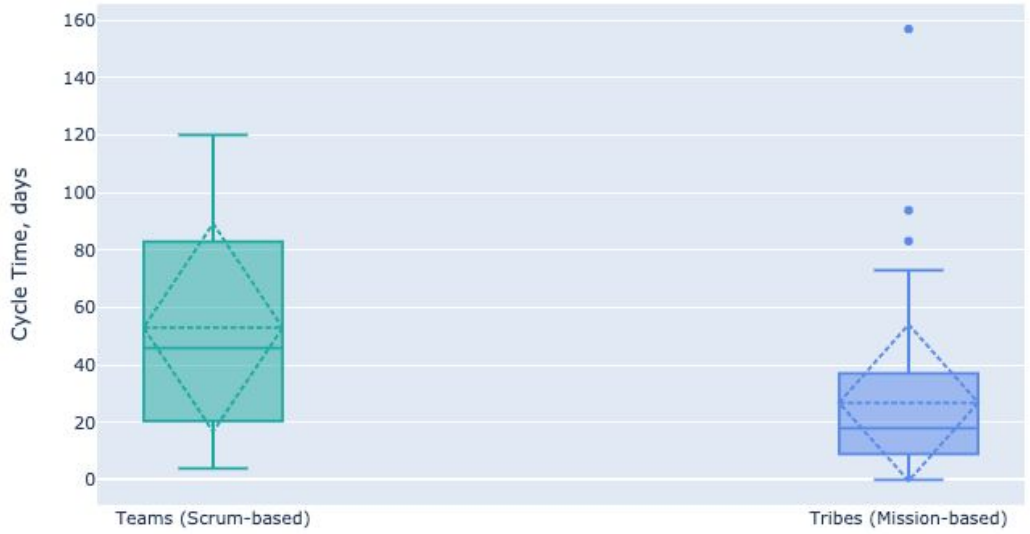

Figure 29. Issues cycle time (epics), no 'back from backlog' cases.

Table 8. Issues cycle time (no 'back from backlog' cases).

	Teams (Scrum-based framework)				**Tribes (Mission-based framework)**			
	All	**Tasks**	**Bugs**	**Epics**	**All**	**Tasks**	**Bugs**	**Epics**
Min.	≅ 0	≅ 0	≅ 0	3.99	≅ 0	≅ 0	≅ 0	≅ 0
Max.	120.07	102.90	77.06	120.07	316.20	316.20	117.11	156.96
Median	2.06	2.97	0.99	45.85	2.80	2.97	1.24	17.93
Mean	3.97	4.26	2.63	52.99	5.32	5.52	3.75	26.85
SD	7.00	5.50	4.86	36.19	9.30	9.25	7.30	27.22

6.6.2 Work in Progress

According to the data extraction procedure (see Appendix I, Table 13), **work in progress** (WIP) is calculated as a mean number of issues an engineer works on during the day (software development and code review activities). Three versions of WIP were measured in this research. Figure 30 shows the WIP calculated for both in-progress activities (development, review). Figure 31 shows the WIP calculated for development activities only. Figure 32 shows the WIP calculated for code review activities only. Table 9 summarizes the results.

Figure 30. Work in progress (total).

Figure 31. Work in progress (development).

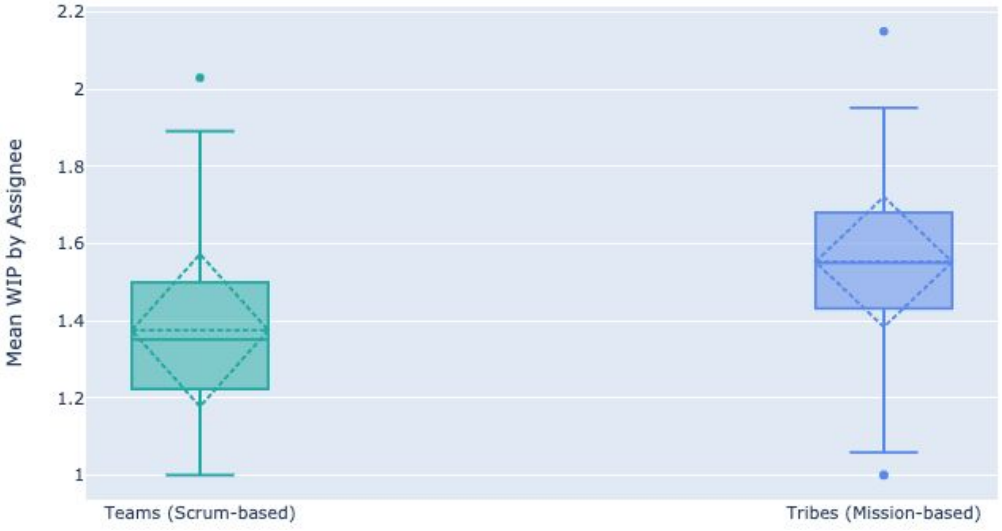

Figure 32. Work in progress (code review).

Table 9. Work in progress.

	Teams (Scrum-based framework)			Tribes (Mission-based framework)		
	Total	**Development**	**Review**	**Total**	**Development**	**Review**
Min.	1.00	1.00	1.00	1.14	1.03	1.00
Max.	2.58	1.85	2.03	2.55	1.79	2.15
Median	1.90	1.35	1.35	2.00	1.46	1.55
Mean	1.81	1.34	1.38	1.94	1.44	1.55
SD	0.37	0.18	0.20	0.29	0.16	0.17

6.6.3 Employee Retention Rate

The **employee retention rate** was measured based on the data retrieved from the internal Mission Tracking Tool database.

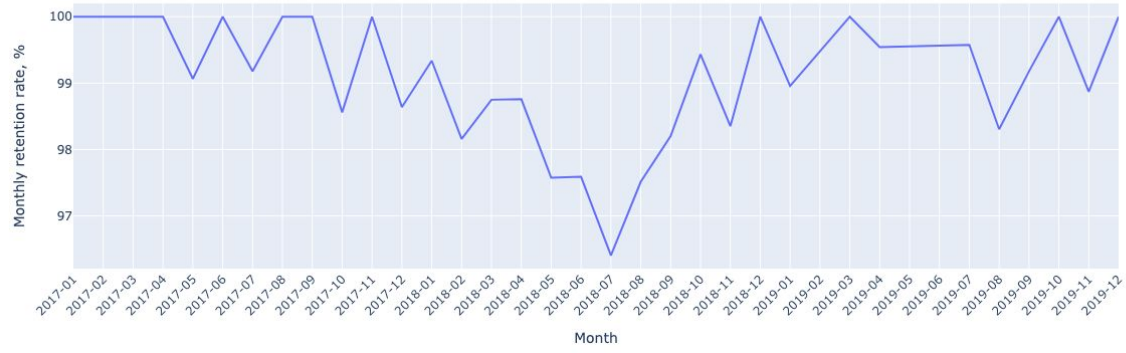

Figure 33. Monthly employee retention rate.

According to the calculated values, the employee retention did not fluctuate much. The monthly retention rate was quite stable, staying high during the whole observation period. The values are between 96.41% and 100% with the minimum registered in July 2018 (96.41%).

6.6.4 Employee Turnover Rate and Team Stability Index

The **turnover rates** and the **inverse team stability index** (TSI^{-1}) were measured on a monthly basis for the Scrum-based framework and Mission-based framework data frames ('teams_data_cut' and 'tribes_data_cur' respectively). Figures 34 and 35 show the monthly turnover rate measurements for newcomers and leavers respectively. Figure 36 shows the monthly inverse team stability index (TSI^{-1}) distribution for both frameworks. Table 10 summarizes the results.

Figure 34. Monthly turnover rate (newcomers).

Figure 35. Monthly turnover rate (leavers).

Figure 36. Monthly inverse team stability index (TSI^{-1}).

Table 10. Monthly turnover rate and TSI^{-1}.

	Teams (Scrum-based framework)			Tribes (Mission-based framework)		
	$T_{newcomers}$	$T_{leavers}$	TSI^{-1}	$T_{newcomers}$	$T_{leavers}$	TSI^{-1}
Min.	0.00	0.00	0.25	0.00	0.00	0.07
Max.	2.00	2.00	1.00	2.00	2.00	1.00
Median	0.00	0.00	0.80	0.12	0.08	0.77
Mean	0.18	0.13	0.76	0.21	0.17	0.70
SD	0.32	0.28	0.23	0.32	0.28	0.25

6.6.5 Defects and Features

We assume all issues belonging to the 'Task' type in the analyzed dataset are related to the development of new functionality. Thus, the **number of features** is equivalent to the number of task issues. Similarly, the **number of defects** is the number of all issues belonging to the 'Bug' type.

The number of created defects and features on a monthly basis is presented in Figure 37.

Figure 37. Number of features (tasks) vs. defects (bugs), monthly.

The same comparison but on a weekly basis is shown below (Figure 38).

Figure 38. Number of features (tasks) vs. defects (bugs), weekly.

Two graphs above are showing the absolute numbers of created issues during the whole observation period (1 January 2017 - 31 December 2019). In order to compare these values with regards to both frameworks, the **ratio of defects** can be introduced as the number of created defects separated by the number of all development issues. These ratios (monthly) are shown in Figure 39.

Figure 39. Percentage of defects compared to all development issues.

As seen from the graph above, the defect ratio for the Mission-based framework (2019) is lower than the defect ratio for the Scrum-based one (2017) throughout the year.

The **net bug score** showing the number of resolved defects divided by the number of created defects is shown in Figure 40.

Figure 40. Monthly net bug score (resolved/created bugs ratio).

As we can see, the net bug score for Scrum-based framework (2017) is similar to Mission-based one (2019): Scrum-based framework had the score more than 100% in 4 months, Framework - in 3 months.

6.6.6 Number of Rework Cases

To make the rework cases data comparable, both the **number of back in development cases** (rework 1) and the **number of back from backlog cases** (rework 2) were normalized according to the number of assignees (engineers). The results are shown below in Figures 41, 42, 43 and summarized in Table 11.

Figure 41. Number of rework cases per employee by months (rework 1).

Figure 42. Number of rework cases per employee by months (rework 2).

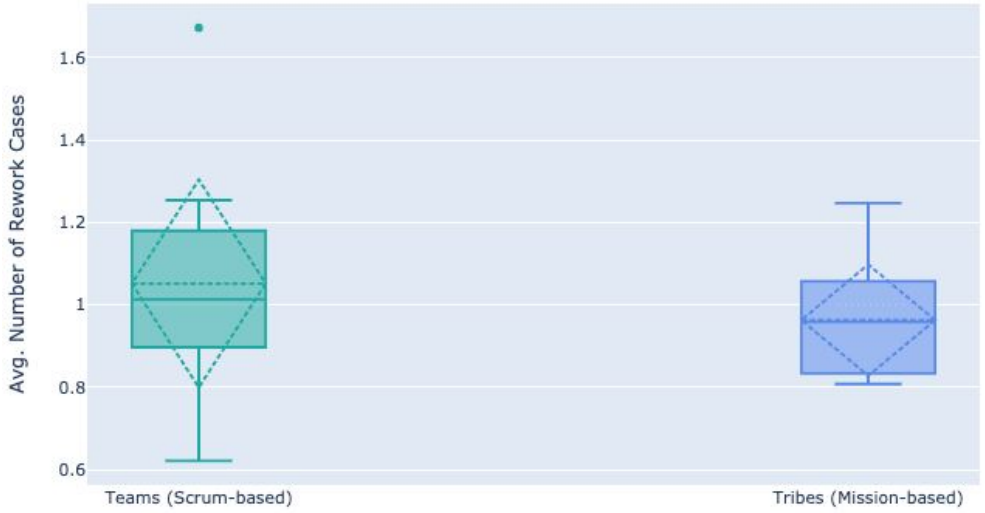

Figure 43. Total number of rework cases per employee by months.

Table 11. Number of rework cases per employee.

	Teams (Scrum-based framework)			**Tribes (Mission-based framework)**		
	Rework 1	**Rework 2**	**Total**	**Rework 1**	**Rework 2**	**Total**
Min.	0.31	0.30	0.62	0.20	0.53	0.81
Max.	0.59	1.08	1.67	0.45	0.80	1.25
Median	0.41	0.60	1.01	0.28	0.70	0.96
Mean	0.42	0.63	1.05	0.28	0.68	0.96
SD	0.08	0.20	0.25	0.07	0.09	0.13

7 Discussion

This section provides the discussion and interpretation of the results retrieved by addressing the research questions. The results for the Scrum-based framework and the Mission-based framework are analyzed to see core differences in used approaches, methods, and outcomes based on the software development process quality, time (productivity) and flexibility. In addition, the limitations of this research are covered.

7.1 RQ1 Discussion

First of all, we will provide a brief summary of the qualitative information retrieved to address RQ1 "*What are the main differences between the Scrum- and Mission-based software development processes*?". According to the gathered data, the Scrum-based framework followed many Scrum principles and concepts (sprints, backlog, sprint planning, sprint reviews, story points, etc.). However, some traditional Scrum concepts (e.g. scrum master) were never adopted, so we cannot describe the Scrum-based framework as an equivalent of pure Scrum. Moreover, the Scrum-based framework used concepts and approaches from many other frameworks and methodologies (Lean software development, Kanban, Scrumban, etc.).

While the usage of mentioned Scrum elements was a company-wide standard in the Scrum-based framework, the Mission-based framework does not prescribe any specific organizational routines for all tribes. Although there are some known recommendations to use plannings, grooming meetings, daily standup meetings and retrospectives, in practice tribes are free to set up their own practices in the launchpad. The analysis of the data retrieved from the surveys and interviews also shows a wider variety of practices and techniques used in the Mission-based framework tribes and missions compared to the Scrum-based framework. This can depend on multiple factors, including the freedom of choice of specific software development practices, methods and approaches on tribes and missions level.

Other key differences that came with Mission-based redesign include (see Table 1):

- the new value delivery model based on longer and not timeboxed mission;
- the updated engineering management structure with tribe engineering managers and rotating mission/launchpad lead roles;
- the ability to deliver complex solutions with cross-tribe missions;
- the updated company-level knowledge sharing with guilds introduced.

Thus, we can say that the Mission-based framework is primarily focused on the organization and management of working units (tribes, mission teams, launchpad, etc.) and the effective value delivery with the mission flows.

During the whole research period (2017-2019), the software product development and delivery were qualitatively supported by coding standards, microservices and collective code ownership, fully automated continuous deployment, the widespread use of unit tests and integration tests. Thus, from the technological perspective, there was no groundbreaking change of coding and deployment practices introduced by the Scrum-based framework.

Nevertheless, some improvement processes, like the migration to the microservice architecture and deployment process enhancement have always been ongoing.

Regarding the problems of Scrum-based approach (see Section 5.3), based on the information retrieved from internal articles, interviews and surveys, we can state the following:

- Problem 1 (Poor focus on product development): According to the Mission-based framework description, tribes have specified units for the delivery of the new features (mission teams), and for the maintenance of already released functionality (launchpads). Thus, we can say that tribes design prevents frequent context switching between new development and maintenance.

- Problem 2 (Mobility): According to the Mission-based framework design and interviews, engineers can work on different product areas by participating in both internal and cross-tribe missions.

Problem 3 (Poor results) is assessed based on the quantitative analysis in Section 7.2.

7.2 RQ2 Discussion

The measured quantitative metrics allow us to compare both frameworks in terms of time (productivity), quality and flexibility of the software product development process to address RQ2 "*What is the impact of the changes on the software development process?*".

Based on the measured engineering headcount (as well as issue assignees) and issues creation statistics, it is clear that the company engineering has been scaling gradually during the whole observation period (1 January 2017 - 31 December 2019).

Although the transition period (2018) is represented on some descriptive metrics figures (employees growth, the growth of created and resolved issues), when comparing both frameworks more attention is given to the values excluding the transition year.

7.2.1 Software development productivity

In this research, several metrics to measure time-related data were introduced: the issue lead time, the issue cycle time, and the work in progress (WIP).

The lead time measurement shows almost the same results for both frameworks: the median values for all issues are 7.76 days and 7.88 days for Scrum-based and Mission-based frameworks respectively. In spite of the similar overall results, in terms of median values, the most significant difference is in tasks (features) lead time (8.03 days for Mission-based framework and 10.05 days for Scrum-based one).

Although there were 2 rounds of issue cycle time measurements (for all issues and for issues without 'back to backlog' cases), the general picture is similar: the overall cycle time for the Scrum-based framework is better (less) than for the Mission-based one. Nonetheless, the Scrum-based framework has noticeably better results only in terms of bugs (defects) resolution. On the other hand, the cycle time for epics in the Mission-based framework is approximately two times less than in Scrum-based one. We assume such differences in epic delivery can be related to the mission flow that allows engineers to deliver big product increments (epics) during comparably short mission timeframes.

Regarding WIP, the values are almost on the same level. The Mission-based framework has slightly worse results, with the overall mean of 2 tasks in progress per engineer (1.9 for

Scrum-based framework). It should be noted that, according to the selected measurement procedure, the measured WIP does not necessarily show the number of tasks being handled in parallel, but the number of tasks that were in progress during the day. This value could not be calculated in a more precise way since the Pipedrive issue management system (Jira) does not track the processing time of the issues.

Based on the data that was calculated in this research, we can assume that the software development productivity stayed on the same level in general after mission-based practices adoption. This means that Pipedrive was able to scale the engineering without compromising productivity.

7.2.2 Software development quality

Regarding quality, several metrics were used: the number of rework cases, the number of defects, the ratio of defects, and the net bug score.

In terms of rework cases mentioned in the research, the total values are similar. The Mission-based framework has less 'back in development after review' cases, but more 'back from backlog' cases than Scrum-based one.

However, the core difference comes if we compare the numbers of defects compared to the overall number of issues. On a global scale (see Figure 14), the total defects ratio registered in the Scrum-based framework (2017) is 29.8%, which is significantly bigger than 18.7% in Mission-based one (2019). Moreover, the monthly defects ratio for the Mission-based framework (2019) stays below the defect ratio for the Scrum-based framework (2017) during the whole observation period (see Figure 39). Also, both frameworks were able to successfully keep a balance of open and resolved defects, which can be confirmed with the net bug score results presented in Figure 40.

Addressing Problem 3 mentioned in Section 5.3, given the results mentioned above and taking into account the engineering growth, we can say that the company is able to produce software of better quality by using Mission-based framework and it is able to focus more on delivering value (new functionality) to the customer than it was during Scrum-based framework.

7.2.3 Software development flexibility

According to the number of assignees per teams and tribes (see Figure 17), we can say that although the mean value of Mission-based tribes size is bigger, the tribes are less uniform in terms of tribe members (the standard deviation is noticeably bigger). This can be interpreted as better tribe size flexibility. As mentioned in Section 7.1, the Mission-based framework does not strictly prescribe the organizational routines and the workflow organization, which also gives more freedom in the internal issue management.

In addition, three metrics were used to measure teams/tribes flexibility: the turnover for newcomers, the turnover for leavers and the team stability index. It appeared that the Mission-based framework has bigger turnover values and smaller stability index (which can be interpreted as better flexibility). However, the difference in flexibility is not as significant as we initially assumed. The reason for this is that turnover and TSI^{-1} are measured only on the team/tribe level. However, the migration of engineers between missions cannot be covered by these metrics, because the equivalent of missions in teams could not be found. Thus, the turnover and TSI^{-1} could be not enough for teams/tribes flexibility, and some other

information about interactions between teams and tribes is needed, which could not be extracted explicitly from the used data sources.

7.3 Limitations

This research covers both the qualitative description of the software development organization in Pipedrive and the quantitative measurement of software development metrics. There exists a thread of bias in the qualitative description of Scrum-based framework due to the lack of documentation, articles and the limited number of survey respondents and interviewees for the time period before Mission-based practices were introduced.

Another problem that made data extraction complicated is the absence of academic research on the topic of Pipedrive software product development. Moreover, Pipedrive did not do any complex internal assessment of the quantitative impact of the software development process redesign, comparing both frameworks. One of the reasons was the lack of data for the Scrum-based framework period (2017 and earlier). Indeed, it was one of the core factors for the selection of metrics in this paper. For instance, the number of deployments, code coverage and deployment stability tracking was initiated only in 2018, so we could not get the relevant information for the Scrum-based framework period. Also, due to the lack of data, the software process quality was not measured directly, but by analyzing the issue types.

Additionally, some agile metrics like velocity and focus factor could not be calculated for both frameworks (story points were used only in several tribes), so they were also not covered in the research. While the time, quality, and flexibility characteristics of the software development process were covered, the cost measurements were not a primary focus of the research and therefore were not analyzed.

The explicit issues, drawbacks and bottlenecks of the Mission-based framework were not identified in the research. Covering this might be a topic of another research, with major emphasis on the qualitative analysis (interviews, surveys, discussions with Pipedrive management and developers).

8 Conclusions

This study covers the topic of the software development process redesign in the growing company Pipedrive, and the primary research goal is to assess the impact of such redesign. In order to assess the impact, qualitative and quantitative analyses were conducted. The qualitative description of the agile frameworks (Scrum-based framework and Mission-based framework) was necessary to elicit the metrics and be able to interpret them.

The qualitative information was retrieved from various sources: documentation, articles, surveys, and interviews. The elicited knowledge allowed us to describe how the software development was organized during the whole analysis period (1 January 2017 - 31 December 2019), what problems Scrum-based framework had, and what changes came with the adoption of customized Mission-based practices. During the qualitative data extraction and analysis, no explicit contradictions in the information retrieved from the used sources were found. The qualitative analysis shows that the Mission-based framework has introduced significant changes with regards to organizational units, value delivery approach, internal agile practices, management, knowledge sharing and collaboration between structural units.

The quantitative analysis was based on the problems identified for the Scrum-based framework. The metrics for the quantitative analysis were derived by using the GQM approach. It appeared that not all initially elicited metrics can be measured due to the data availability and frameworks specifics. The lack of data for the Scrum-based framework period (2017 and earlier) was one of the main restrictions to the data analysis, so some metrics (e.g. the number of deployments, code coverage, deployment stability, etc.) could not be analyzed. The data for metrics calculation was retrieved from various sources (Jira REST APIs, the internal Mission Tracking Tool, internal BambooHR human resources API) and required a lot of data cleaning and processing.

As a result, we were able to calculate both descriptive and comparative metrics covering aspects of the software development process productivity (time), quality and flexibility. It appeared that, according to the dataset used in the research, the software development productivity did not change significantly in the Mission-based framework, meaning that the engineering was able to scale without losing productivity. The quality of the delivered software is noticeably better in the Mission-based framework taking into account the ratio of defects. Finally, the Mission-based tribes can be interpreted as more flexible compared to the Scrum-based teams in terms of workflow organization, development practices choice and the number of tribe members. It is also mentioned in the results of qualitative analysis that tribes are more flexible in terms of resource allocation due to dynamic mission team structures and cross-tribe missions, though it could not be confirmed explicitly with the selected set of metrics.

References

[1] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001). Manifesto for agile software development.

[2] Collier, Ken W. (2011). Agile Analytics: A Value-Driven Approach to Business Intelligence and Data Warehousing. Pearson Education. pp. 121 ff. ISBN 9780321669544. What is a self-organizing team?

[3] Schwaber, K., & Beedle, M. (2002). Agile software development with Scrum (Vol. 1). Upper Saddle River: Prentice Hall.

[4] Sedano, Todd & Ralph, Paul & Péraire, Cécile. (2019). The Product Backlog. 10.1109/ICSE.2019.00036.

[5] Poppendieck, M., & Poppendieck, T. (2003). Lean Software Development: An Agile Toolkit: An Agile Toolkit. Addison-Wesley.

[6] Anderson, D. J. (2010). Kanban: successful evolutionary change for your technology business. Blue Hole Press.

[7] Brenner, R., & Wunder, S. (2015, April). Scaled Agile Framework: Presentation and real world example. In 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (pp. 1-2). IEEE.

[8] Alqudah, M., & Razali, R. (2016). A review of scaling agile methods in large software development. International Journal on Advanced Science, Engineering and Information Technology, 6(6), 828-837

[9] C. Larman and B. Vodde, "Scaling Agile Development," CrossTalk, vol. 9, pp. 8-12, 2013.

[10] Kniberg, H., & Ivarsson, A. (2012). Scaling agile@ spotify. online], UCVOF, ucvox. files. wordpress. com/2012/11/113617905-scaling-Agile-spotify-11. pdf.

[11] Smite, D., Moe, N. B., Levinta, G., & Floryan, M. (2019). Spotify Guilds: How to Succeed With Knowledge Sharing in Large-Scale Agile Organizations. IEEE Software, 36(2), 51-57.

[12] Bass J.M. (2019) Future Trends in Agile at Scale: A Summary of the 7th International Workshop on Large-Scale Agile Development. In: Hoda R. (eds) Agile Processes in Software Engineering and Extreme Programming – Workshops. XP 2019. Lecture Notes in Business Information Processing, vol 364.

[13] Salameh A., Bass J.M. (2019) Spotify Tailoring for Promoting Effectiveness in Cross-Functional Autonomous Squads. In: Hoda R. (eds) Agile Processes in Software Engineering and Extreme Programming – Workshops. XP 2019. Lecture Notes in Business Information Processing, vol 364.

[14] Paasivaara, M., Behm, B., Lassenius, C. et al. Large-scale agile transformation at Ericsson: a case study. Empir Software Eng 23, 2550–2596 (2018).

[15] Razzak M.A., Noll J., Richardson I., Canna C.N., Beecham S. (2017) Transition from Plan Driven to SAFe®: Periodic Team Self-Assessment. In: Felderer M., Méndez Fernández D., Turhan B., Kalinowski M., Sarro F., Winkler D. (eds) Product-Focused Software Process Improvement. PROFES 2017. Lecture Notes in Computer Science, vol 10611.

[16] Brown A.W. (2011) A Case Study in Agile-at-Scale Delivery. In: Sillitti A., Hazzan O., Bache E., Albaladejo X. (eds) Agile Processes in Software Engineering and Extreme Programming. XP 2011. Lecture Notes in Business Information Processing, vol 77. Springer, Berlin, Heidelberg

[17] Buttle, F. (2004). Customer relationship management. Routledge.

[18] Choudhary, V. (2007, January). Software as a service: Implications for investment in software development. In *2007 40th Annual Hawaii International Conference on System Sciences (HICSS'07)* (pp. 209a-209a). IEEE.

[19] Anikin, S. (2015). How we run our growing engineering team at Pipedrive. https://www.pipedrive.com/en/blog/how-we-run-engineering-at-pipedrive

[20] Anikin, S. Scaling Pipedrive Engineering - From Teams to Tribes. https://medium.com/pipedrive-developers/scaling-pipedrive-engineering-from-teams-to-tribes-8f14fd92df8c

[21] Marco Kuhrmann, Philipp Diebold, Jürgen Münch, Paolo Tell, Vahid Garousi, Michael Felderer, Kitija Trektere, Fergal McCaffery, Oliver Linssen, Eckhart Hanser, and Christian R. Prause. 2017. Hybrid software and system development in practice: waterfall, scrum, and beyond. In Proceedings of the 2017 International Conference on Software and System Process (ICSSP 2017). Association for Computing Machinery, New York, NY, USA, 30–39.

[22] Caldiera, V. R. B. G., & Rombach, H. D. (1994). The goal question metric approach. Encyclopedia of software engineering, 528-532.

[23] Newman, S. (2015). Building microservices: designing fine-grained systems. " O'Reilly Media, Inc.".

[24] Kupiainen, E., Mäntylä, M. V., & Itkonen, J. (2015). Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies. Information and Software Technology, 62, 143-163.

[25] M. Sanja and L. Eftimov, “Calculating the cost for employee turnover in the IT industry in Macedonia by using a web calculator,” J. of Human Resource Management, 2016.

[26] A. E. Akgun and G. S. Lynn, “Antecedents and consequences of team ¨ stability on new product development performance,” J. of Eng. and Tech. Management, vol. 19, no. 3-4, pp. 263–286, 2002.

Appendix

I. Detailed Metrics Data Extraction Procedures

Table 12. Data extraction for descriptive metrics.

Metric	Calculation Procedure
Number of all employees	Since Pipedrive Jira issues information does not contain the explicit information of company departments and their members, the source data for this metric was retrieved from the Mission Tracking Tool database, which contains a table with all employees, their departments and start/end contract dates. We used SQL script to transform single rows containing start and end dates for every employee into multiple rows with all individual dates between start and end dates. This transformation was necessary to count the precise number of employees on a daily basis. The output of the used SQL script was stored as 'employees_by_days.csv' file. This file was later read by the python code into the 'employees_by_date' data frame. To finalize the metric information, two more steps were required: 1. The 'employees_by_date' data frame records with dates earlier than 1 January 2017 and later than 31 December 2019 were removed. 2. 'employees_by_date' data frame information was grouped by the department and the date, with aggregation by employees count.
Number of issue assignees	This metric represents the number of development issue assignees. In order to make the results unified with most of the other metrics in the research, this metric was calculated based on 'teams_data' and 'tribes_data' dataset. To extract the data for this metric the following steps were required: 1. 'teams_data' and 'tribes_data' were combined (concatenated) into one data frame; 2. This data frame was grouped by issue created year and month, with aggregation by the unique number of assignees.
Total number of issues	Shows the total number of issues and issue types ratio in Scrum-based framework and Mission-based framework. For overall information, 'teams_data' and 'tribes_data' were merged and grouped by issue types, with aggregation by rows (issues) count.

	For information regarding specific frameworks, ‘teams_data_cut’ and ‘tribes_data_cut’ data frames were separately grouped by issue types, with aggregation by rows (issues) count.
Number of issues by months	The number of issues **created** every month. Was calculated as issue data frame rows grouped by issue creation year and month, with an aggregation by rows(issues) count.
Teams/tribes size	To get an input data for this metric, ‘teams_data_cut’ and ‘tribes_data_cut’ data frames were separately grouped by ‘team_name’ or ‘tribe_name’ columns respectively and by issues creation year and month, with aggregation by the unique number of assignees.

Table 13. Data extraction for comparative metrics.

Metric	**Calculation Procedure**
Lead Time	In terms of Jira issues, by the lead time we mean the time elapsed between the issue creation date/time and date/time when the issue is resolved. The information about Jira tasks resolution was obtained from the following issue data frames (‘teams_data_cut’ and ‘tribes_data_cut’) fields: - ‘status_name’, which is a modification of a standard ‘status’ field containing a string representation of the latest issue status at the moment of the data extraction; - ‘resolution_name’, which is a modification of a standard ‘resolution’ field containing a string representation of the issue resolution type at the moment of the data extraction. It has a value only if the relevant issue is resolved. For this metric, we assume the task is resolved if its ‘status_name’ is in ['Done', 'Closed', 'Live'] list and its ‘resolution_name’ is **not** in ['Canceled', 'Cannot Reproduce', 'Duplicate', 'Won't Do', 'Won't Fix']. In order to calculate the lead time precisely for both frameworks, we had to leave only issues that were originally created and left in the same project. For all resolved issues selected by the criteria above, we calculate the lead time as the difference between 'last_issue_status_date' value (for the resolved issue it is the same as the resolution date) and ‘created’ value, measured in days with floating point.
Cycle Time	In terms of Jira issues, we describe the cycle time as the time elapsed

	between the first moment (date/time) an issue is taken in progress and date/time when the issue is resolved. The earliest transition to 'In Progress' status for every issue was retrieved from the 'changelog' data frame (read from 'changelog.csv') as follows: 1. Select all 'changelog' rows with status changes to 'In Progress' (field == 'status' AND toString == 'In Progress'); 2. Group the retrieved rows by 'key' value and sort by 'date' (ascending). Pick the first row for every group (issue). To make the calculations easier, dates of the earliest transitions to 'In Progress' were joined into 'tribes_data_cut' and 'teams_data_cut' datasets as a new 'in_progress_date' column. Since the cycle time can be calculated only for resolved issues, we filtered out resolved issues by checking if they are originally created and left in the same project and their 'status_name' and 'resolution_name' values correspond to the resolved status (same as described in the lead time calculation). Finally, we calculate the cycle time as the difference between 'last_issue_status_date' value (for the resolved issue it is the same as the resolution date) and 'in_progress_date' value, measured in days with floating point. To calculate this metric, we prepared 2 versions of an input dataset: a) All resolved issues (as mentioned above). b) Resolved issues **without** 'back from backlog' rework cases (the logic to identify such cases is described in **'Number of back from backlog cases**' metric calculation procedure below).

Work in Progress (WIP)	We assume an issue is in progress if it is being developed or reviewed. According to the list of all task statuses mentioned in the 'changelog' data frame, there are 9 unique statuses representing in progress stage (depending on the project they are used): 'In Progress', 'In Progress:Development','In Progress:development', 'Review', 'In review', 'In Review','IN REVIEW', 'Review:InProgress', 'In Review:InProgress'. In order to count the number of tasks that are in progress during the day, these steps were followed: 1. For each of the mentioned in progress statuses, get all in progress periods by capturing all changes **to** in progress status and from in progress status to any other. This can be done by checking 'fromString' and 'toString' values of the 'changelog' data frame rows, sorted by 'key' and 'date' columns (ascending), for rows where 'field' == 'status'. 2. From the periods retrieved during the previous step, select those related to the Scrum-based framework (select all periods with issue keys represented in 'teams_data_cut') and Mission-based framework (select all periods with issue keys represented in 'tribes_data_cut'). 3. Transform single rows containing start and end dates for every in progress period into multiple rows with all individual dates between start and end dates. This transformation is necessary to count the number of issues in progress on a daily basis. As a result, two data frames containing 'key', 'author', 'date', and 'status' columns were created (for Scrum-based and Mission-based frameworks respectively). Consequently, they were grouped by 'date', with aggregation by the mean number of issues in progress for all authors (assignees).
Employee retention rate	For this measure, we calculated the percentage of engineering employees that stayed in a company in a given period. The most precise way of calculating it is to retrieve precise data about employee contract dates. This data was previously obtained from the Mission Tracking Tool database and stored as '**employees_by_days.csv**' file as part of data extraction for the 'number of all employees' metric. This file was read by the python code into the 'employees_by_date' data frame. To get the metric information, the following was required: 1. Filter out employees from the engineering department; 2. For every month we get two sets of employees: the first set contains all assignees that worked on the first day of a month ($\mathbf{E}_{beg}$), the second set contains all assignees that worked on the last day of a month ($\mathbf{E}_{end}$). 3. The employee retention rate was calculated for each month using the formula:

	$$ERR = \frac{|E_{beg} \cap E_{end}|}{|E_{beg}|} * 100 \quad (6)$$ Here, $|E_{beg}|$ means the **cardinality** (number of unique set elements) of the set of all employees worked on the first day of the month, $|E_{beg} \cap E_{end}|$ means the number of employees worked both at the beginning and at the end of the month (intersection of sets).
Turnover rate (newcomers) **Turnover rate (leavers)** **Team Stability Index (TSI)**	According to Table 1, to calculate these 3 metrics, for each team/tribe in the input data frame ('teams_data_cut' and 'tribes_data_cut') the following steps were required: 1. For each month calculate the **number of newcomers** as the difference of a set of assignees in this month and in the previous month. 2. For each month calculate the **number of leavers** as the difference of a set of assignees in the previous month and in this month. 3. Calculate **turnover rate for newcomers** as number of newcomers divided by the sum of assignees in a previous month and in the current month divided by 2 (Eq. 2). 4. Calculate **turnover rate for leavers** as number of leavers divided by a sum of assignees in a previous month and in the current month divided by 2 (Eq. 3). 5. Calculate TSI^{-1} as the initial number of assignees (in the first month of observations) divided by the sum of the number of newcomers, the number of leavers and the initial number of assignees (Eq. 5). As an output we have monthly turnover rate and TSI^{-1} values for every team/tribe.
Number of defects	This metric was calculated as the number of issues with the 'Bug' type created in a specified period of time. For this research, the number of defects was calculated for both 'teams_data_cut' and 'tribes_data_cut' data frames by grouping the rows of 'Bug' type by issue created year and month, with aggregation by issues count.
Number of features	We assume all Jira issues with the type 'Task' in development projects are related to the development of new functionality. Thus, the number of features was calculated for both 'teams_data_cut' and 'tribes_data_cut' data frames by grouping the rows of 'Task' type by issue created year and month, with aggregation by issues count.
Net bug score	Calculated as the number of resolved defects divided by the number of created defects in the same time period (month, week): 1. The number of created defects is calculated by grouping the rows of 'Bug' type by issue created year and month (week), with aggregation by issues count.

	2. The number of resolved defects is calculated by grouping the rows that have ‘Bug’ type and are resolved by ‘last_issue_status_date’ year and month (week), with an aggregation by issues count. For this metric, we assume the task is resolved if its ‘status_name’ is in ['Done', 'Closed', 'Live'] list and its ‘resolution_name’ is **not** in ['Canceled', 'Cannot Reproduce', 'Duplicate', 'Won't Do', 'Won't Fix']. 3. The net bug score is calculated by dividing created defects count (1) by resolved defects count (2) for every month (week).
Number of back in development cases	After analyzing all issue statuses in ‘changelog’ data frame and the most frequent transitions, we assume the following: - Issue is in **development** if its status is in ['In Progress', 'In Progress:Development', 'In Progress:development'] list; - Issue is in **post-development** if its status is in ['Review', 'In review', 'In Deploy', 'Ready for Review', 'Ready for deploy', 'Deploy', 'Development:Done', 'In Review','IN REVIEW', 'Review:InProgress', 'Review:Done', 'In Review', 'In Review:InProgress', 'On hold', 'In Review:Done', 'IN TESTING'] list. This metric was calculated in 3 steps: 1. We retrieved all records from ‘changelog’ data frame, where: ‘field’ value is ‘status’, ‘fromString’ value corresponds to post-development statuses, ‘toString’ value corresponds to development statuses; 2. The retrieved records were further filtered by removing records with keys that do not belong to resolved ‘teams_data_cut’ and ‘tribes_data_cut’ issues; 3. Finally, the filtered data was grouped by year and month of ‘date’ column value, with aggregation by rows count.
Number of back from backlog cases	According to issue statuses in ‘changelog’ data frame and the most frequent transitions, we assume the following: - Issue is in **backlog** if its status is in ['To Do', 'Selected for Development', 'Backlog', 'Discovery', 'ToDo', 'SELECTED FOR DEVELOPMENT', 'Doing this week', 'Selected for work', 'Being Scoped', 'IN SCOPE', 'To do', 'Prioritized', 'TODO', 'IDEAS', 'Ideas', 'Scoping', 'Selected for development', 'backlog'] list; - Issue is **in progress** if its status is in ['In Progress', 'In review', 'In Deploy', 'Ready for Review', 'Ready for deploy', 'Deploy', 'Development:Done', 'In Progress:Development','In Progress:development','In Review','IN REVIEW', 'Review:InProgress', 'Review:Done', 'In progress (biz owner)', 'In Review', 'In Review:InProgress', 'On hold', 'In Review:Done', 'Pending', 'In Dev', 'IN TESTING'] list.

	This metric was calculated in 3 steps: 1. We retrieved all records from ‘changelog’ data frame, where: ‘field’ value is ‘status’, ‘fromString’ value corresponds to in progress statuses, ‘toString’ value corresponds to backlog statuses; 2. The retrieved records were further filtered by removing records with keys that did not belong to resolved ‘teams_data_cut’ and ‘tribes_data_cut’ issues; 3. Finally, the filtered data is grouped year and month of ‘date’ column value, with aggregation by rows count.

II. License

Non-exclusive licence to reproduce thesis and make thesis public

I, Kiryl Lashkevich,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright, “Improving Agile Processes with Customized Mission-based Practices. Case Study”, supervised by Fredrik Payman Milani and Mario Ezequiel Scott.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2

4. I certify that granting the non-exclusive licence does not infringe other persons’ intellectual property rights or rights arising from the personal data protection legislation.

Kiryl Lashkevich

15/05/2020