Aalto University
School of Science
Master's Programme in Information Networks

Markus Tyrkkö

Organizing Software Maintenance in a Small Aviation Software Company:

A Case Study

Master's Thesis
Espoo, July 9, 2019

Supervisor: Professor Casper Lassenius
Advisor: Ville Santaniemi M.Sc. (Tech.)

Aalto University
School of Science
Master's Programme in Information Networks

ABSTRACT OF
MASTER'S THESIS

Author: Markus Tyrkkö

Title:
Organizing Software Maintenance in a Small Aviation Software Company: A Case Study

Date: July 9, 2019 **Pages:** viii + 134

Major: Information Networks **Code:** SCI3047

Supervisor: Professor Casper Lassenius

Advisor: Ville Santaniemi M.Sc. (Tech.)

Software maintenance is usually the longest and the most expensive part of any software system's lifecycle. It generally includes all of the activities that are done after the initial release of the system, although the transition between development and maintenance phases is not clear cut in agile frameworks. The main challenges of software maintenance are generally related to entropy, which tends to increase complexity and degrade quality of the system over time. In small organizations maintenance is also often an ad hoc process, which encourages quick and dirty fixes that effectively increase entropy over time.

This study is a single case study on software maintenance process improvement at QOCO Systems Ltd., a small aviation software company employing 16 persons. The company had previously relied on an ad hoc maintenance process, that was solely dependent on key personnel in direct contact with the customers. This process was replaced with a more structured one before this study, but the new process has not solved the problems of the previous model. The goal of this study is to identify the key challenges on the maintenance process, implement solutions to tackle them and evaluate their effectiveness afterwards. The data collection methods used in this study are interviews, survey, data exports and workshops.

The main challenges of the maintenance process were knowledge sharing and inadequate transition to the more structured process. As a solution to these challenges, three solutions were selected for implementation: rotating triage responsibility combined with pair working, project presentations and a README template. These solutions were evaluated after the implementation phase and the results show that these solutions have indeed managed to improve knowledge sharing and the maintenance process in general, but they also raised discussion about further challenges that are to be addressed in the future.

Keywords: aviation, software maintenance, software process improvement, knowledge sharing

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Informaatioverkostojen maisteriohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä: Markus Tyrkkö

Työn nimi:
Ohjelmistojen ylläpidon järjestäminen pienessä ilmailualan ohjelmistoyrityksessä: tapaustutkimus

Päiväys:	9. heinäkuuta 2019	**Sivumäärä:**	viii + 134
Pääaine:	Informaatioverkostot	**Koodi:**	SCI3047
Valvoja:	Professori Casper Lassenius		
Ohjaaja:	Diplomi-insinööri Ville Santaniemi		

Ylläpito on useimmiten pisin ja kallein osa ohjelmistojen elinkaarta. Se sisältää yleisesti ottaen kaikki ohjelmistoon liittyvät tehtävät sen ensimmäisen julkaisun jälkeen, vaikkakin ketterien kehitysmenetelmien myötä siirtymä kehityksestä ylläpitoon ei ole täysin tarkkaan määriteltävissä. Ylläpidon keskeisimmät haasteet liittyvät ajan myötä kasvavaan monimutkaisuuteen ja järjestelmän yleisesti heikkenevään laatuun muutosten johdosta. Pienissä yrityksissä ylläpito on usein myös epäjärjestelmällistä, mikä kannustaa nopeiden korjausten tekemiseen, mikä puolestaan on omiaan heikentämään ohjelmiston laatua.

Tämä tutkimus on yksittäinen tapaustutkimus ohjelmistojen ylläpitoprosessin parantamisesta pienessä 16 hengen ilmailualan ohjelmistoyrityksessä, QOCO Systems Oy:ssä. Yritys on aiemmin käyttänyt epäjärjestelmällistä ylläpitoprosessia, jossa avainhenkilöt ratkovat ongelmia suoraan asiakkaiden kanssa useiden eri viestintäkanavien välityksellä. Aiempi prosessi kuitenkin korvattiin jo ennen tätä tutkimusta järjestelmällisemmällä prosessilla, joka kuitenkaan ei ole onnistunut ratkaisemaan edellisen mallin ongelmia. Tämän tutkimuksen tavoitteena on tunnistaa ylläpidon keskeisimmät haasteet, toteuttaa niihin ratkaisuita sekä tarkastella ratkaisuiden toimivuutta kokeilujakson jälkeen. Tutkimuksessa kerättiin tietoa haastatteluilla, kyselyllä, työpajoilla sekä analysoimalla olemassa olevaa dataa.

Ylläpitoprosessin keskeisimmät haasteet liittyivät osaamisen jakamiseen sekä vaillinaiseen uuden mallin käyttöönottoon. Ratkaisuna näihin ongelmiin kokeiltiin kolmea ratkaisua: kiertävää triagevastuuta yhdistettynä parityöskentelyyn, projektiesittelyitä sekä README-pohjaa. Ratkaisuita tarkasteltiin kokeilujakson päätteeksi ja tulosten valossa niillä näyttää olleen positiivinen vaikutus tunnistettuihin ongelmiin, mutta tämän lisäksi ne onnistuivat myös herättämään keskustelua muista ongelmakohdista, joihin voidaan paneutua tarkemmin tulevaisuudessa.

Asiasanat:	ilmailu, ohjelmistojen ylläpito, ohjelmistoprosessien parantaminen, tiedon jakaminen
Kieli:	Englanti

Acknowledgements

I would like to thank my supervising professor Casper Lassenius for his helpful feedback during the whole master's thesis process. I am also thankful for my instructor Ville Santaniemi for providing an expert's insight and helping with the practicalities of this thesis. Additionally, I must point out that the process improvement would not have been successful without the positive attitude and active participation of my colleagues at QOCO.

Finally, my greatest thanks belong to Ona, who has been supporting me throughout my entire studies and made this whole process much easier.

Espoo, July 9, 2019

Markus Tyrkkö

Abbreviations and Acronyms

AMOS	AMOS MRO System
AMS	Application management services
CCO	Chief Customer Officer
COO	Chief Operating Officer
CTO	Chief Technology Officer
MRO	Maintenance, repair and operations
PO	Product owner
SLA	Service level agreement
SPI	Sofware process improvement
WIP	Work in progress

Contents

Chapter 1

Introduction

This chapter is an introduction to the topic of this thesis on a general level. It starts by introducing the problem of this thesis and the motivation to solve it. The problem statement also defines the scope and expected outcomes of this thesis together with the research questions guiding the entire process. After this, the overall structure of this thesis is also presented.

1.1 Problem statement

Software maintenance is usually the longest part of software lifecycle since the developed system might be used for years, or even decades, before it is replaced with a new one. Therefore maintenance tasks are usually the most common tasks faced by software engineers on their work (Sommerville, 2011). This might not be initially the case with small startups that are heavily focusing on development, but when they manage to deliver systems to their customers, it is inevitable that they will require some maintenance after the initial delivery. When the development continues and more deliveries are made, the amount of maintenance tasks also increases. Usually the very first model to handle these maintenance tasks is not having a structured model at all. The tasks are solved on an ad hoc basis with the best effort available. The ad hoc processes work well in a small context with occasional maintenance tasks, but when the startup grows and an increasing amount of projects are shifted to maintenance phase, the ad hoc model is no longer scalable and there is a need for a structured maintenance process to handle the incoming requests while maintaining the ability to deliver new features as well.

This has been the background of software maintenance in the case company of this study as well. The ad hoc process has recently been replaced with a more structured one, including a proper incident tracking system and

a defined flow for handling incoming requests. However, it has been noted that the maintenance process does not work exactly as intended as it fails to solve the scalability issue of the previous ad hoc model. The goal of this thesis is to identify the main challenges of the maintenance process, compare different solutions to solve them and implement the most promising ones. The expected outcome of this study is to gain insight about suitability and effectiveness of different solutions on solving the problem. As the scope and time period of a master's thesis is limited, this study does not aim to solve all of the identified problems, but it focuses on the most critical ones. Also the maintenance process after this study is not intended to be perfectly optimized as this study merely acts as the first iteration of continuous evaluation and improvement and provides the tools to develop it further. As the focus is on process improvement, the technical details such as maintainability and testability are left out of the scope of this study.

Research questions

The research problem of this study is divided into three research questions reflecting different research phases:

RQ1: What are the main problems in the maintenance process?

RQ2: What are the most suitable solutions to the identified problems?

RQ3: How well do the selected solutions solve the identified problems?

The first research question aims to define the actual problems in the maintenance process. The starting point of this thesis is a vague feeling that maintenance is not working as intended without any further details about the actual problems in the process. Therefore the first thing to be addressed by this study is to actually model the process in detail and gain further insight about the problems of it. The first question needs to be addressed with extensive current state analysis utilizing several data sources to be able to correctly identify the root causes of experienced issues in the process. As the scope of this thesis is limited, it is also important to prioritize different challenges identified to be able to select the most critical ones to focus on.

After identifying and prioritizing the issues, the second question focuses on finding possible solutions from scientific literature. These suggestions form a basis for the solution proposals, but they might require modifications to be better applicable for the case company's context. Various solution proposals are evaluated in order to find the most suitable ones, which then

will be implemented during the implementation phase in order to address the problems.

The last question then evaluates the efficiency of the solutions implemented. The efficiency of the solutions is defined as the change in selected metrics representing the state of the maintenance process. The initial state is already measured while finding an answer to RQ1 and the final state after this study is measured using the same methods in order to ensure accuracy of the measurement.

1.2 Structure of the thesis

This thesis is divided into ten chapters. After this introductory chapter, relevant research on software maintenance, knowledge sharing and software process improvement is presented in chapter 2. After this the research design and methodology of this study is described in chapter 3. The case company as the context of this study is then introduced in chapter 4, followed by analysis of the initial state in chapter 5. After identifying the points of improvement in the initial state, they are prioritized and proposed solutions to them are explained in chapter 6. The practical implementation details of the selected solutions are then presented in chapter 7 and after the implementation phase, the efficiency of the solutions is evaluated in chapter 8. The outcomes of this thesis are then discussed together with analysis on validity and future research opportunities in chapter 9. Finally chapter 10 summarizes the whole research process including background, used methodology and outcomes.

Chapter 2

Background

This chapter describes the background knowledge related to this thesis. The main focus is on software maintenance and knowledge sharing in agile organizations, but also software process improvement guidelines and frameworks are presented to support change process in the case company. As measuring the change is important in software process improvement, also the most relevant software maintenance metrics are described. Finally to solve the challenges, a set of common solutions is presented, forming a basis of solution proposals applied to the case company's context. The time period of the articles reviewed was typically last ten years for industry specific topics and last 20 years for more general topics.

2.1 Software maintenance

Software maintenance generally includes all of the activities that are done after the software has been delivered (Sommerville, 2011). In modern agile frameworks the transition from active development to maintenance phase might not be as clear cut as presented in the waterfall model, but it can be set to the first successful production deployment that actually provides some business value. Of course there might be new releases requiring active development, but the first version still stays in use in parallel and most likely needs some maintenance every now and then. The maintenance phase is usually the longest phase for any software project as it might last for decades before the system is eventually replaced with a newer one. Taking the long lifespan into account, it is important to consider the maintainability of a system under development as investing more time and money on improving maintainability in the early phase could result in large savings in the future (Sommerville, 2011).

2.1.1 Maintenance task types

Software maintenance activities can be divided into six main categories: *repair tasks*, *adaptation tasks*, *addition tasks*, *preventive tasks*, *perfective tasks* and *user support tasks* (Desharnais and April, 2010)(Sommerville, 2011). These concepts partly overlap as same task could include for example adaptations to a new environment, but also additions based on newly introduced features (Sommerville, 2011). The main categorization still acts as a good guideline for task grouping and is quite widely adopted as a general division. Some researchers such as Desharnais and April (2010) for example combine repair and addition tasks together as *corrective maintenance*, but the idea is still the same as presented by the base concepts.

Repair tasks are generally tasks that are required due to detected errors or faults in design or technical implementation (Sommerville, 2011). They are required for the software to meet its initial requirements and are done reactively, meaning that the error or fault needs to become an issue before it is actually fixed (Desharnais and April, 2010). Generally coding errors are much easier and cheaper to correct in comparison with design errors that could require a major redesign in order to meet the requirements (Sommerville, 2011).

Adaptation tasks are tasks that are related to changing requirements either considering the environment the system is running on, (Sommerville, 2011) or the data it processes (Desharnais and April, 2010). Depending on the severity of the changes these could be fairly simple or require heavy modifications if, for example, a version update on the deployment platform introduces breaking changes that affect the system. In comparison to repair tasks that are done after the fault has already occurred, adaptation tasks are conducted in a preventive manner. This means that they are done to prevent issues from occurring in the future due to changes in the environment or data.

Addition tasks are tasks that are caused by changes in the system requirements due to changed organizational or business requirements (Desharnais and April, 2010)(Sommerville, 2011). These could be combined with adaptation tasks if the changes are related to new features provided by changed environment or data, but could also be entirely separate from them (Sommerville, 2011). Addition tasks are generally much larger and more expensive compared to other types of tasks as they require one or more new development iterations including requirement analysis, planning, implementation, testing and deployment.

Preventive tasks are related to known issues or threats that have not become issues yet, but could potentially cause some harm in the future (Deshar-

nais and April, 2010). They could also be done to improve maintainability and to make it easier to introduce likely changes in the future, thus reducing the cost of future maintenance tasks (Sommerville, 2011). Preventive tasks could include for example refactoring the codebase, improving error handling or adding validations to ensure integrity of the data. The cost of these tasks vary based on their context and the amount of work required, but depending on the case, they could significantly decrease the future maintenance costs.

Perfective tasks are similar to preventive tasks as they are also related to improving quality of the software and preventing future faults (Desharnais and April, 2010). The main difference between these two is that perfective tasks aim to improve the quality of the system by performance improvements or improved monitoring for example. These are not strictly related to issues that could introduce faults in the future, but are related to improving service quality and development experience. Their business value might not be clearly visible during the implementation, but similarly to preventive tasks, they could also decrease the maintenance costs in the future.

User support tasks are tasks that are not directly related to changing the system and therefore do not belong to any of the previously described categories (Desharnais and April, 2010). These tasks are mainly responses to user needs that do not require changes to the system itself, such as providing usage instructions. These tasks could also be done by other than technical personnel, because they do not necessarily require technical knowledge about the implementation of the system, although it would still be beneficial.

The proportion of different task categories varies heavily depending on the context and therefore several distributions have been suggested. They are relatively difficult to compare as the categories might be defined differently, merging several categories together, defining new ones or splitting the main categories into subcategories. However it can be argued that the most common tasks are usually adaptation, repair and addition tasks, since for example a case study by Desharnais and April (2010) suggests that these make up roughly 80 % of all maintenance tasks with user support being the second after them. Sommerville (2011) also argues that addition tasks require significantly more effort than any other category, which draws a clear suggestion about the time usage of maintenance personnel. Perfective and preventive maintenance tasks are not generally common as their direct business value is hard to measure, which makes them difficult to include in the maintenance budget (Desharnais and April, 2010)(Sommerville, 2011).

2.1.2 Software maintenance in agile organizations

It is estimated that there is roughly a 2:1 ratio between maintenance and development tasks in typical software projects (Sommerville, 2011). This ratio has not changed significantly during the last decades and the fact that maintenance is usually the longest and the most expensive part of a software system's lifecycle seems to hold quite well. Therefore the point of view of agile methodologies focusing solely on active development phase, starting from scratch and ending with a delivered release, is odd and unrealistic (Hanssen et al., 2009)(Kajko-Mattsson and Nyfjord, 2009). Although agile methodologies have several well justified benefits and they are generally accepted as a modern way of working (Sommerville, 2011), the lack of maintenance tasks make the models idealistic and could result in inefficient maintenance processes.

Of course it might be the case that the development and maintenance teams are completely separate, which could be because maintenance in general is considered as an activity of lower skill levels (Sommerville, 2011). Therefore the maintenance personnel could be much less experienced and familiar with the system compared to the developers. In this case it is understandable that developers can focus on working with agile methodologies on development phase only, while maintenance team resolves maintenance tickets with their own process, completely separate from development. Even though agile development methodologies result in lower complexity with less errors and generally do not reduce maintainability of the codebase itself (Knippers, 2011), there are still various challenges with them. One of the challenges is lack of documentation since agile methodologies in general do not encourage writing extensive documentation, resulting in knowledge loss during handover between development and maintenance teams (Ito et al., 2016)(Sommerville, 2011)(Stettina and Kroon, 2013). Therefore it is encouraged to have the developers handle the maintenance tasks as well rather than having an entirely separate maintenance team (Sommerville, 2011)(Stettina and Kroon, 2013), which results in combining maintenance activities with agile practices, which is not a well studied subject (Kajko-Mattsson and Nyfjord, 2009).

Fitzgerald and Stol (2014) introduce a definition for "continuous *", bounding together development activities, business operations and process improvements. These concepts bind together process improvements on development and maintenance processes and it is beneficial for the management to understand the close relationship of these activities, no matter whether there are separate maintenance and development teams or not. By improving both processes, or joining them together, the transition between active

development and maintenance phase can be smoother, resulting in a more efficient maintenance process, which will eventually benefit the development process as well.

2.1.3 Main challenges

Maintenance of a software system is a challenging task as the system quality tends to degenerate over time due to an increasing amount of changes and knowledge loss. This phenomenon is called software entropy and just like its thermodynamical paragon, it also increases naturally over time if no actions are taken to reduce its effects. Entropy is also evident in increasing complexity and interdependencies between systems, which makes changes even more complicated as modifications to one system might cause a need for modifications in several other systems as well (Hanssen et al., 2009). Increasing amount of interdependencies and modifications cascading from one system to another could also cause team separation to not reflect system separation. This increases the risk of conflicts between streams as different teams are making changes to same parts of the system. There are several actions that could be taken to reduce entropy and chaos. One of the most effective of them is refactoring, which can drastically reduce the complexity of the codebase, improving maintainability and reducing future costs. In addition to refactoring, Hanssen et al. (2009) also suggest semi automatic code inspections to detect code smells already during the development phase and unit testing to ensure that changes do not break the requirements.

Richardson and von Wangenheim (2007) also suggest that rapidly changing technologies are causing increasing entropy, knowledge loss and general difficulty in maintenance of older projects. Even though it could be argued that advancements in technology could also solve the problems in software engineering practices, Demir (2009) argues that this is not the case, with the most challenging parts of software lifecycle being in estimation and requirements management. Rapidly changing technologies together with the challenges in software engineering practices yield software projects that are difficult to maintain due to lack of knowledge about technologies used or requirement definitions. These challenges are closely related to lack of formal documentation, which is problematic for agile methodologies that generally rely on the codebase itself as documentation (Ito et al., 2016)(Sommerville, 2011)(Stettina and Kroon, 2013). Relying on codebase itself as documentation is risky as it might be insufficient in some cases due to lack of refactoring for example (Hanssen et al., 2009)(Tiarks, 2011). In addition to the code itself, developers also rely on tacit knowledge, which might or might not be available, but also comments in the codebase, which also divides people with

different opinions about whether or not they are actually a useful source of documentation (Tiarks, 2011).

The major problem related to software maintenance processes especially in small organizations of 10-20 people is the fact that most of the suggested maintenance processes are targeted for medium to large size organizations and are therefore not suitable for small companies (Basri and O'Connor, 2010a)(Hasan and Chakraborty, 2011)(Sánchez-Gordón and O'Connor, 2016). These processes usually require an amount of resources that small organizations simply do not have, which has been identified as the main factor preventing small organizations from adopting standardized maintenance processes (Basri and O'Connor, 2010a). Lack of resources generally leads to ad hoc processes, where maintenance is done on a "best effort" basis without a strict structure (Hasan and Chakraborty, 2011). This is also a benefit of small organizations as they can be more agile and provide more customized services with better customer experience for their customers and therefore small organizations should not try to replicate larger ones directly (Aranda, 2010). The significant drawback of these ad hoc processes is their dependability on certain key persons that have usually a long history with the customers and have gained their trust. This relationship becomes problematic as the company grows and most of the maintenance tasks are still handled by these key persons, which prevents knowledge sharing and scaling (Hasan and Chakraborty, 2011). It has been shown that organizational dynamics usually change around 10-20 employees and 50-100 employees at least and it is important to recognize these changes in dynamics to be able to readjust processes, such as maintenance, that do not scale that well any more. This flow of causalities is presented in figure 2.1, which is an adapted version of a model proposed by Hasan and Chakraborty (2011, p. 5).

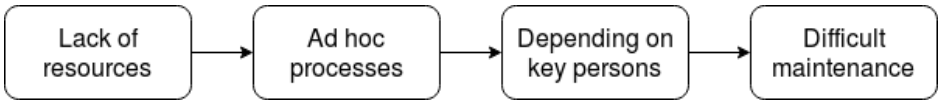

Figure 2.1: Reason for challenges in software maintenance in small companies. Adapted from Hasan and Chakraborty (2011, p. 5)

Many of the heavier maintenance processes also emphasize quality, which might not be the most important thing for rapidly growing small organizations (Sánchez-Gordón and O'Connor, 2016). This poses a challenge on implementing standardized maintenance processes, since it would also require improving quality standards for example by refactoring. It has been shown that the main focus of small companies is usually not in improving quality by

refactoring as there rarely is enough resources for that (Kajko-Mattsson and Nyfjord, 2009). Therefore the small companies need more tailored models of maintenance, some of which are proposed, but not extensively researched (Hanssen et al., 2009).

Another typical challenge for small companies is the lack of explicit portfolio management, which results in multitasking with different projects in parallel (Vähäniitty et al., 2010). This encourages ad hoc processes and can be visible as "fire fighting" while trying to resolve issues with entirely different projects concurrently. Lack of strategic focus and portfolio management could also lead to an overload of projects, which is not beneficial when considering quality and process aspects as the time pressure for ad hoc solutions and poor maintainability grows. Additional effect of multitasking is assigning multiple roles for employees, meaning that the same employee could act as a business analyst, developer, tester and sales representative at the same time, which reduces the productivity and efficiency of all processes that are operated in parallel (Hasan and Chakraborty, 2011).

2.2 Knowledge sharing in agile software development

There are mainly two types of knowledge: *explicit* and *tacit*. Explicit knowledge refers to knowledge that can be easily verbalized, documented and shared with other individuals, whereas tacit knowledge is the exact opposite of that, meaning knowledge that can not be easily described or shared via explicit documents (Ryan and O'Connor, 2013). Tacit knowledge is gained through experience and learning by doing and it can not be learned from a book. In software development, sharing both is important, but tacit knowledge has a much more significant effect on software development skills both on team and individual level and therefore sharing tacit knowledge should be focused over explicit (Ersoy and Mahdy, 2015)(Ryan and O'Connor, 2013). This is especially important in agile software development as there is usually no explicit documentation available and development methods rely on sharing tacit knowledge between team members (Ryan and O'Connor, 2013). Sharing tacit knowledge has also proven effects on performance and innovation capabilities on team level (Wang and Wang, 2012) and these effects are much more significant on smaller teams compared to larger ones as smaller teams tend to work closer together (Ersoy and Mahdy, 2015).

In addition to performance improvements and encouraging innovation,

sharing individual tacit knowledge to other team members reduces risks related to individuals as there is always a possibility that they could have an accident, which forces them on a sick leave or they could just leave the company taking all of the acquired tacit knowledge with them (Ersoy and Mahdy, 2015). The more tacit knowledge the person has, the more serious the risk becomes and therefore it should be addressed accordingly. To reflect the phenomenon of shared knowledge and a "collective mind", Ryan and O'Connor (2009) extend individual tacit knowledge to team level by introducing the concept of team tacit knowledge, representing the knowledge that the group has. This is more than the sum of individual knowledge areas because individuals working as a team can share and create new knowledge much easier and faster than the same individuals working alone.

The theoretical model for acquiring team level tacit knowledge by utilizing transactive memory system (TMS) is presented in figure 2.2, which is originally presented by Ryan and O'Connor (2013). TMS was first defined by Wegner (1987) as a shared storage of internal, external and transactive memories, meaning that each individual in a group has access to certain information through the expertise of each other. The development of transactive memory is directly related to the level of team tacit knowledge and therefore highly developed TMS improves team performance (Ryan and O'Connor, 2013).

The starting point for the acquisition cycle is the initial team tacit knowledge previously acquired by working together and sharing experiences. This team level knowledge forms a base for individuals to construct their own thoughts and learn more, which then leads to a creation of individual tacit knowledge. The most important part of the cycle is sharing tacit knowledge through social interaction and working together, which then eventually leads to an acquisition of new team tacit knowledge through shared experiences. Ryan and O'Connor (2013) emphasize the quality of social interaction as tacit knowledge is best shared in informal face-to-face interactions between individuals. These interactions also help to develop the TMS for the team, which in turn leads to increased team tacit knowledge and improved team performance. Ryan and O'Connor (2013) also recognize that there are several other human factors in addition to social interactions and TMS that affect the acquisition cycle. These factors include for example trust, leadership and team cohesion that also have a significant effect on team tacit knowledge creation and the development of TMS and therefore they should not be neglected.

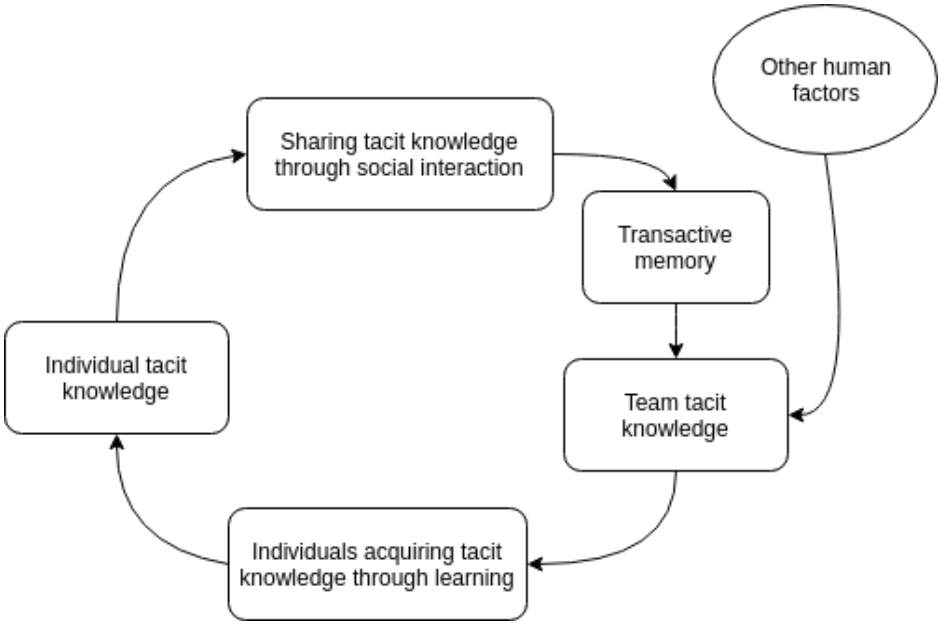

Figure 2.2: Acquiring team tacit knowledge with transactive memory. Adapted from Ryan and O'Connor (2013, p. 8)

2.2.1 Main challenges

There are various challenges in knowledge sharing, some on general level and some specific to software development. Ersoy and Mahdy (2015) group the challenges into three main categories: *sociological*, *documentation* and *implementation challenges*.

Sociological challenges

The sociological challenges in knowledge sharing are related to human factors, not technical implementations or processes. One of the most significant sociological challenges in knowledge sharing is the notion of knowledge as a property and the importance of its ownership. This is caused by considering knowledge as power and rewarding individuals for having it, rather than sharing it, by higher salary for example. This encourages individuals to hoard knowledge as they do not want to risk losing their position by sharing what they know. To prevent this, managers should start to recognize individuals for the knowledge they share, not the knowledge they have and also give credit for knowledge products they create (Dalkir, 2013). This is the key to create a better culture of knowledge sharing in an organization, which

improves the performance and capability of the organization as a whole, but also requires time to build.

Another sociological issue is the developer turnover rate as employees leaving the company will take the knowledge with them (Ersoy and Mahdy, 2015). There might be some legacy projects that no one else has touched, which results in an orphaned code that no one is familiar with. This is especially problematic with strong code ownership as individual employees become the key contributors in certain projects and do not want to share the ownership due to lack of trust for example (Ersoy and Mahdy, 2015). Lack of trust can be present in both ways as those with the knowledge do not want to share it due to lack of trust on other persons, or those needing the knowledge might not trust the knowledge provided due to lack of trust on expertise of the one with the knowledge (Dalkir, 2013). This is definitely a challenge of organizational culture and needs to be taken seriously as it poses threats on team dynamics in general. In addition to employees leaving the company, knowledge also tends to get forgotten over time if it is not actively used (Basri and O'Connor, 2011). This occurs frequently and can not be entirely prevented, but sharing knowledge from individual level to the team level certainly helps to memorize the knowledge if it is needed again and thus reduces knowledge loss.

Other considerations about sociological issues include practical issues such as working in different locations or during different times. This decreases the amount and quality of social interactions which are essential in knowledge sharing. Additionally constant occupational stress reduces knowledge sharing drastically. It could be caused by several different factors including organizational culture, team dynamics and workload for example. It is important to notice the signs of stress as early as possible since it has a significant effect on knowledge sharing and team morale in general causing lower performance and introducing new problems if not addressed properly. (Ersoy and Mahdy, 2015)

Documentation challenges

Documentation challenges in knowledge sharing are related to creating, using and maintaining explicit knowledge with documentation (Ersoy and Mahdy, 2015). Agile methodologies generally do not emphasize extensive documentation and therefore documents used in agile development are generally informal, especially in the case of small companies having less than 20 employees (Basri and O'Connor, 2011). However as Ersoy and Mahdy (2015) argue, transferring explicit knowledge is difficult without proper documentation, which is one of the reasons agile methodologies rely mostly on sharing tacit

knowledge. Stettina and Kroon (2013) also argue that a formal handover between active development and maintenance phases would help preventing knowledge loss, which also supports the argument of creating proper documentation. The need for documentation could be decreased by introducing the maintenance team to the project already during the active development phase, or even having the developers handle the maintenance phase as well. However this does not remove the challenge of introducing new employees to the project either during the development or maintenance phase.

The most common issue with documentation after missing it entirely, is keeping relevant artifacts up to date (Ersoy and Mahdy, 2015)(Stettina and Kroon, 2013). This is problematic as outdated information could lead to wasting time while working on already resolved issues or trying to get newer version of a project to work with outdated documentation. Outdated documentation also lowers the value of updated documentation as developers might have concerns about the validity of it as well due to previous experiences of using outdated documents. This is one of the reasons agile methodologies emphasize the code itself as a documentation with clarifying comments if necessary (Stettina and Kroon, 2013). By using the code itself as a documentation, it is guaranteed to be always up to date and usually properly version controlled, which ensures that the person is always looking at the documentation that matches the executed code. Of course there is a risk that comments will be outdated after changes, which highlights the importance of readable code instead of relying on clarifying comments. Another popular format for documentation is using a wiki, which could be beneficial for larger projects. However the practices should be carefully evaluated as writing documentation for no one and without a purpose is a waste of resources and generally lowers the team morale (Stettina and Kroon, 2013).

Implementation challenges

Implementation challenges in knowledge sharing are related to the practical implementation of different knowledge sharing techniques and practices. One of the proposed solutions to implement knowledge sharing in agile development is pair programming, which is further described in section 2.5.1. Pair programming has several benefits, most importantly encouraging knowledge sharing and improving the quality of the code being developed. However there are several practical issues in implementing pair programming including pairing, lack of resources and motivation towards it. One of the reasons for lacking motivation for pair programming is the skill level gap between the individuals participating on it. If, for example, junior team members are paired with the most senior ones, the seniors might find less satisfaction

in their job due to considering pair programming tasks simple and resource intensive. (Ersoy and Mahdy, 2015) On the other hand, pairing junior team members with each other does not encourage knowledge sharing as much as junior-senior pairs, which partly defeats the knowledge sharing purpose of pair programming (Sun, 2011). Additionally pair programming introduces scheduling issues as it requires more resources and increases the overall time consumed to complete the task at hand (Ersoy and Mahdy, 2015).

Another issue related to the practicalities of knowledge sharing is rapid pace of technological development (Ersoy and Mahdy, 2015). Rapid development of the technologies requires more time spent while catching up with the latest trends and renders older knowledge outdated faster. Therefore the need for a constant effort to keep up with the technologies could cause sociological issues such as stress and lack of trust on knowledge credibility, but also affect team dynamics as there might be differing opinions about the latest development trends in used technologies.

Knowledge sharing barriers

Literature sources generally refer to challenges in knowledge sharing as "barriers" since their presence prevents effective knowledge sharing. There are several barriers suggested in various sources and next the important barriers from software development point of view are listed and briefly explained. Recognizing these barriers helps to categorize and solve knowledge sharing challenges as there are various studies describing the best practices to address each of them.

Lack of formal documentation means missing official documents describing a software product or service. Formal documents are usually considered quite heavy and are more common in large corporations rather than small agile organizations. Lacking proper formal documents could prevent knowledge sharing especially in larger systems, depending on the context. (Ghobadi and Mathiassen, 2016)

Lack of informal documentation means missing unofficial documents. In comparison to a formal documentation, informal documents can be of free format and varying levels of quality. Informal documentation is more common in agile development as it is easier to create and maintain. If formal documents are missing or outdated, informal documents should fill their purpose and lacking them therefore prevents efficient knowledge sharing. (Ghobadi and Mathiassen, 2016)

Lack of trust, meaning lacking trust towards other employees and their knowledge. As already described in the previous sections, lack of trust is a significant issue in knowledge sharing effectively preventing it totally and

encouraging an unhealthy organizational culture. It is a serious issue and should be addressed with proper caution. (Ersoy and Mahdy, 2015)

Lack of comments in code. As code itself generally acts as a documentation in agile methodologies, the importance of its clarity and quality increases. In some cases the code can be further explained with additional comments and missing them in unclear parts of the codebase might lead to difficulties in knowledge sharing. (Stettina and Kroon, 2013)

Messy and complex code. In addition to comments, the code itself acts as a documentation and it generally should be understandable without clarifying comments. Having a messy and complex codebase becomes problematic over time as natural knowledge loss occurs and the codebase becomes legacy that no one knows in detail anymore. Reducing complexity improves future maintainability and decreases the effect of forgetting unused knowledge over time. (Stettina and Kroon, 2013)

Lack of informal communication is closely related to the acquisition cycle for tacit knowledge presented in figure 2.2. As tacit knowledge is best shared through informal social interactions, lacking them effectively breaks the cycle of acquiring team tacit knowledge eventually slowing the learning process of individuals as well. (Ghobadi and Mathiassen, 2016)

Knowledge hoarding means refusing to share tacit knowledge with other persons due to its value or lack of trust. Hoarding is most likely a symptom of issues in the organizational culture and reward system as the reasons to hoard knowledge generally reflect the organizational dynamics. Therefore it can not be solved by forcing individuals to share knowledge, but it needs to be addressed on an organizational level. (Dalkir, 2013)

Difference in experience levels could become an issue if it is so large that different persons do not generally have the same tools to discuss about specific topics. This could cause frustration on more experienced employees when they need to explain the most simple concepts of some topic to less experienced colleagues, who on the other hand will not benefit much from the discussion if they are lacking the basic knowledge on the subject and can not follow the explanation. This situation makes knowledge sharing difficult and time consuming, which could affect other barriers by decreasing the amount of informal communication for example. (Ghobadi and Mathiassen, 2016)

Tight schedule can prevent the usage of pair programming for example as it generally requires more time than solo programming. Tight schedules could also cause stress and prevent knowledge sharing since there basically is no time available for knowledge sharing activities such as informal discussions. Tight scheduling might reflect the problems on managerial side, which is why its root causes need to be identified for effective resolution. (Ghobadi and Mathiassen, 2016)

Lack of motivation could also be a sign of managerial problems, but could also occur due to reasons completely outside of work environment. Either way, lack of motivation towards working in general or knowledge sharing in particular affects the team dynamics negatively and could affect the motivation of team members as well. It is important to identify the root cause for lacking motivation before it spreads further to the organization. (Ghobadi and Mathiassen, 2016)

Used communication tools could be a preventive factor in knowledge sharing, especially if they are the main way of communication due to distributed locations for example. One important factor is the usability of the communication tools as it greatly affects the quality of social interaction, but also the communication practices that are supported by those tools. It is important for the managers to analyze whether the tools actually support or prevent communication and to make changes if considered necessary. (Ghobadi and Mathiassen, 2016)

Multitasking means working on different projects or tasks in parallel so that one needs to constantly switch context from one to another. These context switches decrease efficiency and quality of work as each time the context is switched, some time will be lost while readjusting the mindset to work on a new context. Time spent to understand and solve the tasks is reduced and therefore learning and knowledge sharing is disturbed by constantly changing tasks, which could also increase stress. (Ghobadi and Mathiassen, 2016)

Complex domain, such as aviation industry, could also prevent knowledge sharing as its complexity requires much more learning compared to simpler ones. It also effectively increases the knowledge gap as novices will have to learn a lot to be even able to discuss with the experts using same terms. Complexity could also mean more tacit knowledge and make certain aspects more difficult to verbalize so that they could be understood without a detailed knowledge about the related concepts. Working in a complex domain could also lead to knowledge hoarding as domain specific knowledge becomes an increasingly valuable asset when complexity increases. (Ghobadi and Mathiassen, 2016)

Working in different locations naturally affects the communication between team members by reducing the amount of informal face-to-face interactions. Even though remote working is popular in the software industry, it certainly affects the team dynamics and there should be clear rules about its practicalities to minimize the negative effects. Special concern should also be on the used communication tools so that they will effectively support remote working and lower the barrier of knowledge sharing. (Ghobadi and Mathiassen, 2016)

Difference in age is related to a gap in skill levels, but rather than being related to skills and terminology, it is more concerned about the social constructs during interaction. The challenge is usually even more significant if a younger employee would be the expert and an older one the novice, which turns the social hierarchy upside down and might be threatening or depressing to some persons. (Riege, 2005)

Individual communication skills are essential on daily discussions, both formal and informal, as weak communication skills generally lowers the quality of interactions and could even lead to misunderstandings. Improvement of these skills is the responsibility of each person individually, but organization could help them to improve by encouraging and giving constructive feedback. (Riege, 2005)

Strong code ownership is related to knowledge hoarding as individuals with strong ownership on a certain codebase are unwilling to share their knowledge and ownership of it. This can become an issue if these individuals leave the company or are otherwise unavailable to answer the questions about the codebase. There is no shared knowledge about the codebase and gathering it without the key persons might be difficult. In the worst cases, the project becomes legacy and requires heavy re-engineering before any further updates can be made to it. (Riege, 2005)

Physical work environment can also prevent knowledge sharing, since if the workspace does not support team activities or informal discussions, the amount and quality of interactions will decrease. This is an important aspect to keep in mind while considering changes at the office. Managers could support knowledge sharing by providing enough group working rooms with sufficient equipment such as screens and whiteboards to support co-working. (Riege, 2005)

Organizational culture is one of the key aspects in knowledge sharing since if it does not support and encourage knowledge sharing, there will be significant performance and team working issues. Organizational culture itself is hard to measure and define, but there are plenty of other barriers that could be present due to issues in organizational culture, which makes it an influential aspect that should be considered and evaluated frequently. (Riege, 2005)

Strong organizational structure or hierarchy could also prevent organization wide knowledge sharing and encourage individual work. If there are for example strict rules about communication between different teams, the amount of interactions between the teams most likely decreases. Also if the organization relies too heavily on top-down management, there will not be knowledge co-creation on team level, which could improve performance and innovation capabilities. (Riege, 2005)

2.3 Software process improvement

Processes are a crucial part of software development and their efficiency can make a difference between success and failure. It has also been recognized that software process improvement (SPI) is one of the key challenges for small and medium sized software companies and therefore it should be taken seriously (Sulayman and Mendes, 2009). According to the lean ideology the processes should be under continuous improvement, which makes them similar to agile software development with small incremental iterations to react to changing requirements. Osterweil (2011) argues that software processes can be treated exactly the same as software itself and therefore SPI can be addressed with similar methodologies as software development, including iterations and continuous improvements.

There are various frameworks to implement these improvements, most notably Capability Maturity Model Integration (CMMI), which is used to evaluate the maturity of a software organization. CMMI includes 22 processes to measure and improve, which can be evaluated either with four capability levels in a continuous representation (*incomplete*, *performed*, *managed* and *defined*) or with five maturity levels in a staged representation (*initial*, *managed*, *defined*, *quantitatively managed* and *optimizing*) (CMMI Product Team, 2010). This maturity model can be used to evaluate internal processes or a subcontractor for example. However, it is not exactly suitable for small organizations since small organizations in general do not have the resources required to implement full SPI programs that are more targeted towards large corporations (Mishra and Mishra, 2009b). There has been an identified need for tailored SPI frameworks for small organizations and several have also been suggested, one of them being *an Approach for Software Process Establishment in Micro and Small Companies* (ASPE-MSC) model that is especially tailored to meet the requirements of these companies (Mishra and Mishra, 2009b). It is based on various other approaches consisting of three main phases: *planning*, *monitoring and control* and *post-mortem*. The planning phase includes also the implementation itself whereas monitoring and control acts as a rapid feedback for the effectiveness of the plan and suggests updates if necessary. The post-mortem phase then focuses on gathering feedback about the iteration to prepare for the next one. The whole process is facilitated by a process engineer, that is usually suggested to be an external consultant to achieve the best results.

Generally the most challenging part of SPI or any other process improvement is lack of motivation and commitment towards it, both from senior management and employees (Allison, 2010). Therefore to succeed in pro-

cess improvement, a key factor is senior management commitment that significantly reflects to motivation of other personnel as well. However, the managerial commitment alone is not enough as the improvement will be implemented by the employees after all, which makes involvement and influence to their daily activities a major focus point. (Herranz et al., 2014) The main difference between large and small companies on SPI is the level of commitment, which is usually high in small companies (Basri and O'Connor, 2010b). Therefore the key challenge of SPI in small organizations is not commitment to change, but lack of resources, which is especially problematic when considering implementing SPI frameworks. As budget and deadlines are already tight, there is rarely extra personnel available to facilitate SPI, even though it could yield a high return of investment in the future (Allison, 2010)(Larrucea et al., 2016). Additionally processes in small organizations are generally informal and small, which makes applying standardized SPI frameworks useless (Basri and O'Connor, 2010b). In addition, despite the high commitment on process improvement, there might be change resistance also present in small organizations as people generally would like to stick to the habits they are used to, rather than experimenting with new ones (Larrucea et al., 2016). This could be a result of prior experiences on failed process improvement experiments and therefore it is important to address these experiences before implementing any improvements to avoid facing issues related to them (Allison, 2010).

Despite various challenges and a possibility of a failure in SPI, it has several benefits that make it worth the effort. In addition to improved performance and efficiency, improved processes can also lead to an increased customer satisfaction due to faster delivery times and a better customer service. Improved processes also tend to yield more satisfied development teams as they can focus on completing their work efficiently rather than wasting time on inefficient processes, which can mean for example having unprepared meetings with a wide audience. This naturally increases productivity as well. (Sulayman and Mendes, 2009) On the other hand also the company culture can be improved by SPI as having efficient processes and high job satisfaction correlates with more flexible and open organizational culture. Lee et al. (2016) also argue that a clan-like culture, where the whole organization is like a family with low hierarchy, supports knowledge sharing much better than the traditional hierarchical culture. As a conclusion, it can be argued that SPI greatly supports organizational development on various aspects in addition to improved business performance.

2.4 Measuring software maintenance

To succeed in SPI, it is important to be able to measure the initial state and changes in it. There are various metrics proposed to measure different aspects of software maintenance. The most relevant ones for this case context are briefly listed below and they are later evaluated together with custom defined metrics in section 5.1.

Reopen rate - The ratio of reopened tickets basically meaning that the tickets have been marked as resolved, but the resolution did not satisfy the original reporter and thus they need to be readdressed. A high reopen rate could indicate a need for further technical training for the maintenance personnel or deeper problems with projects in general. (Livy, 2017)

Tickets resolved on the first iteration - The ratio of tickets resolved on the first try without a need for further communication. A higher rate usually correlates with better customer satisfaction and the overall maturity of the maintenance personnel. (Livy, 2017)

SLA violations - The ratio of service level agreement (SLA) violations out of all tickets. Higher ratio indicates severe problems in maintenance process as the company fails to meet its SLA compliance, which is most likely a violation of maintenance contract. A higher ratio could indicate lack of resources on the maintenance process, but could also mean that the maintenance personnel are unaware of the SLA limits. Either way, this is an important metric for the business side of the maintenance process. (Livy, 2017)

Cost of a ticket - Internal cost of solving incidents. This metric helps to identify the most expensive tickets so that their resolution process can be improved. This is one of the important aspects to measure when evaluating the business performance of the maintenance process. (Livy, 2017)

Response time - Time between reporting and the first reaction to a ticket. Could correlate with customer satisfaction as faster response times usually mean more satisfied customers. A longer average response time could also indicate lack of resources and correlate with several other metrics such as resolution times and the number of active tickets. (Paschke and Schnappinger-Gerull, 2006)

Resolution time - The average time to resolve tickets on each level. This fits together with the SLA violation ratio and it can provide early warnings when the resolution times are growing so that changes can be made before the SLA limits are violated. This metric works as a good measurement on the overall performance of the maintenance process and could also correlate with customer satisfaction. Therefore it could be argued that it is a good

general metric for the maintenance process. (Livy, 2017)

Customer satisfaction - Not a well-defined metric like others, but still an important aspect to measure on the maintenance process. Customer satisfaction could be correlated with other metrics such as resolution time and reopen rate, but it would still be beneficial to define a direct way to measure satisfaction itself. (Livy, 2017)

Number of active tickets - The amount of tickets that have been reported, but are not yet resolved. An increasing number of concurrently active tickets could mean lack of resources on the maintenance process as the personnel can not resolve the tickets fast enough to keep up with the pace of newly created tickets. This could correlate with reopen rate and is most likely also visible as longer resolution times, but it is good to have it as a separate metric to identify lack of resources early enough before things get worse. (Livy, 2017)

Tickets by criticality - The amount of tickets on each criticality level. For example a higher amount of high priority issues could indicate issues in the latest releases and therefore quality assurance, but could also mean vaguely defined criticality criteria, which could be a sign of immaturity of the maintenance process. (Livy, 2017)

Tickets from unofficial sources - The amount or ratio of tickets reported outside of the official maintenance process. This metric is only an approximation of all unofficial tickets since not all of them will ever be visible in the official maintenance process, but it still gives an approximation of the problem. A higher amount of unofficial tickets could indicate problems in the customer interface of the main process or just lack of training on the customer side. Either way it would be good to direct more of the unofficial tickets to the official process. (Livy, 2017)

Ticket count per project or team - The tickets grouped by projects or teams. This metric could help indicate problematic projects or teams that require a more detailed inspection of the root causes for problems (Livy, 2017). Could be beneficial, but could also lower team morale if not addressed in a considerate way.

System downtime - A direct measurement on availability of the system. This could be included in the maintenance contract making it an important measure. Tolerance for downtime naturally depends highly on the context and the business criticality of the system. (Paschke and Schnappinger-Gerull, 2006)

2.5 Common solutions

As the challenges of software maintenance and knowledge sharing presented in the previous sections are quite well studied, there are also several proposed solutions intended to solve them. The most suitable solutions for the context of this case study are presented here and together they represent a good variety of different solutions that could be applied to solve the challenges of the case company.

2.5.1 Pair programming

As sharing tacit knowledge is generally hard and requires effort, it is often considered a challenge in software companies. One excellent method for knowledge sharing and especially transferring tacit knowledge is pair programming. It generally means that two persons are working together on one computer so that one is doing the task, being the *driver*, while the other one is constantly observing, being the *navigator* (Williams, 2010)(Wray, 2010). They are constantly discussing about the task at hand, evaluating different aspects of the problem and creating the solution together. The roles are not fixed and they can be constantly changed on the fly whenever necessary to achieve the best results (Gupta et al., 2013)(Wray, 2010).

Although it has not been universally proven to be always beneficial (Maguire et al., 2014), there are several claimed benefits of pair programming that can occur if organized successfully. First of all, pair programming encourages sharing especially tacit knowledge between individuals (Maguire et al., 2014)(Plonka et al., 2015)(Williams, 2010)(Wray, 2010)(Zieris and Prechelt, 2014). By working on the same task on one computer, individuals automatically start to communicate about the task and share thoughts about the solution, which leads to a mutual understanding and knowledge co-creation while solving the task at hand. Knowledge sharing effect works the best with pairs of one junior and one senior when junior is acting as the *driver* actually conducting the task and senior is acting as the *navigator* by giving advice and mentoring the junior to solve the task (Plonka et al., 2012)(Plonka et al., 2015)(Sun, 2011). This forces the senior to verbalize knowledge instead of just working with the task while the junior is trying to keep up with the pace and eventually disengages from the task (Plonka et al., 2012)(Plonka et al., 2015).

In addition to knowledge sharing, literature also suggests that pair programming could help to solve problems faster (Hannay et al., 2009)(Lui et al., 2010)(Wray, 2010), improve code quality (Hannay et al., 2009)(Williams,

2010)(Wray, 2010) and improve team spirit (Maguire et al., 2014)(Stapel et al., 2010)(Williams, 2010). First of all, it has been shown that the overall resolution time for tasks of any complexity could be faster than solving the task individually (Hannay et al., 2009)(Lui et al., 2010)(Sun, 2011). However this comes with a cost of investing the time of two persons rather than just one, which results in a higher cost for solving the task (Lui et al., 2010)(Spohrer et al., 2013)(Williams, 2010). Secondly, improved quality of code is evident as there are constantly two persons who can spot mistakes easier (Spohrer et al., 2013)(Wray, 2010) and push for working according to the best practices (Williams, 2010)(Wray, 2010). The effect of pair pressure also causes both persons to concentrate on the task at hand rather than switching to private tasks frequently (Sillitti et al., 2011)(Stapel et al., 2010)(Williams, 2010)(Wray, 2010). However it has been shown that while gaining more experience and being more familiar with each other, the amount of off-topic discussions is likely to increase (Stapel et al., 2010). Off-topic discussions are not inherently a bad thing as they help to increase team spirit and might ease out conversations about the actual topic as well (Williams, 2010). These off-topic discussions could be directed towards breaks between tasks by agreeing about common guidelines on pair programming (Maguire et al., 2014) and having a physical work environment that supports both concentration and informal discussions (Mishra and Mishra, 2009a).

However these benefits do not come without a cost. The main drawback of pair programming is the increased investment of human capital especially with people without prior experience of pair programming (Spohrer et al., 2013)(Williams, 2010). Due to this, pair programming is not the best approach for situations with a high market pressure or tight scheduling since solo programming is more efficient when considering the total cost of solving the task (Sun, 2011). Another key challenge to address is disengagement, meaning that the one acting as a navigator can be tempted to lose focus on the task at hand due to time pressure or being unable to follow the driver's work (Plonka et al., 2012)(Williams, 2010). This could be solved by letting the junior act as a driver and carefully selecting the tasks to be solved with pair programming (Plonka et al., 2012). In addition to selecting the tasks, it is also important to select the right pairs to solve them (Gupta et al., 2013)(Hannay et al., 2009)(Sun, 2011)(Williams, 2010). As mentioned, junior-senior pairs generally result in the best knowledge sharing experience, but sometimes the task requires the highest possible quality, meaning that senior-senior pair would be a better choice (Sun, 2011). It is also important to rotate the pairs frequently so that the employees get experiences with different partners and their working habits (Spohrer et al., 2013)(Williams, 2010).

2.5.2 README

Since agile methodologies do not generally encourage creating and maintaining an extensive documentation (Basri and O'Connor, 2011)(Gupta et al., 2013), but lacking proper documents is still a major challenge in knowledge sharing (Ersoy and Mahdy, 2015), some compromise between heavy documentation and lacking one entirely is needed. One proposed solution to create and maintain lighter, more informal documentation is a "README" file. It is usually located in the root directory of a project's codebase and written in a plaintext format, such as markdown (Karimzadeh and Hoffman, 2017)(van Gompel et al., 2016). It is especially important in open source community as it gives the first impression to anyone visiting the codebase and acts as a summary of the project (Karimzadeh and Hoffman, 2017)(Prana et al., 2018). There is no single standardized structure for a README and there are various practices that are highly context dependent, although generally the README aims to provide answers to "what" and "how" questions about the project (Prana et al., 2018).

The most common contents of the README include some background or a high-level description of the project, usage examples and building and installation instructions (Ikeda et al., 2019)(Karimzadeh and Hoffman, 2017)(Prana et al., 2018)(van Gompel et al., 2016). In addition to those, also updated support information, environments, release notes and links to further documentation could be included, but these are naturally dependent on the context of the project (Ikeda et al., 2019)(Karimzadeh and Hoffman, 2017)(van Gompel et al., 2016). Ikeda et al. (2019) also point out that some form of a task list or known limitations alongside with the application programming interface (API) documentation could be added to the README if considered necessary. The clear benefit of including these details alongside the codebase is the fact that documentation tends to get outdated quite rapidly and having it in version control system together with the codebase encourages updating it more frequently and usually results in a better updated documentation, even though it would be in an informal format (van Gompel et al., 2016). Informal documents such as README files also act as a support for project handover and ensure that critical knowledge is kept together with the codebase to avoid knowledge loss (Stettina and Kroon, 2013).

2.5.3 Project presentations

Another solution to tackle knowledge sharing challenges and improve available informal documentation, could be regular presentations about different ongoing projects. Presentation as a communication method is probably fa-

miliar to most people already from school since they are commonly used in education, but also regularly in a work environment (Bhattacharyya, 2011). Therefore most people should be quite familiar about giving presentations about different subjects under their expertise. A clear benefit of giving presentations is also the fact that it forces the experts to share their knowledge in a concise and structured format, which might clarify the topic for themselves as well. Also since it is beneficial to have a slide show to support learning (Wecker, 2012), informal and concise documentation is created as a side effect.

The natural drawback of the presentations is the effort that is required to prepare them in contrast to the expected benefits. It can be also seen as an additional work that prevents working on actual tasks that are considered more important, which could cause lack of motivation towards presentations. Additionally the applicability of a slide show as documentation could be debatable and it could get outdated quite fast, which could lead to misconceptions in the worst case.

2.5.4 Triaging

The initial incidents reported to the ticket tracking system could be of varying quality. Some might be very detailed with precise descriptions, while others do not provide enough information to even start working on the ticket. This is challenging as it could be time consuming to actually identify what the ticket is about and who could solve it. The problem effectively gets worse while the amount of tickets grows and therefore a structured approach, such as a triaging process, is needed. In software context triage means gathering all the necessary information needed to solve the ticket, before finding the right person to assign it to (Ruefle et al., 2013). This way developers can focus on development tasks rather than having to deal with vaguely defined incidents that might be entirely out of their area of expertise. With low ticket volumes this process can be done manually, but once the company grows and a growing amount of tickets floods the help desk, the manual techniques do not scale very well and the maintenance personnel will eventually be unable to handle new tickets (Hu et al., 2014). This is why various automatic processes for triaging incidents are proposed. These are usually based on machine learning techniques with hundreds of thousands of sample incidents as a training material and their purpose is usually to evaluate the priority and find the right assignee for the incident (Hu et al., 2014). Because of the need for training data, these techniques are mainly suitable for large corporations or open source communities with a large amount of historical incidents recorded in the database.

Improving ticket quality is beneficial for all stakeholders participating in the maintenance process as an issue tracking system should be the main medium for communication between them, containing all the information related to the ticket handling process (Bertram et al., 2010). However, customer might not have the technical expertise to report incidents with all the details required to efficiently solve them and on the other hand, developers most likely do not have time and motivation to constantly check for newly created tickets when there is also some new development to be done. A solution to this problem of shared responsibility would be job rotation (Fægri, 2009) and a rotating triage responsibility. This way the developers could focus on their main tasks while spending some predetermined time to triage tickets for other developers. This could be an efficient solution, but its implementation needs to be carefully considered as people still like to get their main tasks done efficiently and adding additional job rotation to slow down the main activities could cause loss of motivation (Fægri, 2009).

2.5.5 Automatic alerts

Another way to improve ticket quality is to detect the incidents automatically, before anyone needs to report them manually. This can be done with improved application monitoring, which alerts the maintenance personnel if certain conditions are met. There are various aspects that cloud application monitoring could benefit, troubleshooting being only one of them (Aceto et al., 2013), but as cloud application management is a relatively new concept in the field due to an increased popularity of cloud services, their efficient management is still quite challenging and heavily context dependent (Fatema et al., 2014). One of the most challenging tasks is to find the right set of key performance indicators (KPI) to measure. This is especially important in SaaS business as the amount of customers could be constantly growing and an automatic detection of incidents and performance issues significantly improves the service quality (Shao et al., 2010).

The most significant benefit of automatic alerts in comparison to customer created tickets is the quality and amount of details included. As alerts are generated automatically on predefined events, it is much easier to find the root cause of the incident and make more informed decisions about its priority for example (Aceto et al., 2013)(Fatema et al., 2014)(Shao et al., 2010). These alerts should be connected to the help desk in order to add them to the maintenance process rather than directly notifying individual developers or even teams (Hernantes et al., 2015). The main challenge with alerts is recognizing the need for alert in different situations as they are highly context dependent and it might be easy to create a lot of unnecessary alerts about

performance issues that do not require any actual actions to be taken. On the other hand failing to alert about actual incidents is also a problem and therefore the right balance between unnecessary and missing alerts should be determined based on the business criticality of the system.

2.5.6 Transition model

An important phase in software lifecycle management is the transition from active development to maintenance mode. During this transition there is a risk for losing information due to lack of updated documentation delivered for the maintenance personnel, which results in inefficient maintenance (Stettina and Kroon, 2013). One of the most important factors for any project is the background information about its context, business case and requirements and this information is lost easily as it is rarely documented anywhere while using agile development methodologies (Ito et al., 2016). One solution to this problem is introducing maintenance personnel to the project already during development, or even having developers also do the maintenance work (Stettina and Kroon, 2013). This however does not solve the risk of personnel changes during maintenance by an employee leaving the company or new employees joining to the team (Ito et al., 2016) and therefore it is important to have an actual transition model from active development to maintenance, including standardized requirements for documentation and maintenance practicalities.

Ito et al. (2016) identify three main challenges for transition. These are unrevised documents, unclear background knowledge and omitting necessary information. They suggest that the challenge of having unrevised documentation out of sync with the actual codebase could be solved by adding an update history to each document clarifying the latest changes to them. Stettina and Kroon (2013) also highlight developers' mindset of creating documentation for someone else and thus lacking the motivation for adding detailed information. Commitment to artifacts is generally a challenge alongside with a time pressure leading to lower quality documentation lacking the necessary information (Stettina and Kroon, 2013). This could be tackled by encouraging informal documents like README files (see section 2.5.2) and storing them with the codebase to encourage updates to them. Additionally documentation files should be reviewed by other team members or maintenance personnel, and updated based on their feedback about the necessary information (Ito et al., 2016).

2.5.7 WIP limits

As time pressure and multitasking are a major problem for knowledge sharing, tackling them should be taken seriously. One way to manage time pressure and parallel streams of work in agile development is WIP limits. Their main purpose is to ensure that the previous tickets have been finished before new ones can be started, which could prevent over commitment that easily leads to time pressure and multitasking (Ahmad et al., 2016). If implemented correctly, WIP limits could improve the efficiency of both maintenance and development processes (Concas et al., 2013). However, introducing limits most likely does not solve the root cause of the problems, but it encourages tackling them as multitasking is no longer an option. Also introducing new strict practices to daily activities is not an easy task as it requires commitment from the whole team (Ahmad et al., 2013) and a clear internal prioritization of different tasks, which might be problematic in some contexts.

Chapter 3

Research methods

This chapter describes the design of this study. First, the selected approach is discussed and compared to other suitable approaches. After that, all data collection methods are presented, which is followed by a description of each research phase in detail.

3.1 Research approach

The aim of this study was to solve an abstract problem in the case company's context, meaning that it could not be solved by using only quantitative methods. Since qualitative methods typically assume that the world is socially constructed and context-dependent, they fit the setting of this research better than quantitative methods (Merriam, 2002). A description of possible approaches together with an evaluation on their suitability for this study is presented next, which leads to a justified selection of an approach for this study.

3.1.1 Comparison of possible approaches

As mentioned, the goal of this study was to solve an abstract problem that the case company has identified. Possible approaches to reach this goal could be *case study, design science, ethnography* or *action research* (Easterbrook et al., 2008).

Case study

Case study has many contradicting definitions in software engineering literature causing confusion (Easterbrook et al., 2008)(Runeson and Höst, 2009)(Rune-

son et al., 2012). A wide range of studies varying from lab experiments to large field operations could be called a case study, but at the same time, similar studies are also called *field studies* or *observational studies* (Runeson et al., 2012). Despite the confusion, both Easterbrook et al. (2008) and Runeson and Höst (2009) agree that the definition presented by Yin (2003, p. 13) is the correct one that actually defines the term *"case study"* as it should be used. According to Yin (2003, p. 13) a case study investigates a phenomenon in its natural context, especially when the boundary between the context and the phenomenon itself is unclear. So case studies focus on understanding a certain phenomenon in a certain context describing the reasons and causalities behind it. A case study is a useful approach for real-life situations where the boundary between a phenomenon and its environment is rarely clear-cut, which rules out controlled experiments for example. A precondition for conducting a case study is to have a clear and concise research question focusing on explaining how and why certain phenomenon occurs or exists. (Easterbrook et al., 2008)

There are generally two types of case studies: *exploratory* and *confirmatory*. Exploratory case studies are initial studies that aim to understand some phenomenon and formulate theories about its presence, whereas confirmatory case studies aim to validate existing theories in new cases. Both of them contribute to scientific knowledge by deriving new theories and validating existing ones. Despite the benefits of conducting a research in a natural context, it also poses threats to validity as researchers might be biased on case selection, but also when evaluating the results as the researchers might interpret the findings to the favor of their theory. Therefore it is important to have an explicit framework for analyzing the findings and selecting the cases. Also a case study in a single case context should not be considered as a strong evidence and it should be confirmed with several other cases before the theory can be considered valid. (Easterbrook et al., 2008)

A case study is not exactly fit for the purpose of this study as it focuses on observing the phenomenon and deriving theories out of it rather than solving the actual problem, which is the interest of the case company. A pure case study would be appropriate if this study would be only focusing on the problem definition and its root causes.

Design science

Design science is an approach that focuses on an engineering manner of creating artifacts to solve problems either in a real-life context or in a conceptual setting (Offermann et al., 2009). The problems selected are usually socio-technological, meaning that the researchers need to understand the social

and technological environment they are working on (Baskerville et al., 2009). The artifact could be anything tangible that is created during the research process to address the identified problem, for example a piece of software or a process model (Peffers et al., 2007). Unlike a case study, design science can be conducted in a conceptual environment providing scientific knowledge that can then be applied in a real-life context (Offermann et al., 2009). Generally design science research consists of three main parts: *problem identification*, *solution design* and *evaluation* (Offermann et al., 2009), but also a more detailed process has been proposed by Peffers et al. (2007) including six steps: *problem identification*, *definition of the objectives*, *design and development*, *demonstration*, *evaluation* and *communication*. The general process is still similar starting from identifying the problem, continuing with development and finally ending with the evaluation of the developed solution.

Design science is a suitable approach when a certain problem has already been identified and it is assumed that it can be solved by developing a single artifact, which also contributes to general scientific knowledge by deriving theory about the design of such artifact (Offermann et al., 2009). It could have been a possible approach for this study as it focuses on solving the problem rather than just observing and reporting it. However the problem in the case is abstract and a single artifact to solve it could not be identified, which effectively rules out the possibility of a design science approach.

Ethnography

Ethnography is an approach that aims to understand social culture and practices by objectively observing them in their natural context (Robinson et al., 2007). Ethnographical research does not consider any existing frameworks or theories before observations and it does not make any assumptions about the studied community. The goal of this approach is to understand how the social culture of the studied community creates practices that are present and how the community communicates and collaborates. Ethnography is usually relying on outside observations where researcher stays outside the community and makes observations. Additional details might be added through interviews, but the main findings are drawn from the observations. However in some cases outside observations might not provide an in-depth understanding of the culture and therefore researchers need to become participants in the community. This is called *participant observation* and it might yield a better understanding of the community, but it might also affect the results by introducing a bias to the setting. (Easterbrook et al., 2008)

Ethnography is a useful approach in cases where the culture and practices of the studied community is the focus point of interest. It requires significant

amount of time spent on the field, but it also yields detailed findings about the culture and practices that are in use. The challenge of this approach is the time requirement and threats to validity that are inevitable since the participants are either aware that they are being studied or the researcher needs to be a solid part of the community, which has its own issues of bias. Ethnography is not a suitable approach for this study as it will only consider the culture and practices of the community rather than identifying and solving its problems. It is also notable that objective findings are not possible as the researcher has already been working for the case company for a relatively long time and is already part of the community.

Action research

The last one of the approaches is action research, which Runeson and Höst (2009) define to be focused on a change process through involvement in it. Action research is closely related to case study in a sense that they are both interested in studying a phenomenon in its natural context. The main difference between them is the fact that action research is focusing on solving the problem rather than just identifying it. Another difference is the fact that action research assumes that the need to solve the problem is always justified and self-evident whereas case study does not address solving the problem at all. It is also notable that solving the problem is done through involvement, so researchers are basically involved in the process they are studying (Easterbrook et al., 2008). Action research is also similar to design science on an abstract level as they both aim to solve the problem identified. The main difference between them are the methods of solving the problem since design science relies on developing an artifact whereas action research accepts all means of involvement to solve the problem. Another difference between action research and design science is their philosophical stance as action research is usually considered constructivist and anti-positivist whereas design science tends to have a positivist stance meaning that design science relies solely on logic and observations whereas action research accepts that the social context can not be studied extensively with scientific methods. (Baskerville et al., 2009)

Even though action research has been criticized for being an immature approach and lacking scientific validity, it is still a suitable approach for this study. This is because it fulfills the purpose of this study by focusing on a real effect on an actual problem rather than creating new general scientific knowledge. Even though action research is usually characterized as an iterative method having more than one cycle of development, testing and validating (Easterbrook et al., 2008), only one cycle will fit into the scope of

this study due to limited time and resources. However this study serves as the first cycle of continuous improvement that will most likely continue after this study as well. Another exception to the typical action research approach is the philosophical stance of positivism instead of anti-positivism (Easterbrook et al., 2008) as the change is measured using predetermined metrics that are selected to reflect the aspects that the case company considers important to improve. These metrics and their justification are presented in detail in section 5.1. In addition to the normal set of data collection methods for action research including interviews and observations, also a survey was used alongside with the data from existing systems to quantify the initial situation. Survey and data exports were selected to provide triangulation for qualitative data and to gain further insight on the problem and the effectiveness of changes. Data export methods are presented in section 3.2.4 and details of the survey in section 3.2.2 and appendix A.

3.2 Data collection

This study utilized four data collection methods mainly focusing on qualitative instead of quantitative data. This is because the problem to be solved is abstract and action research relies on qualitative data on problem identification and solution evaluation. For qualitative data, interviews and workshops were used as a data collection method. Additionally there was a need to measure the effectiveness of the implemented solutions, which effectively requires quantitative data. This was collected with a survey about maintenance knowledge and analyzing data exports from existing systems. Each of these methods are presented in detail in the following sections.

3.2.1 Interviews

Interviews are commonly used for data collection in action research (Easterbrook et al., 2008). They can be seen as a natural way of communication through talking, which might explain their popularity (Doody and Noonan, 2013). They are generally used to understand individuals, their situation, experiences and thoughts. There are generally three types of interviews that vary by their format: *structured*, *semi-structured* and *unstructured* (Doody and Noonan, 2013). This study utilized unstructured interviews as they are as close to a natural everyday interaction between the researcher and the interviewees. The term "unstructured" is a bit misleading as the interviews had a predetermined agenda and questions on the broad topic were prepared

to avoid wasting time and resources on irrelevant topics (Doody and Noonan, 2013).

Interviews were used to identify managerial viewpoints to the process improvement. At the beginning of the study, two interviews were arranged to firstly identify the used maintenance models and problems in them and secondly, to prioritize the company focus points and metrics to represent them. After the initial interviews, the next interviews were related to presenting findings from the initial state analysis, defining goals for the solution proposals and evaluating possible solutions before presenting them to a wider audience. After the implementation phase, the results were also discussed with the executives in order to find different explanations and interpretations on them.

These interviews were not audio recorded as they were on a general level and brief notes about the interviews were sufficient enough to memorize the key points.

3.2.2 Maintenance knowledge survey

Survey is a widely used method to collect both quantitative and qualitative data about a certain population (Easterbrook et al., 2008). Surveys are usually used to gather knowledge about a large population by picking a sample of it, but in smaller cases also the whole population can be included in the sample (Kasunic, 2005). The purpose of using survey in this study was to gather quantitative data about knowledge sharing that was already identified as an issue before this study. It is important to realize that survey is not a useful method to collect data about the past as people tend to remember things differently rendering its validity close to zero. Therefore surveys can only be used to measure the present situation (Linåker et al., 2015), which was also the intended use in this study. The previously identified problem and a hypotheses to be tested with the survey was that knowledge of certain projects was piling up on certain people. This means that there would be usually only a few persons who know certain projects in detail causing most of the maintenance tasks to fall on these persons. There was also a gut feeling about tight schedules being one of the main knowledge sharing barriers, which was also to be confirmed with the first survey in addition to evaluating several other knowledge sharing barriers identified from the literature (see section 2.2.1 for details).

For the purpose of sharing the survey for the whole company and to get the employees to answer to it, a web based survey was used as it has been proven to be easy to use and yield higher response rates than traditional paper surveys (Kasunic, 2005). Also the answers are much easier to process

when they are already in a digital format. The survey was created with Google Forms [1] that provides an easy to use interface for survey creation and analysis. To get the response rate as high as possible, the questions were kept simple and the overall survey short (Linåker et al., 2015) having only five questions: two for demographic purposes, two main questions to test the hypothesis and one voluntary open-ended question about insights outside the main questions. The survey was presented to the COO of the case company as a pilot test after which the wording was improved (Linåker et al., 2015) and some project names changed. Also the order of both projects and barriers was randomized so that there would not be any unwanted patterns due to certain order of questions. The survey was then distributed via a shared channel in Slack [2] and reminders were sent via the same channel and during face-to-face discussions. The same survey was also repeated after the implementation phase to measure the change in maintenance knowledge. The final survey form used in both cases is presented in appendix A.

The survey results were then analyzed and visualized using spreadsheets with predetermined metrics agreed with the executives. These metrics are presented in section 5.1 and the initial survey results are fully presented in section 5.2.1. The final results are presented in section 8.1 and analysis on them in chapter 9.

3.2.3 Workshops

A more collaborative way of gathering qualitative data instead of using interviews is workshops. The purpose of both workshops and interviews is the same: to gather knowledge about individuals' thoughts and feelings on some subject. The fundamental difference between interviews and workshops is their methodology, because interviews are usually discussions mainly with the researcher even in a group setting, whereas workshops encourage discussion between the participants while the researcher is facilitating and observing the discussion. Workshops are a suitable method for data collection in action research as it is a collaborative process of addressing certain problem after all (Lüscher and Lewis, 2008). Workshops also engage participants more than group interviews, which is highly beneficial for collaborative problem solving (Coghlan, 2011)(León de la Barra et al., 1997)(Tsoukas, 2009). Additionally with good facilitation, workshops encourage all participants to share their thoughts, not just the ones with the loudest voice or in top managerial positions. From researchers point of view, they are also less time consuming, which means that with limited time available, there can be several workshops

[1] https://forms.google.com
[2] https://slack.com

arranged on different topics compared to having just one set of individual interviews. It is also notable that workshops compared to interviews cause a participatory feeling for the participants, who are part of the change implementation in this study. This reduces change resistance as the change has been co-created rather than just being decided in a top-down manner by the researcher and managers. Therefore it is evident that workshops are more suitable as a main data collection method than interviews for this study and thus qualitative data was collected mainly with workshops.

When designing a workshop it is important to identify the key participants, prepare necessary materials and design the structure of the session. Smeds et al. (2015) emphasize three key aspects of a successful knowledge co-creation workshop: *participants*, *boundary objects* and *external facilitators*. First of all, the participants should be selected so that each relevant team is represented to provide different viewpoints to discussions (Smeds et al., 2015). This is addressed by arranging the workshops at a time that is suitable for most of the relevant participants. The focus group of this study is development teams so the schedule was arranged according to their needs. Secondly, it is important to prepare necessary materials for the workshop to get the discussion flowing as it might be difficult to discuss about the issues if everyone has a contradicting idea about the general topic to begin with. These materials are the *boundary objects* for the discussion and their purpose is to ensure that everyone is talking about the same subject with the same terms (Akkerman and Bakker, 2011). For the purpose of this study, boundary objects such as a process diagram was used. This diagram is presented in the figure 4.2. Lastly, according to Smeds et al. (2015) it would be beneficial to have an external facilitator as he would be outside the social norms and have an external, objective viewpoint. However due to the setting of this study and the researcher's relationship with the case company this is not possible, but it is still important to remember that the goal of the facilitator is to act as an external supervisor facilitating the discussion, rather than actively participating in it.

The structural framework for the workshops organized in this study was selected to be Think-Pair-Share that was created by Lyman (1987) in the 80s. It consist of three phases: *individual thinking*, *pair discussions* and *group discussion*. The method starts by the facilitator presenting the topic and questions to be addressed. It is important that examples are not given at this point to avoid limiting the individual thinking of the participants (Lyman, 1987). The first phase, individual thinking, lasts for a short time and its purpose is to think about the problem individually while writing notes to sticky notes for example. After that the next phase is to find a pair or a group of three persons and then share the ideas from the first phase and

discuss about them again taking notes about new ideas on sticky notes. An important aspect of pairing is aiming to form pairs or groups that are not already familiar with each other to break the social boundaries and encourage truly creative thinking (León de la Barra et al., 1997). After discussing in pairs, it is time to share the thoughts together with all the participants of the workshop. At this phase, similar sticky notes are collected together to form general concepts. Group discussion phase is the longest one as it will yield the results of the whole workshop that are then documented on sticky notes. In general Think-Pair-Share is an efficient way to use time so it is suitable for organizing short 1-hour workshops on the scope of this study. During this study there were three workshops organized in total, further information about their contents are presented in section 3.3.

3.2.4 Data export from tracking systems

In addition to collecting qualitative data that is useful for gathering individuals' thoughts and feelings, which are hard to measure, also quantitative data was required to represent the effectiveness of change. The case company generally had two quantitative data sources available for this study: an issue tracking system (Freshdesk [3]) and a time tracking system (Toggl [4]). Freshdesk holds information about reported incidents and their resolution process. For example it provides information about issue resolution times, reporters, projects and categories. This information was used to evaluate the efficiency of the overall process and to identify the points of improvement. Toggl on the other hand is used for tracking time that employees spent on both development and maintenance tasks and it provides information about the projects they have been working on and the time they used for them. This data is useful for measuring overall time spending and to measure the balance between maintenance and development tasks. These datasets can also be combined to find explanations for certain aspects such as long resolution times due to busy development activities for example.

The datasets were provided as Excel spreadsheets, which were then converted to CSV format and analyzed with Python [5]. Python scripts were used to analyze both ticket data from Freshdesk and time tracking data from Toggl. The values for metrics were then copied to a spreadsheet again for visualization purposes. The reason to use Python for analyzing the data was complexity of the initial dataset that was not directly suitable for spreadsheet analysis on selected metrics, thus requiring intermediate parsing with

[3]https://freshdesk.com
[4]https://toggl.com
[5]https://python.org

Python. Metrics selected to be analyzed were agreed with the CTO of the case company to reflect the most important focus points from the company's perspective. These metrics are presented in detail in section 5.1 and their initial values in section 5.2. The exports were also repeated after the implementation phase to measure the change in selected metrics. The results of the change analysis are presented in section 8.1 and implications on them are further discussed in chapter 9.

3.3 Research phases

Action research in general has five main phases: *diagnosis*, *action planning*, *intervention*, *evaluation* and *reflection* (Davison et al., 2004). This study follows the general guideline by starting with analyzing the initial state, which is followed by proposing solutions. The next phase after agreeing about the actions to be taken was the implementation phase, which was followed by evaluation and reflection. Generally it would be beneficial to have several iterations of this cycle (Davison et al., 2004), but due to limitations of a master's thesis, there is only one cycle in total, which provides the methodology and a foundation for future improvements. Next each of these phases are presented in detail.

3.3.1 Initial state analysis

The first phase of this study was naturally analyzing the initial state. The goal of this phase was to identify the actual problem and the case company's needs for improvement. After this phase there was a mutual understanding of the key problems in the initial process and their internal priority, since they were not equally important to be solved from the case company's perspective. The phase began by having an interview with the company's CTO, COO and CCO about the initial maintenance processes and their evolution over time. During the first interview also already identified problems or thoughts about possible problems were discussed on a general level, which guided the decision of focus areas for a further study. One of the problems initially identified was knowledge sharing and it was decided that it requires further studying in a form of a survey to get a quantified analysis of the initial situation. The survey was designed based on literature findings about knowledge sharing barriers in software development and it was conducted at the beginning of February. This survey is presented in appendix A.

In addition to the first interview and the survey, also another interview was organized. The second interview was with the CTO and its purpose was

to identify the ways to measure the change and the most important aspects for improvement. A list of possible metrics were gathered from literature in addition to the initial ideas for focus points, which were then discussed with the CTO resulting in a prioritized list of a few metrics to focus on. These metrics are presented in more detail in section 5.1.

Also a workshop was organized to further analyze the problems of the initial state and to get a broader viewpoint from the employees. The workshop was organized at the beginning of February with a total of six participants in addition to the researcher. Three different teams were represented in the workshop by four developers and two consultants, which covers most of the case company's organization (see section 4.1). The executives were also invited to the workshop, but were unable to attend due to unexpected changes in schedules. The topics of the workshop were covered later with the CTO and CCO to get an executive point of view to the discussed topics. The workshop notes were collected on sticky notes and later analyzed to form a basis for the solution proposals. Results of the first workshop are presented in section 5.2.

3.3.2 Proposing solutions

The findings from the initial state analysis were then formulated into goals that the solution proposals would aim to meet. These goals were discussed and prioritized together with the executives to find out the most important goals to focus on, as all of the challenges can not be solved with limited resources. These goals and their priorities are presented in section 6.1 together with justification for the prioritization. After the initial analysis of the problems and prioritization of goals, a further literature review was conducted in order to find possible solutions from theoretical frameworks and previous case studies. The findings from the literature review were then applied to the context of the case company for better applicability and formulated into concise solution proposals. These proposals were then evaluated with the management to filter out those with low expected return of invested resources and those not suitable for the company's high level business strategy. The remaining proposals after strategic filtering with the executives are presented in section 6.2.

These proposals were then presented to employees in the second workshop to gather wider feedback about their suitability and implementation. The workshop was organized at the end of February with the same target audience as in the first workshop. The reason why proposals were evaluated and modified together was to ensure involvement and commitment to the change process, which is essential to succeed in SPI. Therefore rather than presenting

finalized solutions, the workshop consisted of a brief introduction to the idea behind each proposal, which were then evaluated with the Think-Pair-Share method. After gathering feedback and some modifications to the initial proposals, the most promising ones were selected for the implementation phase. The selected solutions are presented in section 6.3.

3.3.3 Implementation

A few days after selecting the solutions for the implementation phase in the second workshop, another meeting with the COO and CCO was arranged to address the practicalities of implementing the selected solutions. During the meeting, the solutions were discussed one by one to identify the practical actions required to implement them. To ensure that no time is wasted on misunderstandings and lack of communication, each task was also assigned to a responsible, who would ensure that it will get done on time. Most of the preparatory tasks were naturally assigned to the researcher, but some tasks required modifications to the time tracking service for example, and therefore could only be conducted by the executives with an administrative access to the service. Once all preparations were done, the guidelines were first evaluated together with the executives, before presenting them to all employees in a company wide Slack channel. After ensuring that the implemented solutions started working as intended, it was time to observe the implementations and discuss about them informally to get early feedback about them. This was important because the short time period of the research required fast reaction to possible issues in the solutions to ensure that no time is wasted. The implementation phase lasted for a total of seven weeks between March and April.

3.3.4 Evaluation and reflection

After seven weeks of implementation phase, the third workshop was organized at the end of April marking a conclusion to the implementation phase. The purpose of the third workshop was to gather feedback about the implemented solutions and their efficiency in solving the challenges identified at the beginning of this study. The workshop was open for the whole organization to ensure that everyone has a chance to share their thoughts on the topic. The workshop had six participants in addition to the researcher representing all teams of the case company and therefore it can be stated that the feedback gathered represents the whole organization and not just a single team. However some key persons, including the CTO for example, were unable to attend due to busy schedule, but their insights were addressed

separately with an unstructured interview based on the workshop findings.

In addition to the third workshop, also the knowledge sharing survey was repeated together with new data exports to measure the change in selected metrics. The analysis on these metrics supported the qualitative findings of the workshop and gave further insight on the effectiveness of the implemented solutions. As the time frame of this study was limited and the whole cycle of this study was considered as the first iteration of continuous improvement, it was important to accurately measure the resulting state of the organization. This measurement could then be used as a starting point for the next iteration of improvements, building on top of the findings and experiences from this study.

Chapter 4

Context

This chapter describes the context of this study. It starts by generally introducing the case company and its core businesses. After this the focus is on software maintenance processes in the case company. After describing the processes, also the company focus points are presented to clarify the initial motivational factors behind this study.

4.1 Case company

The case organization of this study is QOCO Systems Ltd., a small software company founded in 2009 focusing on aviation industry with solid expertise in software solutions for maintenance, repair and operations (MRO) in aviation business. During all of its lifespan there has been collaboration with Airline 1, which is still the most important customer at the moment. A few years ago there was also quite much collaboration with a railway company since their challenges with MRO were relatively similar to those of airlines. However, this collaboration has been discontinued due to strategic focusing on the aviation market. At the start of this study, there were 16 employees at QOCO, which marks a 100 % increase in 12 months. Also the revenue has been growing steadily by around 200k€ annually, currently being a bit over 1M€ (Finder.fi, 2019). Basically it could be argued that QOCO is past the initial startup phase as it has become a steadily profitable company with several employees and a solid customer base.

Main services offered by QOCO can be classified into two main categories: software services and consulting. During the last few years consulting has been the most important revenue source for QOCO and therefore it still has a major role in QOCO's service portfolio. Recently the main focus has been shifting towards Software as a Service (SaaS) business since consulting

services require specialized expertise, making them hard to scale up. SaaS solutions are selected as a strategical focus point because they can be scaled much easier and do not require as much in-depth technical knowledge about aviation business, which is a relatively rare skill especially when combined with software development skills.

On the consulting side, QOCO's expertise is deeply focused on AMOS MRO System (AMOS), a system developed by Swiss AviationSoftware Ltd. and used by many airlines worldwide for organizing their MRO processes. The main focus is on developing integrations between AMOS and various other systems, but QOCO also provides support for data migration and several other AMOS related processes.

On the software solutions' side, QOCO has developed some SaaS solutions that are readily available for new customers. The portfolio consists mainly of web applications and integrations, but QOCO has also worked with mobile applications in the past, however they are not the main focus area at the moment. One example of a web application is MROTools.io, a tool management software meant for airlines to track the usage of tools during aircraft maintenance and repair. Tool tracking is especially important in the aviation industry as a single missing tool could prevent an aircraft from departing before it is found. On the integration services, QOCO has developed a data exchange service, EngineData.io, for automatic data transfer from aircraft engines to the engine manufacturer's data analytics system so that it can be cross-checked with AMOS. The engine manufacturer's data analytics system then calculates the wearing of different engine parts based on real measured values, which then can be used to plan the maintenance schedule according to actual wearing, rather than using overly estimated values that are currently being used. This will optimize the maintenance schedule and increase the effective flight time of engines, which will result in significant cost-savings over time. As the interface to the data analytics system is quite complex, QOCO has developed EngineData.io to hide the complexity, making it much easier for the airlines to connect to the system and utilize its benefits.

The organization of QOCO is roughly divided into executives and four consulting and development teams. The executives include Managing Director, Chief Technology Officer (CTO), Chief Customer Officer (CCO) and Chief Operating Officer (COO) and they are supervising the development and consulting teams. There is only one pure consulting team consisting of two consultants allocated for Airline 1 and they are responsible for AMOS related migrations, reporting and other tasks typically related to data engineering. Another team mainly on the consulting side, but more focused on developing highly tailored web applications and integrations is also allocated for Airline 1. The team consists of one senior consultant that acts mostly

as a product owner (PO) and is therefore ensuring that the team will constantly provide value for the customer by prioritizing tasks and managing the communication between the team and the customer. In addition to the PO, there is also one senior and two junior developers assigned to the team. There are also two product teams: one with four members focusing on MROTools and the other one with two members developing EngineData. There are no separate POs in those teams, but one of the executives will ensure that the product is developing into the right direction. The teams and their members are also presented in figure 4.1.

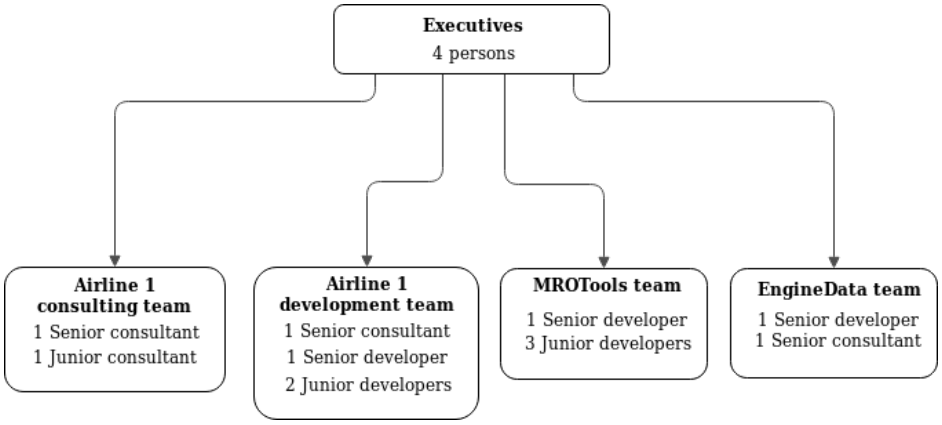

Figure 4.1: Organization structure as of January 2019

4.2 Software maintenance in the case company

As described, QOCO has grown rapidly during the last five years and that led to a point where the lack of a well defined software maintenance process was causing trouble in day-to-day business and continuous interruptions made it difficult to focus on actual development tasks. Previously the maintenance process was based on directly contacting the contact person via email or phone. Usually the contact person was the CTO who is also the founder of the company so he has a long background with the customers making him the most natural point of contact for them. At the same time there were also new application management service (AMS) contracts being signed and those included requirements for response and resolution times for example. Because of these new service level agreements (SLA), the previous ”best

effort" type of maintenance process could not be accepted anymore and a need for a well defined maintenance process was identified.

As a core of the process improvement, QOCO decided to compare different options for issue tracking in spring 2017. A few requirements for the issue tracking system were identified: most importantly it should be easy to use with a possibility for different contacting methods, including email and phone. It should also be reasonably priced for a group of 5 - 30 users and it should support SLA modeling out of the box. Nice things to have in addition were support for reporting out of the box and grouping users by different customers, but these were not considered as critical requirements. Five different issue tracking systems were evaluated in addition to considering implementing a custom integration for reporting incidents directly to an internal issue tracking system. After a careful evaluation, QOCO chose Freshdesk out of its competitors to be taken into use in June 2017. Some key aspects affecting the choice were its good quality for a reasonable price, ease of use and a good variety of out of the box features such as reporting and integrations to different contacting methods. There is also a support for grouping users for different customers which improves its scalability in the long term if QOCO continues to grow as it has been for the last few years.

In QOCO's case, maintenance activities can roughly be divided into two categories: general support and application management with further development. The maintenance tasks vary between SaaS products and consulting services, because SaaS products are sold with the maintenance plan including further development whereas consulting services are sold as a single delivery with a minimal support plan keeping the service up and running, but further development will always require new negotiations. QOCO focuses mainly on in-depth technical support rather than providing basic user support such as password resetting. This is mainly due to company's history of being a specialized consulting service provider for Airline 1 having only a few employees, which makes it incapable of handling basic user support that would require dedicated customer service personnel. Also the nature of services provided is deeply technical so there has not been a need for basic user support in a large scale. However due to the recent strategic shift to providing products, like MROTools and EngineData, alongside with consulting has raised the demand for it, which also poses new challenges for existing processes on scalability.

Currently there are two main channels to contact QOCO in case of incidents: a support email address for incidents that are not business critical and an emergency phone number for critical incidents. In addition to these channels there are also some customers contacting their contact persons directly by email, mostly because of the previous model that was based on

direct contacts, but QOCO is trying to direct these requests to the official channels as well. Next both of the official channels are presented in more detail.

4.2.1 Support email address

Support email address is used for low, medium and high priority incidents during the normal office hours. The process starts with a message to the support email address that then automatically creates a ticket to Freshdesk with low priority and notifies relevant users by email. Relevant users are determined currently by certain domain rules, mapping each customer to a group of Freshdesk users. After the notification, the ticket waits until someone will have a look on it and determine its priority, tasks to be done and who will be addressing it. The ticket is then assigned for a consultant or a developer who will do the actual work. It is important to note that, depending on the issue, this might be the same person that also did the prioritizing, but not in all cases as all of the developers and consultants do not have access to Freshdesk. The access is limited because most of the incidents require further clarifications from the customer, which makes it natural for the contact person to handle the communication with the customer, letting the developers and consultants focus on their work. After the clarifications, the contact person will then describe the issue for the consultant or developer who then can work efficiently on resolving it.

During the resolution phase the assignee works on the issue and discusses about it with other team members and the contact person. In some cases there might be some further information required from the customer or some third party, which will be shown as "waiting" status on Freshdesk meaning that QOCO's SLA timers are not running while waiting for a response. When the issue gets resolved, its resolution is discussed with the customer and it will be agreed whether it requires any further actions to be taken.

The whole process is presented in figure 4.2. As can be seen, the main interaction channel with the customer is the support email that will send notifications to the customer if there are new comments added to the ticket. This way all of the information related to some incident is always available on Freshdesk and not on someone's personal email account like it used to be in the previous model.

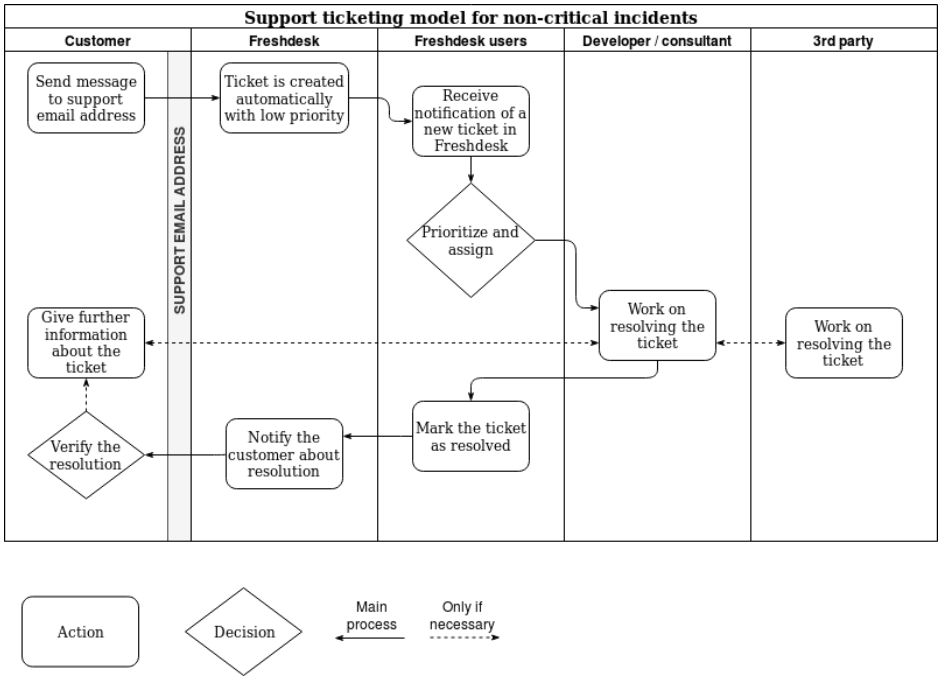

Figure 4.2: Process for handling low, medium and high priority incidents

4.2.2 Emergency phone

In addition to the general support ticketing process, there is also an emergency phone number that the customers can call at any time. The emergency phone is meant only for business critical issues that need to be dealt with immediately regardless of the time of day. This is crucial in aviation business as the business is operated in three shifts 24 hours a day, 7 days a week, and if the systems were to encounter critical incidents, they can not wait for the next morning before someone addresses them. This is why QOCO has taken the emergency phone into use. It currently works as a service that first tries to redirect the call to the COO and if he does not answer, it will be redirected to the CTO and the loop goes on until one of them picks the call. Due to nature of the incidents and their possible occurrence outside of normal office hours, QOCO's executives usually deal with them alone without any need to involve other employees. However in some cases the developers and consultants might have particular expertise that helps to solve the issue faster and thus they might provide some help if necessary. The process is presented in figure 4.3 and as can be seen, it is much more straightforward compared to the normal process of handling low, medium and high priority incidents. However, the main process is still quite similar: the customer informs QOCO

about the issue, QOCO deals with it and discusses about the resolution with the customer who then verifies the resolution.

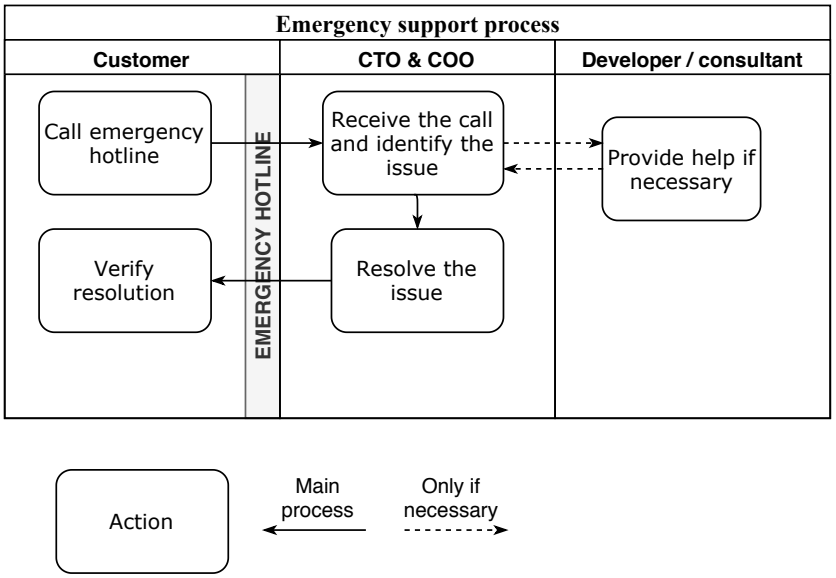

Figure 4.3: Process for handling critical incidents

4.2.3 Application management and further development

In addition to resolving incidents and providing support for customers when they ask for it, QOCO also has agreed to provide application management as a part of maintenance. In general this means updating dependencies when necessary, monitoring capacities and reacting to alerts raised by crossing certain limits for example. In some cases the alerts have been integrated with Freshdesk by sending automated messages to the support email address, which will then trigger the normal process for handling such incidents. However the alerts are usually not integrated to Freshdesk right away and currently it is more like a nice additional feature rather than being required from the very beginning for each project, which is a clear point of improvement. Other tasks such as updating the dependencies are also not addressed on a regular basis. Of course as QOCO follows the latest discussion on its field, the version updates on relevant libraries and frameworks sometimes become a hot topic due to known vulnerabilities or support plans coming to an end. This then triggers the action on QOCO's side to take the necessary steps to keep the delivered software up to date to avoid those issues. Other than that, general updates to newer versions of the dependencies occur in an

unstructured way for example with new releases.

For further development part, the process is more open for different communication channels. Sometimes requests for new features or changes to existing ones are sent to the support email address which then starts the process of incident handling. In those cases the ticket will be most likely closed in Freshdesk without further actions and the requirements and schedule for the new feature are discussed with the customer outside of Freshdesk since it is not a support task that is meant to be handled in Freshdesk. The requests can also be presented in meetings, by email or by phone directly to QOCO's personnel that then discusses and analyzes the requests in detail eventually agreeing about a development plan with the customer. There are some differences between custom made consulting services and general products in terms of further development, the main difference being that further development is usually already included in the maintenance plan for delivered products. This means that there is no need for additional discussions about budgeting if the issue is estimated to fit into the agreed workload. On the consulting side, new features and changes to existing ones are not usually sold with the initial contract that only covers the development of the first version and its support. This means that all requests need to be discussed with the customer to agree about the budget and implementation schedule. This also makes it harder for QOCO to suggest new features that could generate more sales in the future.

4.3 Company focus points

The key focus points for QOCO were identified in two separate interviews, first with the CTO, CCO and COO, the second with only CTO. The topics for those interviews were to discuss about the current maintenance process, already identified issues with it and metrics that could represent the situation. The metrics would also be used to measure the change after the implementation phase to see if the implemented solutions had any effect on the identified issues. During the first interview it was evident that the main focus of the thesis should be on internal process improvements rather than having significant efforts on external processes that are more or less out of QOCO's reach.

One of the main problems already identified at this point was knowledge sharing, meaning that most of the maintenance tasks were handled by the CTO and COO without engaging other personnel. This poses a risk as this way of working is not scalable in the future. It is assumed that there will be

a growing amount of maintenance related activities as the company matures and more and more projects are shifted to maintenance mode from active development leading to more serious problems in the future. The fact that the CTO and COO handled most of the tasks affected further sales and managerial activities that consume an increasing amount of time when the company grows and therefore maintenance knowledge needs to be shared to other personnel as well. In addition to that, there was no clear responsibility for anyone to respond and resolve the newly created tickets, which easily leads to a "someone will do it" attitude. This naturally causes further uncertainty on overall process and causes unnecessary delays.

Another important initial idea about the main challenges was the relationship between maintenance and development tasks. The overall feeling was that maintenance tasks interrupt the development work, which results in frequent context switches slowing down both development and maintenance tasks. There was also a feeling that this could be due to undefined process practices at least on some level, but at this point it could not be evaluated further in detail. One of the reasons for the unstructured relationship between maintenance and development tasks was thought to be lack of a structured handover to the maintenance phase. On the other hand, there is no separate maintenance personnel since the developers handle also maintenance of the developed systems, which explains the close relationship of different tasks.

Customer satisfaction was also discussed in both interviews as it was identified to be important for the company. The initial thought was that customer satisfaction is relatively good, which was based on an overall feeling and some interviews conducted a while ago. However a well defined metric to actually measure it was not defined and therefore the initial feeling about a relatively good customer satisfaction was not based on any recent measurements. This was clearly a point of improvement although it was considered to be less critical than internal process improvements.

Chapter 5

Current state analysis

This chapter presents the analysis on the initial state of the case company. It starts by defining the metrics selected to represent the state of knowledge sharing and maintenance process and after that, continues by presenting the initial values for these metrics.

5.1 Measuring the maintenance process

The main topic for the second interview was to identify the key metrics that represent the situation and could be used to measure the change after the implementation phase. The interview was prepared by finding general software maintenance metrics from the literature (see section 2.4) in addition to custom defined metrics specifically designed for QOCO's context.

Custom created metrics

In addition to metrics from literature, also custom created metrics specific for QOCO's challenges were added to the discussion. These metrics are more suitable for QOCO as they are specially created for the context, rather than being general metrics. This also means that their validity needs to be carefully addressed to ensure that they work as intended. These custom metrics are presented below.

Projects per person - The average amount of projects on different skill levels a single person can contribute to. This works as a measure of knowledge sharing as a higher amount of projects means better knowledge sharing and wider group of employees who can handle different maintenance tasks.

Persons per project - The average amount of persons on different skill levels for a typical project. In addition to measuring the amount of projects,

it is necessary to also measure the amount of persons for different projects to analyze how much skill sets overlap between persons. For example a high number of projects per person, but a low number of persons per project could indicate that there are some projects that are relatively well known by many people, but also some that are relying on certain key persons.

Knowledge sharing barriers - A measure of the most and least significant barriers for knowledge sharing. This gives an idea about the main factors preventing knowledge sharing and their significance, but on the other hand, also information about which barriers are not an issue at QOCO. This is important to measure since knowledge sharing was considered a major problem in the initial state, but there was no clear idea about the significance of individual barriers. The evaluated barriers are presented in section 2.2.1.

Tickets by type - Amount or ratio of tickets by their types. Freshdesk has seven different types defined, including info, question, request, change request, alert, incident and problem. Especially interesting is the amount of actual incidents (alert, incident, problem) in comparison to informative and request tickets, because it represents the true load of maintenance tasks that actually might require fast response with development activities. These are especially interesting due to initial thoughts about the challenges between development and maintenance processes.

Amount of preventable tickets - The amount of tickets that could have been prevented before a customer needs to report them. Measuring this directly is difficult as there is no general definition for a preventable ticket, but it is still an important factor since an increasing number of preventable tickets could mean lower customer satisfaction. As a proxy for measuring preventable tickets, tickets by type could be used with a focus on *ratio of alert tickets*. Higher ratio of alerts out of total tickets would mean earlier reaction without the need to wait for the customer to report the incidents. This is not exactly the same as the amount of preventable tickets, but it works as a proxy quite well.

Tickets that actually belong to QOCO - Amount or ratio of tickets that require actions from QOCO rather than being related to some third party system. This is an interesting measure as there are several third party systems that QOCO has developed integrations to, but does not have the maintenance responsibility on the system itself. Still a considerable amount of tickets related to those systems are reported to QOCO causing unnecessary work on QOCO's side, when there is nothing that can actually be done for the ticket. Lower amount of these tickets would mean better communication of QOCO's responsibilities towards customers and would increase productivity of the maintenance process.

Time to production - Time required for a ticket resolution to actually

reach production as there are some tasks that can not be deployed to production directly, even though they have been resolved. Lower time would mean faster actual resolution times correlating with better customer satisfaction, but also reducing the amount of multitasking since already resolved tickets do not require additional focus in the future.

Development speed - There is no single metric that would always be applicable to measure the development speed, but finding the right metrics for the case company would be beneficial to be able to measure the actual effects of maintenance tasks on development process. However, there are no such metrics used at QOCO currently so measuring the initial state of development speed is impossible, but it is still worth considering as a point for improvement.

Waiting for customer - The time that the tickets are waiting for customer reply over the total resolution time. This metric represents communication issues with the customer mainly due inactivity on the customer side. It would help to identify whether long resolution times are actually an internal or an external problem.

Waiting for a third party - Similarly to previous metric, this measures the time ratio of waiting for a third party over the total resolution time. This also helps to identify external issues with communication to third parties and clarifies whether issues are internal or external.

Selected metrics

The presented metrics were carefully evaluated during the second interview, which purpose was to identify the main focus points for improvement and the most critical metrics to measure the effectiveness of the implementation. The selected metrics can be divided into two categories: maintenance knowledge metrics and process metrics.

First of all, three metrics were selected to measure maintenance knowledge: *projects per person*, *persons per project* and *knowledge sharing barriers*. These were considered important as one of the main issues identified already in the first interview was sharing knowledge about maintenance of different projects, which was highly concentrated on key persons. However there was no clear understanding of the barriers that actually prevent sharing and therefore adding knowledge sharing barriers as a metric is essential. All of the three metrics can also be used to measure the effectiveness of the implementation since success should be visible as a higher amount of projects for each person with a wider distribution of skill sets at the same time. Also the most significant barriers should also be lowered if not entirely removed after the implementation while not presenting any new barriers to the organization.

The most important factors of improving the process was identified to be reducing *tickets from unofficial sources* and increasing the *ratio of alerts* out of total tickets to enhance the ability to recognize and resolve tickets before customers needs to report them to QOCO. Additionally *average resolution time* was considered as an important general metric that should be kept within reasonable limits, but the main focus was not to aim for faster resolution rates as SLA violations were not an issue. *Waiting for third party or customer* was also a metric that raised interests, but after a discussion it was evident that access to data about it would be problematic and thus they were left out of focus.

To measure multitasking, a decision to use time tracking data about different *projects and time entries per day* was made. These metrics generally represent the amount of context switching on average for different persons and teams and thus help to identify whether the problem is team specific or more general. An additional idea was to measure *maintenance hours ratio*, but the dataset was not sufficient for that purpose. The idea of that would have been to measure the actual ratio between maintenance and development activities, which would represent the relationship between maintenance and development, but there was no reliable way to measure it and thus it was left out of focus.

5.2 Findings

The initial state analysis utilized three data collection methods to identify the challenges. Next the most important findings from each method are presented together with analysis on the causalities and implications behind the data.

5.2.1 Findings from the knowledge sharing survey

The initial state analysis phase began with conducting a knowledge sharing survey as knowledge sharing was identified as a challenge during the first interview. The survey consisted of two main questions: knowledge about several projects that QOCO has implemented throughout its history and an opinion on knowledge sharing barriers. The detailed survey description is presented in appendix A. The survey was distributed utilizing Google Forms and a company-wide Slack channel to reach the whole organization. The response rate was significantly high as 12 out of 15 persons responded to the survey. Two persons were out of office during the whole survey period and one person did not respond at all (additionally one person was the researcher,

who also did not respond to avoid bias). Expertise level of respondents was evenly distributed with four juniors, four seniors and four executives. The sample is quite representative as the distribution is quite close to the actual expertise level distribution of the organization, although it is notable that there are four executives in total at QOCO so they are a bit overrepresented in the data. Also work experience distribution that is presented in figure 5.1 is close to that of the population's since the company has more than doubled its size during the last two years, which can clearly be seen in the graph.

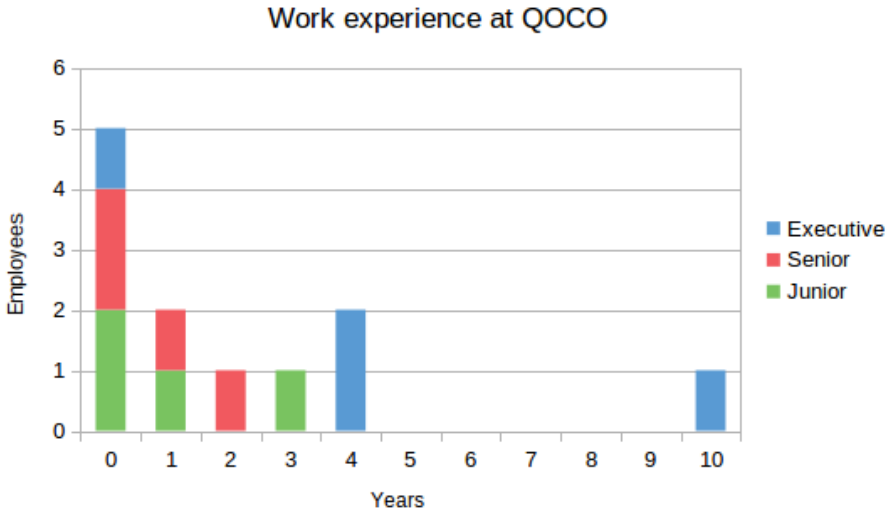

Figure 5.1: Work experience of respondents at QOCO

Project knowledge

The question about project knowledge introduced six levels of knowledge with different scores ranging from zero to five:

0. No knowledge at all
1. Knowledge about the business case
2. Knowledge about the business case and general architecture
3. Able to do minor tasks
4. Able to do major tasks
5. Detailed knowledge about the whole project

Skill levels can also be combined together. For example level 3+ means levels three to five, so basically an ability to do at least minor tasks on the project. Combining levels helps to discuss about the differences between skill sets and projects and thus the notation is used in the upcoming sections.

For the projects question, the best known projects were unsurprisingly those that have been developed lately and discussed informally during lunch and coffee breaks. For example the best known project was MROTools.io with an average knowledge of 2.8 and five persons able to do at least major tasks related to it. This was unsurprising as it has been a hot topic in discussions and has been developed entirely during the last 1.5 years with a team of four persons dedicated to it. Another example is the second best known project, Airline 1 technical operations digitalization project with an average knowledge of 2.1. It has also been going on for several years now, but it still has had quite much development interest of Airline 1 development team lately and is therefore quite well known by many developers. It can be argued that clearly having company interests on some project and discussing about it formally and informally on a daily or weekly basis helps to share the knowledge and introduce new persons to the project.

The difference between the most well known and the least known projects is quite significant. For comparison, the least well known project at QOCO is an old legacy project, with an average knowledge of 0.5 and only one person who knows anything more than just the business case. This is quite typical profile for several legacy projects that have been developed years ago by the CTO himself with possibly some employees who are no longer working at QOCO. There are still some maintenance tasks related to these projects also and there is on average only one person who can do anything about them, which is definitely a challenge for the maintenance process and a risk for the company. However it is also important to note that supporting legacy projects is not the main source of income for the company and thus having too much effort on them can reduce the capability to deliver new projects.

In addition to listing the most and least known projects, the more important aspect about project knowledge was identified to be average knowledge about projects for each expertise level and an average skill distribution for a typical project. A hypothesis from the interviews with executives was that the executives have significantly more knowledge about projects than juniors and seniors (H1), who do not have much difference between them (H2). Also it was assumed that usually juniors and seniors know only zero to two projects in detail, depending on years of experience at QOCO (H3). For the skill distribution, a hypothesis was that there is usually only one to two persons that can do at least major tasks on the project (H4), while most of the personnel hardly know the business case (H5).

The results of analyzing the project knowledge by different expertise and skill levels are presented in table 5.1. By analyzing these results it is evident that H1 holds, since the executives have a level five knowledge for 9.3 projects on average, which is much higher than 1.0 of juniors and seniors, which also fulfills H2. The gap is a bit smaller on 3+ level, where juniors have 5.5 projects on average compared to 3.0 of the seniors. This is explained by longer experience at QOCO on average and different roles, since a product team of EngineData.io consists of seniors whereas Airline 1 development team with several active projects has junior developers for example. This also fulfills H3 as it seems like years of experience at QOCO affect more to project knowledge than the level of expertise. It is still notable that the executives have 11.8 projects on average with a skill level of 3+, which is more than twice the amount of juniors. This gap means that many of the tasks related to less known projects fall for the executives who are already busy with many other tasks and new sales at the same time.

	Junior	**Senior**	**Executive**
5	1.0	1.0	9.3
3+	5.5	3.0	11.8
1+	14.5	9.8	22.8

Table 5.1: Average amount of projects at skill level

Analyzing the typical skill distribution for an average project is interesting since it represents the division of skill sets over different projects that is not present on expertise and skill level analysis. The typical distribution presented in figure 5.2 is extrapolated from 12 survey responses to cover the whole organization of 16 people so it might not be entirely accurate, but the message is still quite clear. From the distribution we can state that there are usually approximately 2.4 persons with 4+ knowledge on a certain project, so the actual amount is a bit higher than initially estimated with H4. In addition to that, there are approximately 3.2 persons with 3+ knowledge, which most likely means CTO, COO and some third person depending on the project. Even though H4 does not hold, H5 still does since in most cases more than half of the organization does not even know the business case of the project. Of course as this is just an average of all projects, there is a significant difference between older legacy projects and newer projects that are being developed constantly.

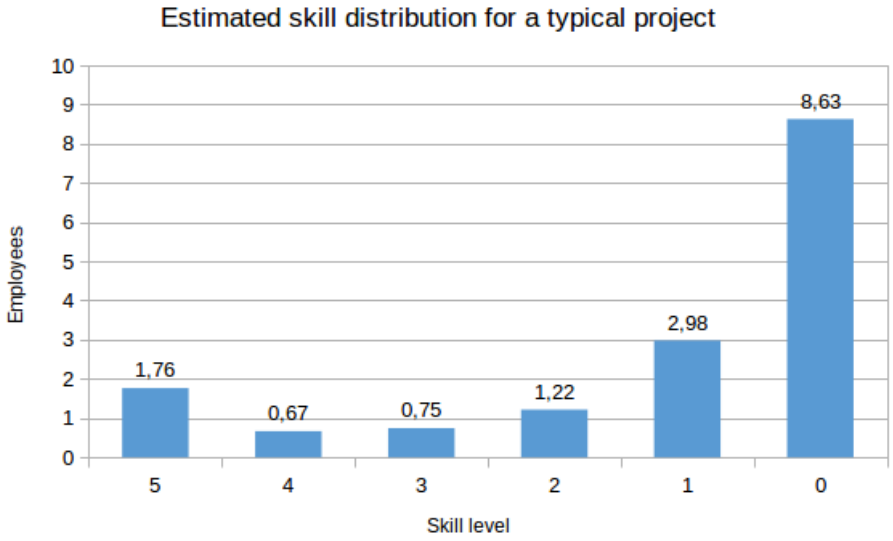

Figure 5.2: Estimated skill distribution for a typical project as of February 2019

The overall distribution looks promising in a sense that there is potential for knowledge sharing from 3+ level to lower levels since there is for example on average 1.2 persons already familiar with the business case and architecture. This could help to share some of the workload from CTO and COO to other employees so that the executives could focus on more important tasks. The fact that half of the organization does not even know the business case of a certain project is not necessarily a thing to worry about, since considering the ability to contribute to a project by completing tasks, there is not much difference between levels zero and one. The problem of tasks piling up on executives is not solved by introducing new persons only to the business case.

Knowledge sharing barriers

The second main question was about opinions on significance of different knowledge sharing barriers at QOCO. The opinion was presented using a 5-step Likert scale (Boone and Boone, 2012) with opinions from *Strongly disagree* to *Strongly agree* for each barrier. 20 barriers were evaluated in total to find out the main knowledge sharing barriers, but also those barriers that are not an issue at QOCO. The initial hypothesis from two interviews with the managers was that lack of time and resources would be the biggest

challenge together with multitasking (H6). This was assumed by a mutual understanding that schedule had been quite tight on several projects with multiple concurrent development streams constantly active. Another hypothesis was based on an overall feeling of the company culture and team spirit that was assumed to be strong and thus should not be experienced as a significant knowledge sharing barrier (H7).

The most significant knowledge sharing barriers are presented below together with their scores. The score was calculated by taking a sum of positive votes for each barrier, ie. +1 for *agree* and +2 for *strongly agree*. As can be seen, H6 clearly holds since tight schedule and multitasking were considered to be the two most significant knowledge sharing barriers. A notable barrier is also lack of informal documentation with a score of seven. This represents the situation that most of the knowledge is tacit knowledge of the CTO and COO, which is not documented anywhere. On the other hand, lack of formal documentation was not considered a significant barrier, which represents the agile culture of QOCO, since formal documentation does not fit exactly well in agile frameworks. Other significant barriers were also the complex domain of aviation industry and strong code ownership. The domain is a barrier that can not be entirely solved as QOCO is focused on aviation industry and knowledge about it is learned through years of experience. Code ownership on the other hand reflects the results of project knowledge question as the code ownership of especially legacy projects is usually at the hands of the CTO and therefore it could be solved with similar solutions that are aimed to share the knowledge of different projects from the executives to other employees.

The most significant knowledge sharing barriers at QOCO:

1. Tight schedule (15)
2. Multitasking (10)
3. Lack of informal documentation (7)
4. Complex domain (6)
5. Strong code ownership (6)

The other end of the scale is the least significant knowledge sharing barriers and as can be seen from the list below, H7 holds quite well. The list was constructed similarly by giving score of +1 for *disagree* and +2 for *strongly disagree*. The barriers related to organizational culture are considered as least significant at QOCO and therefore it can be stated that the overall culture

of QOCO is open for discussion and supportive towards knowledge sharing even though *organizational culture* itself did not make it to the list with a score of seven. It is evident that the employees have high motivation and trust on each other, which is required to share knowledge in the first place. It is good to see that the main challenges of knowledge sharing are not related to attitudes and organizational culture, that are harder to solve than the barriers related to organizing work such as scheduling and multitasking.

The least significant knowledge sharing barriers at QOCO:

1. Physical work environment (14)
2. Difference in age (14)
3. Lack of motivation (13)
4. Lack of trust (13)
5. Strong organizational structure (12)

In addition to the two main questions there was also an additional voluntary question about open thoughts on knowledge sharing challenges at QOCO. The responses reflect the results of the main questions as one respondent says that "*... there is very little 'free' time for knowledge sharing unless absolutely necessary*", which emphasizes the challenge of tight scheduling as there is no surplus time when all the deadlines are closing in. Also several respondents emphasized the knowledge gap between the executives and others by stating that "*those with the most knowledge are also the busiest*" and "*key persons such as (CTO) and (COO) have lot of maintenance responsibilities which is a challenge for new business*" latter of which raises the concern about the ability to sell new projects due to maintenance tasks that keep these key persons busy. Another point to note about general knowledge sharing is the fact that "*... the tasks are usually given to the most experienced in each task*", which effectively prevents knowledge sharing and enforces code ownership that has been identified as the 5th most significant barrier. One respondent also pointed out the strong team boundary between Airline 1 consulting and development teams since "*there is little communication between consulting and coding team*", which is a bit odd considering the fact that they are working for the same customer on similar projects even though the focus of the tasks is different. Increased communication could clearly benefit both of them.

To conclude the findings from the knowledge sharing survey, the company culture is supportive towards knowledge sharing and therefore the focus should be on solving the tight scheduling and multitasking issues that are considered to be the two most significant barriers. This leads to a situation where many of the maintenance tasks are solved by the CTO or COO, which effectively prevents knowledge sharing and will not be scalable in the future. To encourage sharing of knowledge some of the maintenance tasks could be shared to other employees as well even though it might mean increased costs in short term, but this will be necessary considering long term scalability. By delegating some of the tasks to other employees, the executives could also focus on more important future sales and current development projects that will generate majority of the revenue in the future.

5.2.2 Findings from the first workshop

The first workshop was organized to identify the issues related to the maintenance process as knowledge sharing was already addressed with the survey. The workshop was targeted for MROTools and Airline 1 development teams as special interest towards relationship between maintenance and development was considered important. Additionally the executives and Airline 1 consulting team were invited, but only one person from consulting team and none of the executives could attend the workshop. However, the topics were later discussed with the executives separately. The results of the first workshop can be divided into three main categories: process related findings, customer related findings and findings related to the tools being used.

Process related findings

First of all, one of the main challenges of the initial maintenance process was the fact that there was a lot of informal communication outside of the official Freshdesk process. This is clearly evident on the reaction of the participants when the official model in figure 4.2 was presented as a boundary object. Comments such as "*that is a nice model, but it is nowhere near the actual process*" were presented, which represents clearly how well the official process is adopted internally. This informal communication via different channels often leads to situations where some of the information has been lost or altered when the issue reaches the person responsible for solving it. This is problematic since missing information means that the responsible person needs to dig for that either from the customer or colleagues and in the worst case, starts to work on a completely different problem than was originally reported. There is also a practical problem of billing the hours used for

the maintenance task if it is reported by chatting by the coffee machine rather than actually reporting it to Freshdesk. The problem of informal communication about maintenance tasks is also visible as concentration of the customer interface around individual persons for each project who then delivers the information forward, if he remembers to do so. This means that if the person will not be available at some times, the process is essentially broken since the customer does not have anyone to contact in case of issues.

Another internal process related finding was that initially there was no clearly defined person responsible for addressing the issues on Freshdesk when they are reported there. The initial process was usually that "someone" will address them when they have time, that "someone" usually being the CTO or COO. It was also said that developers do not know about the time limits on SLA contracts, which makes it rather hard to prioritize maintenance and development tasks among each other. Prioritization of the maintenance tickets is also relatively difficult as there was no clear policy on different priorities, meaning that most of the tickets were of "low" priority. The main issues of lacking proper internal prioritization guideline and vaguely defined responsibility on maintenance tasks means that the maintenance tasks will disturb development work by pausing it in the middle of a flow just to realize that the task is actually of low priority and could be addressed later with a better time slot. These interruptions cause developers to lose their focus for a while, which slows down the development work and causes inefficiency.

In addition to unclear responsibilities, also the tagging feature in Freshdesk was rarely used. If properly used, it could help tracking issues related to certain project for example, rather than just having a list of all tickets visible at the same time. This results in inefficient search functionality and causes loss of time when searching for issues related to own line of work. One additional finding related to the process was about customer satisfaction that is not measured with any actual metric. There have been some interviews a while ago, but the current understanding of a relatively good customer satisfaction is not based on any tangible metric. Of course it could be argued that the customer would tell if they were unsatisfied with some aspects, but there is still room for improvement in the current situation. Specified and tracked metrics of customer satisfaction could help QOCO to react to changes in customer satisfaction faster, resulting in better overall customer experience.

Customer related findings

The root cause for informal communication are the customer's representatives who use several channels for contacting QOCO personnel including, but not limited to, email, phone calls, Slack and chatting by the coffee machine. The

use of these channels then continues on QOCO's side and the task might never be visible on Freshdesk even though it is a clear maintenance task. Of course QOCO as a small company can not enforce a strict maintenance process since flexibility and customer satisfaction are its main advantages as a small company. If QOCO would enforce the official maintenance process too strictly, it would be an easy decision to switch partner to some large general software service provider with a similar process, but benefits of a large corporation. Therefore even though informal communication is problematic and usually initiated by the customer, the solution to it is not trivial and requires careful assessment of both internal and external processes including customer satisfaction as a highly important factor.

In addition to informal channels being used for communication, also initial requests are usually of low quality and require further clarifications from the customer. Relatively common issue is that "something is not working" with a screen capture of some system QOCO's developers have never seen. This might be due to technical skills of the end users that are experiencing the issues, but another point noted in the workshop is that many of the issues are "highly critical" according to the customer, while still lacking information required to resolve the issue. It might even be the case that the reporter of the ticket is not available or has no skills required to provide clarifications. When the low quality of initial reports, informal communication channels and an undefined responsibility of responding to tickets discussed earlier are combined together, it is no wonder that the maintenance process can be considered as inefficient and causes interruptions on development.

The expectations on response and resolution times are also unbalanced in some cases since the tickets might be waiting for customer reply for a long time, but still the customer considers every tickets as critical and expects responses and resolutions fast. This causes some of the prioritization and interruption issues mentioned earlier, but also frustration on developers' side as the ticket might be pending for a long time, but when the information arrives, it suddenly becomes a top priority ticket.

Findings on tools being used

On the tools side, the single most important finding was that QOCO had a plan on Freshdesk that allows five users to be added to the system. This is highly problematic since there are currently 16 employees at the company with at least 12 of them participating in maintenance activities. There has been a habit of rotating the fifth user for the one in need in each situation, but it is obvious that the official process can not be used by having five licenses for 12 employees. This problem is also only going to get worse in the future if the

company keeps on growing. The limited number of user licenses will further encourage informal communication outside of Freshdesk as not everyone will have access to the information stored there. Additionally it was also noted that the email client provided by Freshdesk has several usability issues, which will further encourage direct use of informal email communication outside of Freshdesk.

In addition to the usability issues of the email client, another major drawback on usability is the search functionality that does not allow users to query tickets by their content, which would be highly beneficial in cases where the ticket title only contains "not working" for example. This makes it hard and time consuming to search for certain tickets and makes the use of Freshdesk inefficient. This usability issue goes together with missing tags, that could be used for querying. Together these two challenges make it unnecessary difficult to find tickets and could be beneficial to improve the overall process.

What comes to the challenge of internal prioritization, another challenge in the current process is separation of maintenance and development tickets to two completely different systems: Freshdesk and JIRA [1]. This means that these two systems are often out of sync, which makes organizing and prioritizing work difficult and may cause delays to both maintenance and development tasks. In addition to internal JIRA, at least Airline 1 development team has customer's own JIRA for higher level issue tracking, which introduces a new problem of synchronizing three separate systems. Generally having multiple separate systems slows down all processes as the systems need to be manually updated and checked for new updates, which causes unnecessary trouble for the development teams and should be solved.

To conclude the main findings of the first workshop, it could be stated that the old ad hoc process of best effort is still visible as informal communication and customer interfaces that are concentrated on certain key persons. On the other hand it is also hard to enforce the official model even internally when there are not enough user licenses for each person participating in the maintenance process. Also ticket quality and proper prioritization should be emphasized as there is currently an unbalanced understanding of priorities as customer tends to consider everything as highly critical, whereas most of the tickets are classified as low priority on Freshdesk. These are the main points of improvement identified in the workshop. These should be carefully considered when creating the solution proposals and comparing them with each other.

[1] https://fi.atlassian.com/software/jira

5.2.3 Findings from the data exports

The last data collection method used in the initial state analysis was analyzing data exports from existing systems, namely ticket tracking system Freshdesk and time tracking system Toggl. The goal of the data analysis was to find the state of the maintenance process by evaluating several predetermined metrics including the amount of automatic tickets, ticket reporters, resolution times and context switches.

Freshdesk

Freshdesk data export from 2018 provides information about individual tickets including for example their resolution time and priority. The first interesting factor to measure was the amount of automatic tickets, ie. alerts, in relation to other ticket types. Distribution of the ticket types is presented in figure 5.3. From the graph it is notable that only 7 % of tickets are automatic alerts while most of the issues are still reported manually. It has been identified that the amount of alerts in comparison to other ticket types should be higher since it would mean that the tasks could be resolved already before the customers notice anything. This would mean a better customer experience, but also improve efficiency of the maintenance process as it is usually easier to track down the source of an alert compared to a manually reported incident.

The measure of tasks that are reported through informal channels can be approximated by analyzing the reporter of each ticket. In most cases the tickets that are reported internally have originated through informal channels and have been added to Freshdesk by QOCO's employee meaning that the reporter domain will be qoco. The analysis of the ticket reporter domains is presented in figure 5.4 and it can be seen that approximately 10 % of all tickets are reported through informal channels. However it is notable that the whole truth about tasks in informal channels is not visible in Freshdesk tickets as not all of the tickets in informal channels end up as tickets to Freshdesk, which makes it hard to estimate the actual proportion. From the graph we can also see that only 2 % of the tickets come from 3rd party vendor that is handling the first level of support for Airline 1. Basically this means that Airline 1 reports some first level tickets directly to QOCO instead of going through the official process of reporting to the third party vendor, which then escalates the ticket to QOCO if necessary.

As argued, the changes in reporter domain proportions work as a proxy for estimating the amount of tickets that are reported through informal channels. It needs to be noted that increased proportion of internally reported

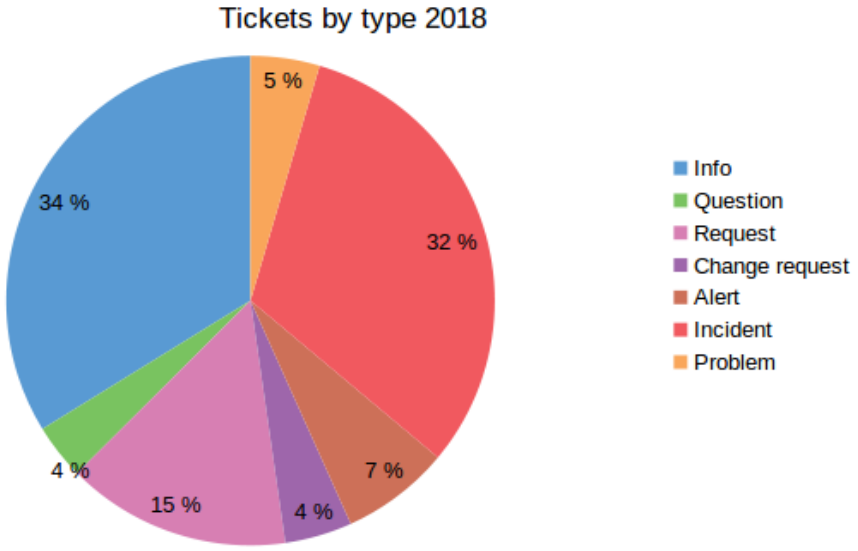

Figure 5.3: Ticket types, excluding tickets without a type

tickets could be interpreted as increasing amount of tickets in informal channels, when it actually meant that just larger amount of informal tickets are also reported to Freshdesk, which naturally is a good thing. Therefore the reporter domains can not be analyzed alone, but they need to be analyzed together with the total amount of tickets, which is presented in figure 5.5.

One of the most important measures of process efficiency and customer experience is the average resolution time of tickets. This is presented as a monthly average over 2018 in figure 5.5 together with the total ticket count to emphasize the relationship between them. The graph includes only low priority tickets, because the few tickets that were on other priority levels had several hundreds or even thousands of hours in resolution time, meaning that they were exceptions to the general maintenance process. As can be seen in the graph, the average resolution time has been in a downtrend throughout 2018, but interestingly there seems to be an upward spike every four months. There was no clear explanation for these spikes, but each time there seems to be an increase in total ticket amount during the previous month, probably at least partly affecting the resolution times of the next month as well. Another point to note was that there was also different production deployments and initializations of several systems during those spikes, which also partly explain longer resolution times. Generally it could be stated that the overall resolution times and total ticket counts have been in a downtrend, which is

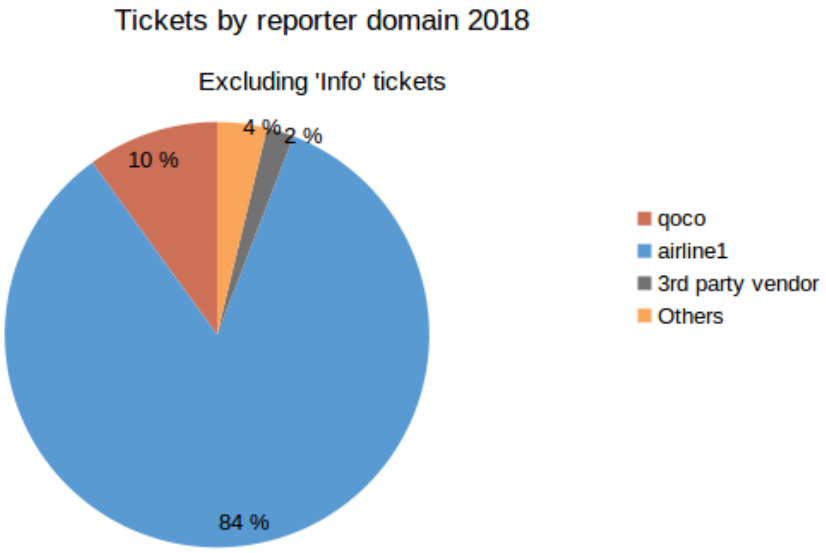

Figure 5.4: Tickets by reporter domain

positive and should continue similarly in the future as well.

Toggl

In addition to the ticket data from Freshdesk, there was also time tracking data available from Toggl. Especially interesting aspects about that was to measure the amount of multitasking and therefore context switches that cause interruptions to productivity. The results of the team based analysis are presented in table 5.2. The table excludes EngineData team since it has been established in January 2019 and thus was not present in the 2018 dataset. The analysis consisted of two metrics: projects per day meaning the amount of different projects the teams work during the day on average, and the amount of time entries per day each basically representing actual context switches. The dataset does not include all of QOCO's projects and therefore the values are only approximations of the real values, but they still can be used to compare different teams with each other.

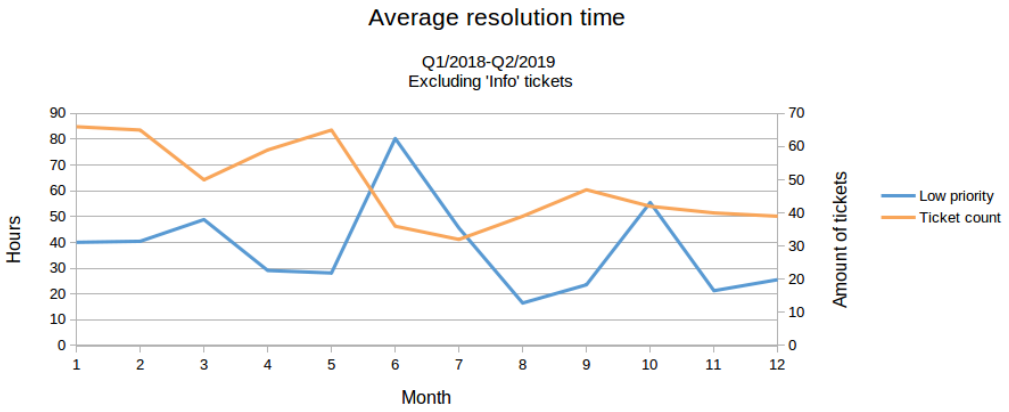

Figure 5.5: Average resolution time by month

	Projects / day	**Time entries / day**
Airline 1 consulting	1.1	1.4
Airline 1 development	1.7	3.3
Executives	1.8	2.4
MROTools	1.1	2.1

Table 5.2: Average context switches per day

From the table it is notable that the executives and Airline 1 development team tend to multitask more than Airline 1 consulting and MROTools teams. This can be explained by different types of work as MROTools team focuses on a single SaaS product development and the consulting team handles mainly support tasks that are all grouped under the same project. Airline 1 development team on the other hand has had many parallel development projects on at the same time, resulting in a higher multitasking factor. To further analyze this challenge, the multitasking factor of Airline 1 development team is analyzed on a monthly basis in figure 5.6. It can be seen that the average level of time entries per day is between 2.5 and 3.5, but during July and August the value has been significantly higher than that. This is alerting as an increased amount of context switches generally slows down both maintenance and development tasks and therefore it should be considered when proposing solutions at least for Airline 1 development team.

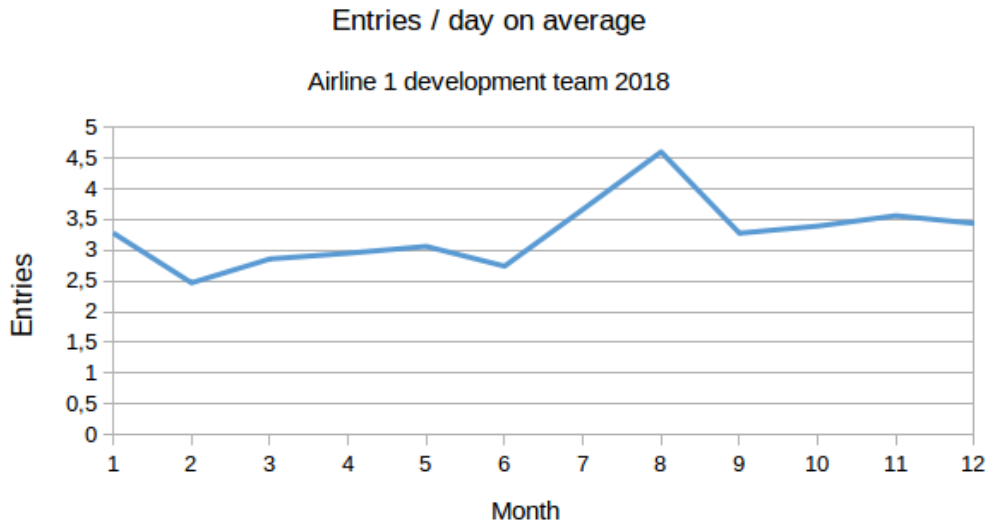

Figure 5.6: Average time entries per day for Airline 1 development team

To conclude the main findings from the data exports it is evident that the maintenance process is not exactly working as it is supposed to. Even though the resolution times and total ticket amounts have been generally decreasing, it is alerting that there are still regular spikes on resolution times every four months. Also multitasking seems to be an issue mostly for Airline 1 development team, which lowers the productivity of that team and slows down both development and maintenance processes.

Chapter 6

Solution proposals

This chapter presents the proposed solutions to solve the challenges identified in the initial state analysis. Before presenting the actual solution proposals, goals for the solutions are identified and prioritized to better understand the intended outcomes of the implemented solutions. After this, each of the initially proposed solutions are presented after which the feedback and modifications to them are described. This finally leads to selecting the solutions for the implementation phase.

6.1 Goals for solutions

The findings of the initial state analysis were discussed with the COO together with an evaluation for goals of the implementation to identify their internal priority. These goals and prioritizations are next presented in detail, but also listed in table 6.1 for easier comparison.

It was agreed that the most important challenge to be solved was knowledge sharing and the barriers preventing it. Knowledge sharing was considered a top priority goal for the improvement since it was the most significant threat for scalability of the maintenance process. Initially there was a significant gap on project knowledge at 3+ level between the executives and other employees (see table 5.1). The gap was even larger at level 5 skills, but as the time period of the thesis is quite short and acquiring skills required at level 5 might require years of practice, the focus is directed to 3+ level (G1), which already improves the situation. In addition to lowering the gap between the executives and others, another important aspect identified was the average amount of persons with 3+ skills for a typical project, which effectively represents the variety of skill sets in the company. It would be much better to have a variety of projects employees can contribute to, rather than having

them all focus on the same ones. Initially the amount was 3.2 (see figure 5.2) and it was agreed that 4.0 would be a good goal for improvement since it would most likely mean that there are two persons on average in addition to the CTO and COO who can contribute to the maintenance tasks (G2).

For knowledge sharing barriers the focus was directed to the top three most significant ones: tight schedule, multitasking and lack of informal documentation. Multitasking and tight schedule are closely related to each other, because they are both result of having multiple projects active in parallel with customer pressure for fast deliveries. They can also be solved with similar techniques and thus are grouped together as G3. The goal is to lower the significance of tight scheduling and multitasking initially having significance scores of 15 and 10 respectively (see section 5.2.1). The goal was not necessarily to change the order of the barriers as it could mean that the solutions have just shifted the problems to other barriers and that is why lowering the significance was agreed to be the goal for improvement on knowledge sharing barriers. Also lack of informal documentation was added to goals with high priority as it was relatively easy to tackle, but still had a relatively high significance score of 7 (G4).

Secondary goals were more related to Freshdesk and the process itself. First of all, the amount of alerts was considered as a good measure for tickets that were handled before customer needed to report them thus making it an important point for improvement. However exactly defining the tickets that could have been recognized earlier is hard and the initial situation was not considered as worrying as the situation with knowledge sharing. Therefore it was agreed that increasing the ratio of alerts out of all tickets (excluding "info" tickets) should be increased from 7 % to 10 % during the implementation phase (G5) and further improved in the future if considered necessary. Additionally another measure of Freshdesk process improvement was the ratio of internally reported tickets that serves as an estimation of tickets from informal sources. The target for implementation phase was to halve the ratio of tickets from 10 % to 5 % by encouraging the usage of official channels (G6). It was also important that because of the process improvements, the average resolution times should not increase significantly and thus it was agreed that solidifying the average resolution time to below 30 hours would be a good goal (G7). This goal also meant that the cyclic increase in resolution times (see figure 5.5) should be taken into account and tackled with process improvements.

Additionally two optional goals were specified as nice side effects of primary and secondary goals, but not necessarily goals to put much effort on. First of all, decreasing the ratio of low priority tickets in Freshdesk (G8) was considered as a measure of process clarification, since initially 94 % of the

tickets were of low priority and the definitions between different priorities were not well defined. However measuring just the ratio of low priority tickets could be potentially problematic as it would encourage to create tickets of higher priority, which is not an improvement on the process. Additionally not increasing the total amount of tickets during the implementation phase was also considered a positive factor (G9), but it was also problematic due to growth of the company essentially leading to an increasing amount of tickets. Therefore it was not a definite goal for success, but still recognized as a thing to consider, especially if the amount of tickets was to rise because of the implemented solutions.

Goal	Code	Priority
Significantly decrease the gap between the executives and others at 3+ level	G1	Primary
Increase the average amount of employees with 3+ skill on a project from 3.2 to 4.0	G2	Primary
Lower the significance of tight schedule and multitasking barriers	G3	Primary
Lower the significance of lack of informal documentation barrier	G4	Primary
Increase the ratio of alerts from 7 % to 10 %	G5	Secondary
Decrease the ratio of internally reported tickets from 10 % to 5 %	G6	Secondary
Solidify resolution time below 30 hours on average	G7	Secondary
Decrease the ratio of "low" priority tickets from 94 % to 90 %	G8	Optional
Keep the total amount of tickets constant at 40-50 monthly tickets	G9	Optional

Table 6.1: Goals for the implementation

6.2 Solutions

Challenges identified during the initial state analysis are quite well studied in software engineering literature and there are several proposed solutions to them. This section presents the proposed implementation for each of them in QOCO's context. To gain further insight on the theory behind these proposals, see section 2.5. The proposals presented here were evaluated

together with the development and consulting teams in the second workshop, after which some modifications to initial proposals were made to better fit the specific context of the company. These solutions, received feedback and modifications to initial proposals are presented in the upcoming sections.

6.2.1 Pair programming

As stated in section 2.5.1, pair programming is a good solution to solve knowledge sharing challenges and it helps to prevent tasks from piling up on certain key persons. In QOCO's case, the best effect of pair programming is most likely achieved by having the CTO or COO act as the navigator guiding the other employee who would actually learn to solve the tasks. As the maintenance tasks are not plain programming tasks, but can consist of updating configurations, verifying certain things or inspecting application logs, the term *pair programming* is a bit misleading and the actual term in QOCO's case is *pair working*, meaning working as a pair on a variety of maintenance tasks, not just the ones including programming. The tasks selected for pair working should be relatively small at least at the beginning, since strict pair working has not been tried before. A good rule of thumb would be that pair working tasks should not take more than one hour to complete at the beginning. After a while of working and gaining experience, this limit could be reconsidered. For the physical environment to work as a pair, either a group working room, that is regularly used for daily meetings, or a dedicated pair working room could be utilized. Anyway the pair working space should be different from the open cubicle to avoid disturbing other employees.

6.2.2 README template

Since lack of informal documentation is an issue for knowledge sharing and QOCO's projects rarely have README files, it was considered that a general template for a good README file could be created to encourage writing informal documentation alongside the codebase itself. As described in section 2.5.2, README file is generally an introduction to the codebase and acts as the first point of documentation for newcomers on the project. There are plenty of README examples available online that could be used as a template and after a short review of several candidates, it was decided that it would be better to just pick ideas from several templates while constructing an entirely tailored one from scratch. The template would be stored in QOCO's GitHub account to keep it easily accessible and to enable easy version control for updates to the base template. The template would then be

added to at least newly created projects and older ones with active development to improve the overall quality of available README files.

6.2.3 Project presentations

Another idea to improve informal documentation and share knowledge about interesting projects was regular project presentations. As argued in section 2.5.3, giving presentations should be quite familiar for everyone, although not necessarily an enjoyment. Therefore for the best benefit, it should not be forced for anyone, but some encouraging is most likely still needed. Ideally there could be weekly presentations during a casual company-wide meeting occurring every Friday. The meeting lasts usually about an hour and therefore for example 10 minutes could be dedicated to a weekly presentation about different projects. 10 minutes does not take too much time from casual chatting, but it is still enough to provide a clear and concise update about the project's situation. The presentations could also act as a basis for casual discussion during the weekly meeting, if there are some interesting topics for further discussions. In addition to the presentation itself, the materials are stored in a shared folder, which also addresses the problem of lacking informal documentation. Even though it was pointed out that these materials could become outdated quite fast, they still act as a general description of the business case and at least provide a contact person to ask for more information. The presentation responsibility could either be rotating between the teams in certain order, or could be reserved beforehand if there are some updates with on-going projects. The proposal is to share the responsibility on an on-demand basis, meaning that each team presents their projects if they consider it meaningful, instead of enforcing a strictly rotating responsibility on a weekly basis. However, it should still be ensured that the workload of preparing presentations is somewhat evenly distributed between different teams to ensure that each team participates equally to the knowledge sharing process.

6.2.4 Rotating triage responsibility

Another challenge that is at least partially caused by problems in knowledge sharing is maintenance tasks that keep on piling up on key persons, namely the CTO and COO in most cases. It was also said during the first workshop that initial incident reports from customers are usually of low quality and additional details are required before the work can actually be started on the incident. Low quality tickets also result in challenges with prioritization and do not generally make knowledge sharing easier in any way. A com-

mon solution to these issues generally encountered in help desk contexts is using triage principles before passing the ticket forward to a person actually responsible for solving it (see section 2.5.4).

The solution in QOCO's context is to implement a rotating triage responsibility by changing the person responsible for triaging incoming tickets on a daily basis. During the day the responsible's main task is to ensure that the newly created tickets are of good quality, correctly prioritized and assigned for a developer after all necessary details are included in the ticket. This would also support knowledge sharing as one person is responsible for handling all of the tickets created for any project during the day, which most likely requires some training and knowledge sharing at first, but will eventually result in a wider knowledge base and a more balanced workload on maintenance tasks. Before this can be implemented, the amount of Freshdesk users need to be increased from the initial five licenses. For example adding new licenses for Airline 1 development and MROTools teams is essential for the implementation of rotation. Another practical issue to consider is the amount of part-time employees as the idea is not to have them working in triage every other working day. This needs to be addressed by agreeing a certain rule of the ratio between triage and other working days. This would make the balance between triaging and working equally distributed for both full-time and part-time employees and act as a transparent way to assign responsibilities. Some way to assign the triage responsibility is also required, especially since there are quite many part-time employees, which could lead to communication and coordination issues. One way to do it would be to add a physical triage scheduling board to the office, but the same thing could also be achieved by using Doodle [1] or some other scheduling application for the purpose.

6.2.5 Improving monitoring with alerts

Another solution to tackle low quality tickets would be to detect incidents before customers need to report them. This would most likely mean improving monitoring of applications by adding automatic alerts for certain events, such as system reboots or failures to connect to a database. This aspect was also emphasized by QOCO as it would most likely result in faster resolution times and better customer satisfaction as described in section 2.5.5. Despite the benefits, it was also noted that previous experience at QOCO shows that certain events such as failure with a database connection could trigger multiple consecutive alerts when the system tries to reset the connec-

[1] https://doodle.com

tion periodically. This can be problematic as it floods the support process, but it can be tackled by better alert configuration and triaging.

The solution proposal in QOCO's context is to improve alert practices by reading about the best practices and applying them to alert configuration in one or more test projects that are newly shifted to maintenance mode or are under development. After a while, adjustments could be made and lessons learned could be discussed with other employees to share knowledge about alert configurations. These practices could then be documented and spread to new projects and added as a requirement for newly developed projects, which would improve the overall maintenance process little by little. Further evaluations based on new projects would also be beneficial to iteratively improve alert configuration knowledge in the future.

6.2.6 Transition model

As identified already in the initial interviews, there was no structured process for the transition between active development and maintenance phases. Although this was not necessarily required, since there was no separate maintenance team, it was still considered a point of improvement as having a well defined transition model could reduce knowledge loss and prepare the projects properly for the maintenance phase (see section 2.5.6).

The solution proposal in QOCO's case is to create well defined guidelines for transition from active development to maintenance mode. These guidelines could include for example the README template proposed in section 6.2.2, but in addition the practicalities about the maintenance process, informing it to the customers and configuring alerts if necessary as described in section 6.2.5. It could also take into account practices for introducing new personnel to the maintenance of different projects. The outcome of this solution would be general best practices for easy transition to avoid knowledge loss and to enable easier maintenance of the project after the main development has been completed.

6.2.7 Kanban WIP limits

In addition to other knowledge sharing barriers, multitasking was identified as problematic and it seemed to be focused on Airline 1 development team that is constantly working on several parallel projects. Airline 1 development team has decided to use Kanban as its task coordination method since previous attempts to implement Scrum iterations were not considered feasible due to changing requirements, priorities and the amount of available resources.

The basic flow of Kanban in Airline 1 development team is moving backlog

tickets to "groomed" column when they are ready to be worked on. From "groomed" they will be moved to "in progress" column when the task is under development, "waiting" when it is ready for review and "concluded" when it is deployed to some test environment. The way to limit the amount of concurrent tasks in Kanban is generally known as "WIP limits" and it means that there is a hard limit for the amount of tasks that can be under development at the same time (see section 2.5.7). The solution for Airline 1 development team is to define and enforce these hard limits for tasks under development. The actual limits are left for the team to decide as to be actually useful they need to be collaboratively agreed by all team members.

6.3 Feedback and modifications

The solution proposals were evaluated in the second workshop, which was organized at the end of February. Structure and methodology of the workshop was similar to the first workshop as Think-Pair-Share methodology worked well and resulted in a lively and versatile discussion. The discussion in the workshop focused on evaluating each solution's suitability for QOCO's context and expected efficiency in terms of invested resources and expected results. The researcher also remained strictly in the facilitator's role as presenting opinions before gathering extensive feedback would introduce bias to the workshop setting and ruin its purpose.

6.3.1 Knowledge sharing related solutions

There were initially three knowledge sharing related solutions, *pair working*, *README template* and *project presentations*. Pair working or pair programming was generally a well known method and there was a mutual agreement about its benefits. However the main concern about it was related to tight schedule and an increase in total effort that could not directly be billed from the customers. It was considered that "*it could work, IF there would be enough time allocated for it*", as stated on one of the sticky notes. There was also a need for documenting actions taken during the pair working sessions somewhere to be used as a future reference. However, there was no clear and concise agreement about what exactly this place should be, but for example README, Freshdesk and Google Drive were proposed as possible candidates.

Another solution to tackle knowledge sharing issues and improve quality of informal documentation was a README template. It was generally considered to be a good solution, since its implementation requires relatively

low effort and its existence could potentially greatly improve the quality of README files as they did not even exist in some projects. It was considered that writing README files is easily forgotten or considered too troublesome since there are no general guidelines about things to include in them or emphasis on their importance. It was considered helpful to have some template for README files, but also concerns about its contents were raised as having too much or too specific content would render the template practically useless.

Also project presentations were proposed as an alternative solution to informal documentation issues. Presentations raised some discussions concerning their actual benefit, since presenting the business case would not benefit the organization much. It was also noted that they would easily be forgotten or considered as a mandatory burden if not organized properly and therefore it was agreed that it would be better to organize them when considered useful rather than enforcing a strict policy of having one presentation every week. Also storing the presentation materials raised some concerns since they would easily be outdated and forgotten in Google Drive, rendering them most likely useless. However, the effort to organize short presentations every now and then was considered to require low effort with some potential benefits and therefore they could be useful.

As a sidenote, it was also pointed out that Google Drive had become messy and hard to use due to complex and inconsistent folder structure and filenames. It was proposed that it would require some cleanup if it was to be used more. However, there was no further suggestions about the concrete actions required and its benefits were debatable as Google Drive was not considered as a major tool that is used on a daily basis.

6.3.2 Process related solutions

Initially four process related solutions were presented: *triage responsibility*, *transition model*, *alerts* and *Kanban WIP limits*. The most interesting and discussed solution out of these was the rotating triage responsibility. It was generally considered good, but there was also concerns about the actual ability of implementing it as it would require training and an overall understanding of the projects. As it was seen that triage and pair working had relatively similar concerns about time management and skills required, it was proposed that they could be combined together. Basically this would mean that the person responsible for triage on each day would also be the one participating in pair working during that day, if suitable maintenance tasks are available. Of course this would not be strictly enforced as triaging could also be done remotely or some maintenance tasks would be more beneficial to complete

with someone else, but generally the person with triage responsibility should be the first one to consider when selecting a pair for pair working.

Alerts as a solution to low quality incident reports and improving customer service by earlier incident detection also raised a lot of discussion as it has been already discussed within the company for at least a year. Most of the discussions were technical considering the possible solutions that have been proposed earlier, but also practical concerns about AMS contracts were discussed. It has been identified that some dashboard to monitor all of QOCO's internal services would be beneficial and is most likely going to be implemented sooner or later, but it was not considered as a top priority at the moment. Concerns about practical issues related to alert configuration was also presented as prior experiences show that it is usually balancing between having to deal with false alarms and missing the important ones.

The transition model also received some feedback, mostly considering its relationship with other solutions, such as the README template and alerts. It was considered a good idea that QOCO should consider at some point in the future, but it was not considered as a top priority solution since transitions from development to maintenance mode are not that frequent after all. Additionally it was pointed out that as there is no separate maintenance team, the handover between development and maintenance phase is not well defined and could be considered more of a checklist that the project is ready for the maintenance phase, rather than an official handover. This could be implemented as a checklist of important aspects such as documentation and necessary alerts, but anything more official than that was considered unnecessary.

The last solution for process improvements was WIP limits suggested specifically for Airline 1 development team to tackle the issue of multitasking. It did not receive much feedback, but there were comments reflecting previous experiences on enforcing strict WIP limits. Based on these experiences, it was considered that WIP limits do not actually solve the initial problem, they just hide it by moving tickets back and forth on the Kanban board, which would mean more unnecessary work and most likely decreased willingness to keep the board up to date. As an alternative solution to multitasking issues of Airline 1 development team, an emphasis on ticket quality rather than strict WIP limits was suggested as many of the parallel streams of work will become blocked at some point due to missing requirements or customer feedback. Therefore focusing on ticket quality would be a better solution for multitasking since it would most likely result in fewer blocked tickets, eventually leading to fewer active parallel streams.

6.3.3 Selected solutions for implementation

At the beginning of the workshop it was agreed that depending on the solutions selected, roughly two to four solutions could be selected for the implementation phase. After carefully evaluating each solution, the workshop was concluded by selecting *pair working combined with rotating triage responsibility, README template* and *project presentations* as solutions to be tested during the implementation phase. Pair working and triage responsibility were considered as generally good solutions despite the concerns about increased total effort. They were considered worth giving a try as they would focus on identified primary goals and therefore possibly solve some of the most significant challenges. Pair working and triage responsibility could be considered as the main solution, but it was additionally supported with the README template and project presentations. The README template was selected for the implementation due to its low effort requirement combined with relatively high expected benefits. It would also directly address the primary goal G4 by improving the quality of informal documentation. The project presentations would also address the same goal, but they would also serve as a background knowledge required to effectively triage maintenance issues. Storing the presentation materials in Google Drive could possibly improve the amount of informal documentation, but it was considered more of a nice addition rather than an actual realistic benefit. Project presentations were also considered to require relatively low effort, but the expected benefits were not that significant either.

Rest of the solution proposals were not selected for the implementation phase, although some of them were considered as nice ideas to keep in mind. First of all, configuring alerts was not considered a suitable solution since it would require significant investment of resources before yielding any results. Also the requirements and relationship with AMS contracts was not well defined and therefore it was considered to be unsuitable to conduct in the scope of this study. However, it will most likely be implemented in some form in the near future. Also the transition model was considered unnecessary as it was likely that there would not be any transitions during the period of this study and therefore the expected return of invested resources was considered quite low. Despite this, it was considered an idea that QOCO needs to focus on in the future as there was no structured transition model defined for projects entering the maintenance mode. Also enforcing WIP limits did not receive any positive feedback based on previous experiences and it was not expected to solve the actual problems and therefore it was rejected without further discussion or any plans to implement it in the future.

Chapter 7

Implementation

Three solutions out of all proposals were selected for the implementation phase: pair working combined with rotating triage responsibility, project presentations and a README template. Initial considerations about practicalities were discussed with the COO and CCO to assign responsibilities and agree about the practical implementation of each solution. This chapter presents these initial implementation plans together with adjustments done during the implementation phase.

7.1 Triage responsibility and pair working

Rotating triage responsibility combined with pair working was considered the main solution and it also required the most preparations as it required careful planning to succeed. The very first preparation for the triage process was to double the amount of Freshdesk licenses from five to ten. This allows sufficient amount of people to participate in the triage process, although five of the licenses were assigned for part-time employees. After increasing the amount of licenses, a need for guidelines for both Freshdesk triage principles and pair working practices was quickly identified as there was no structured instructions on how Freshdesk was actually being used and how it was meant to be used. Previously the practices had been tacit knowledge and based on a mutual agreement between Freshdesk users. However, as the amount of Freshdesk licenses was doubled, this was no longer scalable and there was a need for explicit instructions. It was agreed that it would be a responsibility of the COO, as he had a clear understanding of the Freshdesk process and its relationship with billing for example. The initial instructions were also reviewed by other employees and added as a part of QOCO playbook, which contains general instructions about organizational practices.

In addition to the Freshdesk usage instructions, it was also noted that a pair working guideline is required as everyone intended to participate in pair working was not familiar with detailed pair working practices beforehand. This was assigned as a responsibility of the researcher as he was already familiar with pair programming literature. It was also noted that pair programming is a well known practice and there are plenty of good guidelines available online, which could be utilized as a reference material. After a brief review on available materials, a GitHub gist by Sarrafieh (2017) was selected as a template to be customized for QOCO's purposes.

It was also identified that writing instructions about the usage of Freshdesk and pair working was not enough to succeed in adoption of new practices. As the plan was to change the person responsible for triage on a daily basis, selecting the responsible for each day required a well planned procedure. To reduce change resistance and encourage independent participation, it was considered a good solution to let employees reserve their triage turns with Google calendar. As Google calendar was already used for vacation planning and other daily activities, it was a natural tool to be used for triage turns as well. In addition to the calendar, also a dedicated Slack channel was created to discuss about triage related matters and gather feedback. Another convenient feature with a dedicated channel was an integration to Google calendar so that it would automatically report who is in charge of triage for the day and additionally report a summary about oncoming week on Monday mornings. In addition to these reminders, it was noted that there should still be someone overseeing the process and ensuring that each day actually has a responsible assigned. This was agreed to be responsibility of the researcher and he would check every Wednesday that next week's turns are assigned, and if they are not, remind people about it.

Shortly after the initial launch of the triage process, concerns about noticing tickets were raised. It was considered that it would be the easiest if Freshdesk would notify the daily responsible automatically about newly created tickets so that the responsible would not need to constantly check for updates. As Freshdesk did not support this functionality out of the box, a separate Slack channel for ticket notifications was created. The idea was that at the beginning of the shift, the new responsible joins the channel and reviews the tickets that have been created after the last responsible left the channel. Then Freshdesk would continue to post notifications about newly created tickets to the channel during the day and the responsible does not need to check Freshdesk itself frequently. Additionally as Slack supports a wide range of emoji reactions to messages, it was also agreed that the daily responsible should put a check mark on ticket notifications after they have been handled. This way it is easier for the next person to see which tickets

are still waiting for triaging.

It was also evident already during the first days of the triage process that SLA limits and definitions for different priority levels required clarification. These clarifications were added to QOCO playbook together with other triage guidelines. In addition to that, also a response template was added to Freshdesk describing resolution times for different priority levels. Its purpose was to remind the customers about agreed resolution times, since it was considered that they were partly unclear. The intended use for the response template was to be automatically added to the first response on new tickets and it could be left out in consecutive messages.

7.2 README template

README template did not require as much preparations as its use case was much more straightforward. The first version of the template was created to GitHub by the researcher after studying the literature and several example templates available online. There was no single template that would have acted as an initial starting point to be adapted to QOCO's context as quite many of the example templates were aimed for public open source projects rather than acting as an informal internal project documentation for some company. Therefore it was decided that the template is created from scratch based on general headlines from the literature and open source templates. The first version of the template is presented in appendix B. As said, it is the first version and it is meant to be updated if there are mistakes or things to be added to it later on. The template was then explained in a company wide Slack channel and its usage was encouraged in projects under active development.

In addition to the template itself, it was also considered useful to create an example use case of it to some recent project. There was one project that had recently entered the maintenance phase and was considered a useful example case. Therefore it was agreed that the researcher will update its README to match the newly created template. This was also a good evaluation for the template as using it in an actual case could point out some missing features that could not be noticed while creating dummy content.

7.3 Project presentations

There were a few possible preparations that could have been done to decrease the effort required to prepare the project presentations. At first it was discussed that some slide show template could potentially lower the barrier of preparing presentations as it would have all the necessary points already listed. However it was also pointed out that the same effect could be achieved with a plain checklist as it could be difficult to create a useful slide show template to serve various use cases. It was agreed that plain checklists for different use cases would be enough as a starting point and slide show templates could be added later on, if considered necessary. The checklist was added to QOCO playbook so that it would be easily accessible and updatable by everyone. In addition a shared folder in Google Drive was created to store the presentation materials.

There was also an idea about having presentations in different phases of a project and therefore it was agreed that it would be beneficial to try out presentations with projects in different phases to gather feedback about which presentations were considered useful and which not. The initial plan was to have at least four presentations: the first production deployment of one project, the newest features of a SaaS project, the business case of a relatively new project and the initial plan of an upcoming project. To ease feedback gathering, a simple feedback form was created with Google Forms with a simple scale from zero to five as the only mandatory question and a voluntary open comment field.

Chapter 8

Evaluation

This chapter describes the results of taking the selected solutions into use. First the changes in selected metrics (see section 5.1) are presented, which is then followed by insights from the third workshop.

8.1 Changes in metrics

The effectiveness of the solutions were evaluated by measuring the values of predefined metrics after the implementation phase using exactly the same methods as in initial state analysis. The selected metrics addressed three different aspects: maintenance knowledge, maintenance process and multitasking. Next the changes in the selected metrics are presented by comparing them to the initial values. Further implications and analysis on the goals are presented in section 9.1.

8.1.1 Maintenance knowledge metrics

Maintenance knowledge and especially sharing of it was measured with three metrics: *average projects per person*, *average persons per project* and *significance of knowledge sharing barriers*. The measurement was based on knowledge sharing survey initially conducted during the initial state analysis and then repeated with the same target group after the implementation phase. To ensure validity of the results, it was confirmed that exactly the same persons responded to the survey on both times, which is necessary since the population is so small and therefore even slightly changing the sample between measurements would render the end results invalid.

Average projects per person

The first metric, average projects per persons on different skill and experience levels, represents the average diversity of an individual's skill set. Special interest was focused on the executives compared to other experience levels since due to QOCO's history, the gap between the executives and other employees was initially quite significant. The results after the implementation phase are presented in table 8.1 together with the change compared to the initial measurement.

	Junior	**Senior**	**Executive**
5	2.0 (+1.0)	1.0 (+0.0)	8.3 (-1.0)
3+	5.5 (+0.0)	3.0 (+0.0)	14.5 (+2.7)
1+	15.3 (+0.8)	10.0 (+0.2)	22.5 (-0.3)

Table 8.1: Average amount of projects at skill level after the implementation

Judging by the table it is evident that junior employees have gained the most knowledge during the implementation phase, which can be seen as a 1.0 increase in level 5 knowledge and 0.8 increase in overall knowledge at 1+ level. A significant improvement on the most detailed level 5 knowledge is mostly explained by previous level 4 knowledge that has now become level 5 due to increased confidence and overall experience. The increase has occurred mainly on older projects that have traditionally been in the hands of the executives, but now it seems that knowledge has been shared to juniors as well. The improvement on overall knowledge at 1+ level is partly explained by natural learning during the implementation phase, but additionally project presentations have clearly had a positive impact, since the improvements have been larger on projects that were presented during the implementation phase.

The knowledge of the seniors on the other hand has not changed much during the implementation phase. The only difference is on the overall knowledge at 1+ level with an improvement of 0.2, which could be mostly explained by natural learning. Another explanation is also that some knowledge has been lost while some has been gained and therefore the overall change appears as zero. Having a more detailed look on individual answers shows that this is the case as most of the changes are between levels 1 and 3 and the increases and decreases are approximately evenly balanced.

Lastly, the executives that have traditionally had the most knowledge at QOCO have managed to maintain their position, although there is a decrease of 1.0 in level 5 knowledge. This is a result of having a high initial value, which would require continuous effort to be maintained. Also knowledge has clearly been shared to juniors and therefore there is no need to keep knowledge at level 5 on all older projects. On the other hand, the executives have also gained the most on 3+ level during the implementation phase, which could be explained by several older projects that have been raised from levels 1 and 2 to 3. This has occurred due to personnel changes on the customer side, which requires the executives to revise their knowledge in order to be able to introduce the projects to new persons. Despite revising legacy knowledge, the executives have still lost knowledge on 1+ level, which is most likely due to natural unlearning as older projects get forgotten over time if they do not require any maintenance activities.

Analyzing the most increased and decreased projects reveals that project presentations had a positive impact on overall knowledge since all of the presented projects are in top 5 of the best known projects, most of them with a significant increase in average knowledge level. Additionally one of the best improvements on average knowledge levels is on a project that has intentionally been shared recently to reduce dependability on a single developer. Therefore it can be stated that the project presentations clearly have helped to increase overall knowledge throughout the company. On the other hand, it is evident that the projects with the most lost knowledge are the ones that have been developed a while ago, but have not been recently touched. One exception to that is the Airline 1 technical operations digitalization project that was initially the second best known project, but has lost knowledge during the implementation phase, even though it has been under continuous development. The explanation for this is most likely increasing complexity and quite independent development process of Airline 1 development team, which effectively decreases the knowledge of those who are not actively participating on the development.

Average persons per project

The second metric, average persons per project on different skill levels, represents the diversity of all skills sets of the organization. Even though projects per person metric would look promising, a low average amount of persons per project would mean that the skill sets are highly focused and overlapping while leaving many projects dependent on a single person. The change in this metric is presented in figure 8.1 for easy comparison between measurements.

As expected, 4+ level knowledge has remained mostly unchanged, since

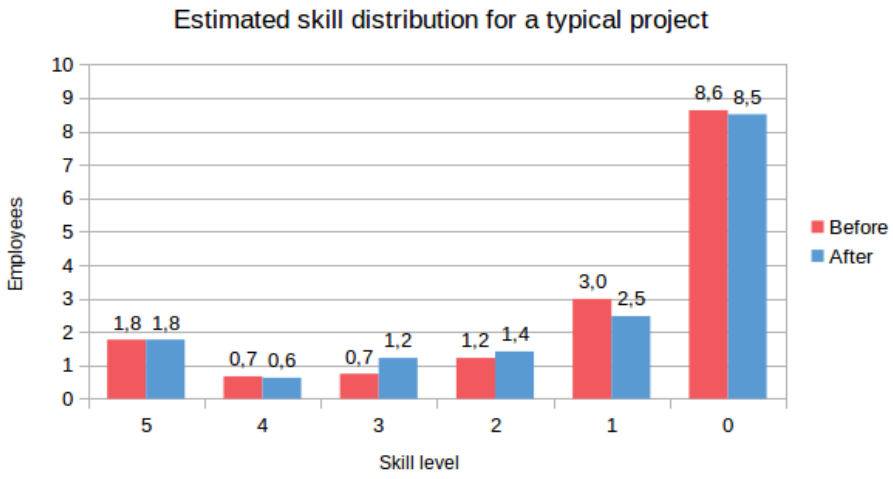

Figure 8.1: Estimated skill distribution for a typical project as of May 2019

the time frame of this study was relatively short and gaining 4+ knowledge requires quite much time and effort. However, there has been a significant improvement on level 3 knowledge, which was one of the goals for the solutions. After the implementation, there is an average increase of 0.5 persons on a typical project, which is already a significant improvement on the previous state. Another positive fact on this metric is that the largest decrease has been on level 1, which means that knowledge of the business case has been successfully updated to an ability to do minor tasks. Therefore it seems like the learning has occurred on various different projects, rather than being focused on a small group of projects.

Knowledge sharing barriers

The third knowledge sharing metric, knowledge sharing barriers, aimed to identify and measure the main challenges that slow down or even entirely prevent knowledge sharing. The most significant barriers with updated significance scores are presented below and as can be seen, the list is mostly the same as before the implementation (see section 5.2.1). Tight schedule and multitasking are still the two most significant barriers at QOCO, but their significance scores have decreased considerably. Especially tight scheduling is not as significant issue as before since its score decreased by five points from the initial score of 15, which is a huge positive development. Both multitasking and lack of informal documentation barriers decreased by one point,

which is also positive, although not as large as that of the tight schedule barrier.

The most significant knowledge sharing barriers at QOCO after the implementation:

1. Tight schedule (10)
2. Multitasking (9)
3. Lack of formal documentation (8)
4. Complex domain (7)
5. Lack of informal documentation (6)

In addition to the developments on the main focus areas, also the significance of strong code ownership decreased by three points and it has been replaced by lack of formal documentation on the list. Lack of formal documentation has one of the largest increases in significance together with working in different locations, both of which have increased their score by three points. Lack of formal documentation is especially interesting as the increase has ranked it to the top three of the most significant barriers after the implementation and therefore it should be addressed in the future.

Identifying the most significant barriers is important, but it is also important to analyze the least significant ones as well to understand the strengths of the organization. The updated list of the least significant barriers is presented below, and as could be expected, it is also quite similar to the initial measurement (see section 5.2.1). A notable fact is that generally all of the barriers on the previous list have increased their score by several points, which means that the barriers are even less significant than initially. The only exception is lack of motivation that has remained unchanged with a score of 13, which is already quite high. Therefore it does not raise much concerns, although it is still worth noting.

The least significant knowledge sharing barriers at QOCO after the implementation:

1. Physical work environment (17)
2. Difference in age (17)
3. Lack of trust (17)

4. Strong organizational structure (16)

5. Lack of motivation (13)

Despite the positive development on the top five list, the largest difference was on individual communication skills with an increase of nine points on its score. This was a bit surprising as the development was so significant compared to other barriers. There was no single explanation for this, but it was thought to be caused by generally improved maintenance process, which simultaneously improves the overall communication.

Another interesting point on the knowledge sharing barriers is the difference in experience levels. It clearly divides opinions since it has increased its score by two on most significant and four on the least significant barriers. By analyzing the survey responses in more detail, the explanation for this development is the difference between junior and senior employees. Two seniors selected "strongly disagree", whereas three juniors selected "agree" and the executives were inconclusive between "agree" and "disagree". Therefore it can be argued that the seniors do not consider difference in experience levels as significant barrier as juniors, which has many possible implications and explanations. However, based on this dataset no strong implications one way or another should be made and it should be kept in mind that it is still quite far from the most significant barriers in terms of significance score.

8.1.2 Maintenance process metrics

The process itself was addressed with three metrics as well: *tickets from unofficial sources*, *ratio of alerts* and *average resolution time*. The data source for these metrics was the data export from Freshdesk consisting of tickets created between January and April. This period was then compared to previous export, which included the whole year of 2018.

First of all, the tickets from unofficial sources or outside the official workflow are represented by reporter domain. The tickets that are reported through informal channels are forwarded to Freshdesk by QOCO's own employees, meaning that tickets from QOCO's domain are from unofficial sources in most cases. Reporter domains after the implementation phase are presented in figure 8.2 and as can be seen, the ratio of internally reported tickets has actually increased to 19 % from the initial 10 %. The increase is also reflecting to Airline 1 domain, which has decreased by 10 percentage points after the implementation. The proportions of other domains remain mostly unchanged with a one percentage point increase in the third party vendor domain.

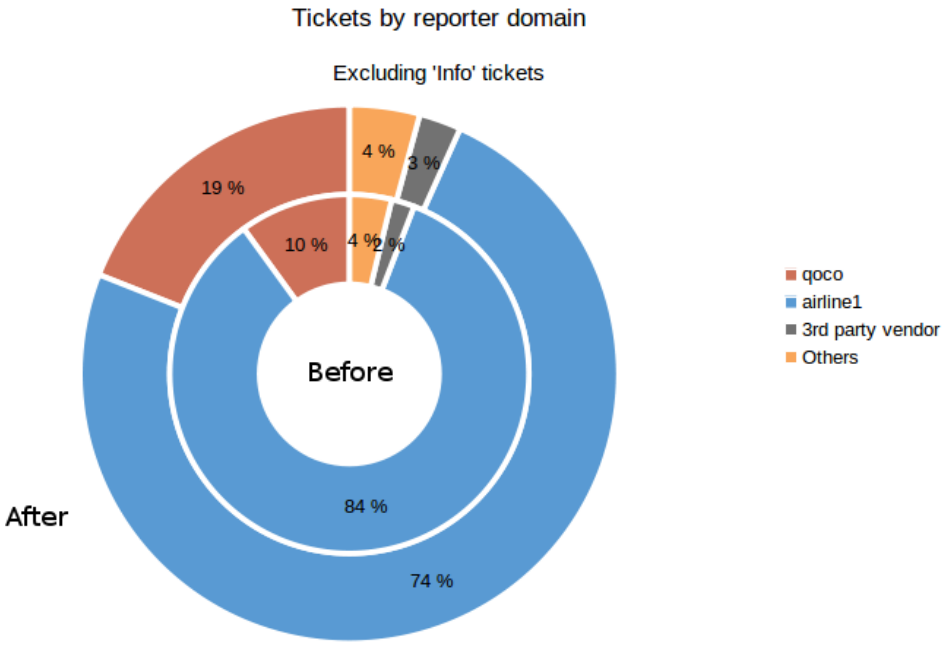

Figure 8.2: Tickets by reporter domain after the implementation phase

The second metric, *ratio of alerts*, was also interesting since it represents the automatic detection of incidents before the customer needs to report them and therefore results in a better customer experience, but also improves the quality of incoming tickets. The distribution of ticket types after the implementation phase is presented in figure 8.3. From the graph it is evident that the ratio of alerts has decreased significantly from 7 % to 2 % during the implementation phase. Digging deeper into the ticket data reveals that most of the alerts in 2018 have been about network connectivity of a single application, which has not alerted anymore between January and April. This explains the decrease in the alert ratio, but it is also worth noting that it means that there has not been many other alerts as well, which could be a point of improvement in the future.

Another notable finding on the new distribution is an increase in request and problem tickets, although an increase in problem tickets is most likely explained by a decrease in incident tickets since they are quite close to each other as concepts and the total ratio of problems and incidents has not changed significantly. The reason for the increase in request tickets did not have any single explanation, but some part of the increase could be caused by a single application requiring a bit more actions than during 2018.

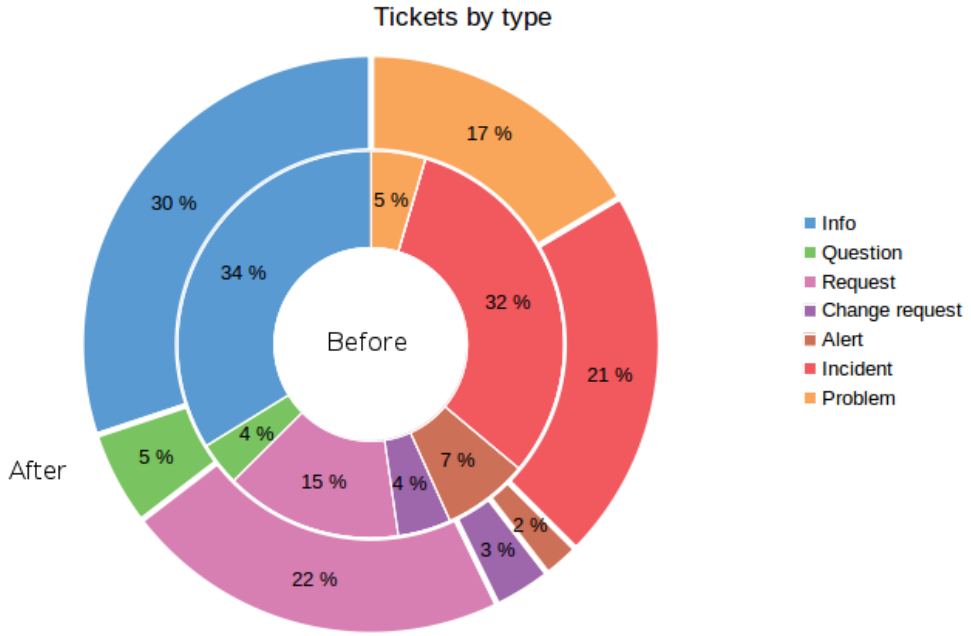

Figure 8.3: Ticket types after the implementation, excluding tickets without a type

The last process metric was average resolution time as it usually directly represents customer satisfaction and maintenance process efficiency in general. The monthly average resolution time together with the total ticket count is presented in figure 8.4. The resolution time is again plotted only for low priority tickets as they still make up 94 % of total tickets and represent the main process quite well. As there was only a few medium and high priority tickets, it was not meaningful to calculate the monthly average for them.

Judging from the graph it is evident that the average resolution time for low priority tickets has been continuing its downtrend also during the implementation phase. It is also worth noting that the average amount of tickets created during the implementation was a record high of 70 tickets per month. This most likely means that an increasing amount of the tickets that were previously not visible in the official process have been reported to Freshdesk either directly by the customers or forwarded by QOCO's employees. Despite this increase, it is also notable that April marks a record low for resolution times at approximately 12 hours on average. This is a clear sign of improvement in the process and it suggests that the solutions have managed

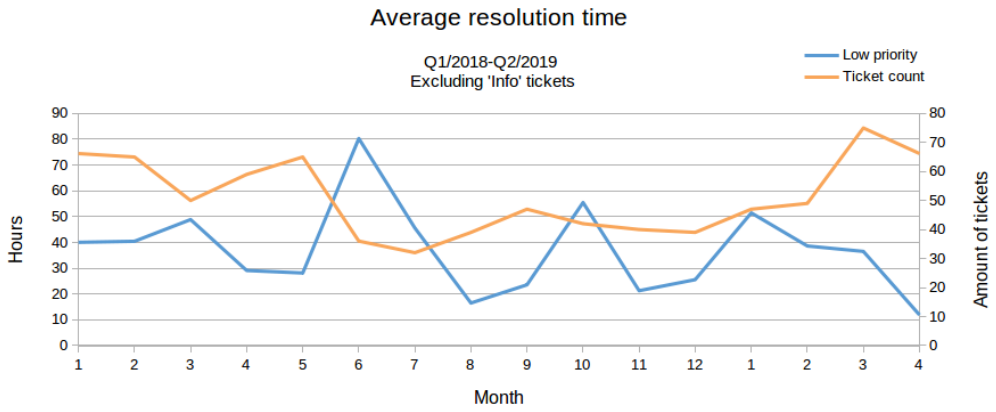

Figure 8.4: Average resolution times by month

to improve the efficiency of the maintenance process in general, although a further confirmation is still required. Another notable thing is the presence of a periodic increase in resolution times, which was already identified during the first data export. The reason for it remains unclear, but it is definitely something that should be further studied in the future.

8.1.3 Multitasking metrics

The last topic to be addressed was multitasking, which was measured by time tracking data with two metrics: *average projects per day* and *average time entries per day*. The results of the measurement after the implementation phase are presented in table 8.2 together with the change in multitasking factors compared to the initial state. As can be clearly observed from the table, multitasking has increased during the implementation phase on almost all teams, excluding MROTools team that has remained unchanged. Also EngineData team as a new one has quite similar multitasking levels compared to the MROTools team, which is natural since they are both product-oriented SaaS teams mainly working on a single project. What remains alarming is the fact that the Airline 1 development team had relatively high multitasking factor already before the implementation phase and it has only increased on both metrics. As already argued during the initial phase of this study, it is worrying since an increasing amount of multitasking could lead to increased stress, lower productivity and dissatisfaction at work.

	Projects / day	Time entries / day
Airline 1 consulting	1.5 (+0.4)	1.8 (+0.4)
Airline 1 development	2.0 (+0.3)	3.5 (+0.2)
Executives	2.1 (+0.3)	2.9 (+0.5)
MROTools	1.1 (+0.0)	2.1 (+0.0)
EngineData	1.1 (NEW)	1.1 (NEW)

Table 8.2: Average context switches per day after the implementation

As Airline 1 development team has a high multitasking factor, which has previously varied quite much over time, it is taken into further analysis as presented in the figure 8.5. As can be seen, the multitasking factor has remained mostly constant between 3.4 and 3.7, which is higher than during the first half of 2018, when it was mostly below 3.0. However, it is also notable that there has not been as significant spikes as experienced in July-August of 2018, which adds a little bit of positivity to the overall situation. This is still a real issue that should be addressed soon to prevent multitasking factor from increasing even further.

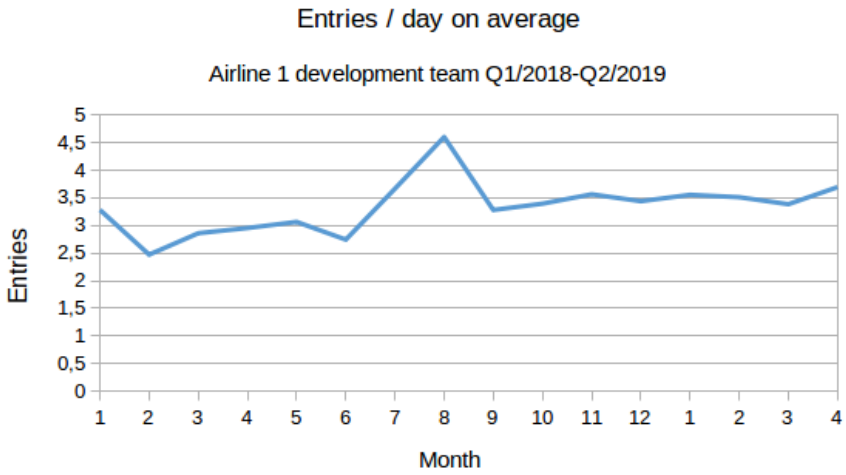

Figure 8.5: Average time entries per day for Airline 1 development team

8.2 Feedback from the third workshop

The third workshop was organized at the end of April and its purpose was to gather quantitative data about the implemented solutions and their efficiency in solving the identified challenges. In total there were six participants in the workshop from different development and consulting teams ensuring a wide variety of viewpoints. The workshop utilized the same Think-Pair-Share methodology as previous ones and addressed each of the three solutions individually.

8.2.1 Triage responsibility and pair working

The main solution in the scope of this study was considered to be rotating triage responsibility combined with pair working. It was generally complimented as a good solution to the identified challenges and its implementation was considered generally successful. The most significant benefit of it was considered to be shared responsibility, which forces to learn about other projects and share knowledge about them. As mentioned by the COO during the workshop, the overall feeling after the implementation phase was that rotating triage responsibility has indeed managed to reduce the workload of the CTO and COO by removing the stress of frequently checking Freshdesk and reacting to newly created tickets. It was mutually agreed that initially it was a bit challenging to start with the newly introduced practice as knowledge of different projects was limited and the whole process was new to all participants, but as time passed and the participants became more familiar with the support process and different projects, also the triage responsibility became more clear.

On the practical side, the triage process was considered to be working quite well. Especially Google calendar and Slack integrations were considered useful, since they managed to integrate the triage responsibility as a part of normal activities, rather than being just another additional process to be kept in mind. It was also identified that allocating triage responsibilities a week beforehand on a voluntary basis was not the best solution, since it lead to several cases where finding a responsible was difficult due to vacations for example. Also allocating responsibilities on a weekly basis was considered somewhat troublesome from the triage administrator's point of view, since he had to frequently remind people about missing allocations. In addition this, also the scalability of the process was considered, because the company aims to be growing in the future as well. Introducing an increasing number of employees to the triage process would mean less frequent turns for each

participant, which at some point is no longer practical. Also if the company continues to grow its business, also the ticket amount will most likely increase, which is no longer maintainable by a single triage responsible at some point. However, these are still challenges of the future and it was considered that the current model works well at least for now, although the future challenges are good to keep in mind.

Another challenge that was not initially considered was the amount of old tickets that are still unresolved after several months. Although this is not directly related to triaging incoming tickets, it was considered that the responsibility to check these should be added to the triage responsibilities. This would ensure that they do not get forgotten and that Freshdesk is actually representing the overall state of the maintenance process and does not get flooded with old tickets.

Pair working as a part of the triage process was also used a few times, although not nearly as much as was initially thought. When it was used it was considered helpful and fulfilling its purpose, but its implementation was still considered challenging. This was mostly because many of the incoming tickets require access to different AMOS or cloud environments, which is a highly limiting factor. As these accesses are managed by the customers and they can not be shared to the whole organization for security reasons, it is inconvenient to use pair working on tickets that can only be resolved with specific access rights.

To conclude the findings related to triage and pair working process, it was agreed that the triage process should continue after the implementation phase as well. The overall process was considered to be working well and serving its purpose, but two minor changes was added to it after the discussion. First of all, the triage turns are no longer allocated on a voluntary basis as it was considered problematic. The new practice on allocating turns is that the triage administrator assigns the responsibilities for the next month onwards by using Google calendar invites. This ensures that all days are assigned well beforehand and reduces the stress related to allocations. Secondly the responsibility to check old unresolved tickets was added for the triage responsible to ensure that Freshdesk actually represents the state of maintenance process accurately. Pair working was also considered as a good practice that could continue in the future, but it was agreed that it requires further discussions to make it meaningful and more beneficial.

8.2.2 README template

The solution to improve the quality and quantity of informal documentation was to create a general README template for QOCO that could be

used in all projects as a general guideline. The template itself was considered useful and it was added to 17 projects in total, which is a significant amount, although it does not include every single project in an active development phase. One of the main contradictions on writing README files was considered to be the fact that they should be written when they are needed the least, meaning that the README file should be written when the project is under active development even though it might not be needed then. This leads to writing README files being considered as boring and unnecessary, which leads to low quality documentation. It was also discussed that updating and maintaining these files and the template itself is somewhat challenging and easily forgotten, which decreases the benefit of them, as they might get outdated easily.

Another discussion related to the README template and the files in general raised a question about the role of a README file compared to other documentation tools that are also in use. It could be stated that the documentation practices at QOCO are still under development and therefore the relationship between different tools for documentation is still unclear, which is a topic to be addressed in the near future. It was still agreed that the README template should be used at least for now, but it was also pointed out that an even more important aspect to be improved in the near future should be unified practices of software lifecycle management, including handover practices and documentation. The README template could be included in these practices as an important part or it could be replaced with a more formal documentation, but either way, the practices should be mutually agreed and implemented.

8.2.3 Project presentations

In addition to two other solutions, project presentations were also added as a supportive solution to improve knowledge sharing. They raised a lot of discussion during the workshop and it was mutually agreed that the project presentations were considered interesting and useful. The most important benefit of them was naturally knowledge sharing as intended, but also practicing presentation skills was considered a positive side effect since they are constantly required during the customer meetings for example. It seemed that the presentation topics were selected correctly since all of them received positive feedback and it was considered interesting to learn about different projects across the team boundaries. As the presentations were arranged during the casual Friday meetings, the presentations had a wide audience and a relaxed atmosphere, which was deemed as something to be continued in the future as well.

The challenges of the project presentations were mainly around the selection of topics and scheduling. There were quite a few suggestions about future topics related to different parts of AMOS for example, which is still a relatively unknown field for many employees. On the scheduling side it was noted that a weekly presentation is maybe a bit too much, but the schedule should not be too loose either. Therefore at least a monthly cycle was suggested and generally considered as a balanced option, which still leaves the schedule open for more presentations if there is a natural spot to have one, after a release for example.

Additional challenge of short presentations was considered to be focusing since a short and concise presentation could not cover much of the details of a larger topic. Many of the presentations exceeded the loose time limit of 10 minutes even though they were not that detailed and therefore it is a challenge to be kept in mind. It is mostly tackled by a careful selection of the topics, which would also keep the preparation time reasonable, because the purpose is not to give a detailed 1-hour presentation on different projects.

To conclude the feedback from the last workshop, all of the implemented solutions were generally considered as good and they managed to address the identified challenges properly. All of the solutions will continue after this study as well with some minor changes to practicalities. They also raised some discussion about future challenges of them and the organization in general, which acts as a starting point for the next iteration of improvements.

Chapter 9

Discussion

This chapter is the analysis of the results and the research process itself. First the research outcomes as answers to the research questions are presented one by one. After summarizing the main findings and analysis on them, the validity of the research process itself is discussed by considering different threats and how they were addressed during this study. The discussion ends by suggesting topics for a more detailed research in the future.

9.1 Research outcomes

The main outcomes of this study are summarized and analyzed here. The more detailed background information on them is presented in chapters 5, 6 and 8, respectively.

9.1.1 The main problems in the initial state

The first research question aimed to identify the main challenges in the initial maintenance process. During the initial state analysis, it was evident that the transition from previous the ad hoc process to the new structured process based on Freshdesk was still ongoing. This became clear during the first workshop, when the official process diagram raised discussion about how the process was actually working, which was quite far from the official model. One of the most significant problems was considered to be informal communication channels, which were used partly because the customers preferred them, but also partly because Freshdesk had only five user licenses available at the time, which was clearly not enough for an organization of 16 people. The problem of using informal communication channels was frequent knowledge loss and unclear initial requirements, because once the email or Slack

messages finally reached the person responsible for working on the issue, some parts were quite often missing and the original request could even be misunderstood because of this. Usage of informal communication channels was also effectively hiding the maintenance tasks from the official process as they rarely resulted in tickets, which makes organizing maintenance and development tasks challenging and results in multitasking. Additionally it was also considered that Freshdesk had usability issues related to the email client and search functionality for example, which was also one of the reasons keeping the communication out of the official process.

Judging by the challenges related to informal ad hoc processes, it can be stated that QOCO is a typical small software company. The development of QOCO's maintenance process has been almost identical to the model proposed by Hasan and Chakraborty (2011) (see figure 2.1). Lacking the sufficient amount of resources to run proper structured processes easily leads to a situation where tasks are handled on an ad hoc basis by certain key personnel, which leads to further challenges while the company grows. Limited resources is a challenge for small companies (Basri and O'Connor, 2010a), which can not be entirely solved without growing the company itself. However, the difficulties caused by it should be addressed with special attention, once there are more resources available. Also the usability issues on communication tools experienced at QOCO is a well known knowledge sharing barrier described by Ghobadi and Mathiassen (2016), for example, and therefore it can be stated that the challenges identified in this study are generally in line with the literature.

Another challenge not exactly related to the process itself was knowledge sharing in general. This was mostly visible as a gap on project knowledge between the executives, namely the CTO and COO, and other employees. There were many reasons for this situation, one of them being the long history of the company as a small startup, which had recently grown rapidly. The processes were not following the growth of the company, which resulted in many tasks being handled by the CTO and COO without time to share the knowledge to others. This was already noticed at the company, but not much had been done to solve the issues as their root causes were unclear. According to the knowledge sharing survey during the initial state analysis, the most significant knowledge sharing barriers were considered to be tight schedule, multitasking and lack of informal documentation (see section 5.2.1). After a further review it was evident that multitasking was particularly a problem of the executives and the Airline 1 development team. The overall feeling of time pressure and constantly switching between different tasks effectively reduces, or even entirely prevents, knowledge sharing as it is considered that there is no extra time to discuss with colleagues about different matters, even

though it would be beneficial for the future development.

All of the knowledge sharing barriers identified in this study were already described in previous research by Ghobadi and Mathiassen (2016), Dalkir (2013) and Riege (2005) for example (see section 2.2.1), and in that sense this study supports the findings of the existing literature. Especially lack of documentation in agile methodologies is a well known issue (Ito et al., 2016)(Sommerville, 2011)(Stettina and Kroon, 2013) and as it was a significant issue at QOCO as well, this study further confirms the findings of the previous studies. Lack of documentation is also closely related to an inadequate transition model between development and maintenance phases pointed out by Ito et al. (2016) and Stettina and Kroon (2013) for example. As such model is not defined at QOCO, this study further confirms this relationship. Therefore it can be generally stated that the challenges found in this study are well in line with the existing literature.

Another point to note is the implications of multitasking on inadequate portfolio management (Vähäniitty et al., 2010). As this was not the main topic to be studied in the scope of this study, no definite conclusions can be drawn one way or another. However, as QOCO has a growing number of projects with maintenance responsibility, portfolio management is a topic that should be further addressed in the near future.

After discussing about the findings and their criticality with the executives, there was a mutual understanding that the knowledge sharing issues were the most important ones to be addressed. This was due to their effect on daily activities, but they also posed a significant threat to company's scaling plans in the future and therefore they had to be solved one way or another. Improvements on the maintenance process itself were considered also important, but still secondary compared to the knowledge sharing challenges, although knowledge sharing also improves processes and vice versa. In addition to solving the initial problems, it was also important to not introduce new ones to the process. Therefore it was considered as a loose requirement to not worsen external metrics, such as average resolution times.

9.1.2 The most suitable solutions to identified problems

The second research question focused on finding solutions to the problems identified by the first question. This was done by analyzing existing literature to gather solutions that could be applicable to the case company's context. The literature review together with the initial ideas from the first workshop yielded 11 suggestions in total that were then evaluated with the executives to filter out the ones that were not suitable to the case company's overall business strategy. Seven solution proposals made it through the initial busi-

ness critical filtering and these proposals were presented to a wider audience of employees in the second workshop to evaluate their suitability and select the most promising ones to be implemented. As a result, four solution proposals were selected to the implementation phase, although it was suggested that pair working is combined together with rotating triage responsibility. Other two solutions were README template and project presentations.

The main solution to address the issues in both knowledge sharing and the process itself was considered to be rotating triage responsibility combined with pair working. To reduce the workload of the CTO and COO on resolving maintenance tasks, the role of a triage responsible was introduced. The renewed process model with the triage responsible is presented in figure 9.1. The idea of the triage role is to check all newly reported tickets before they are assigned to a person responsible for resolving them. As QOCO is still a small company, the automatic process suggested by Hu et al. (2014) is not sufficient in its context and therefore the solution was to handle the triaging process manually although it will not scale very well in the future. The main task of the triage responsible was to ensure that Freshdesk tickets were reported with enough details and that tags, priorities and categories were used correctly. After this the ticket would be ready to be assigned for a person responsible for solving it. The person responsible for triage was rotated on a daily basis, which was considered a useful practice. First of all, this ensures that there is always someone assigned for triaging on each day, but also it is a transparent and fair way to distribute the triage workload equally to all participants. This way the challenge of decreasing motivation described by Fægri (2009) could be minimized, but of course not entirely eliminated. As said, the initial five licenses was far from enough to properly operate the renewed support model and therefore the number of licenses was increased to ten, which was already enough to cover most of the developers and consultants. This should be enough for now, but if the company continues to grow with a similar rate, it will not be long before the amount needs to be reconsidered again, depending on the organizational structure and other factors of course.

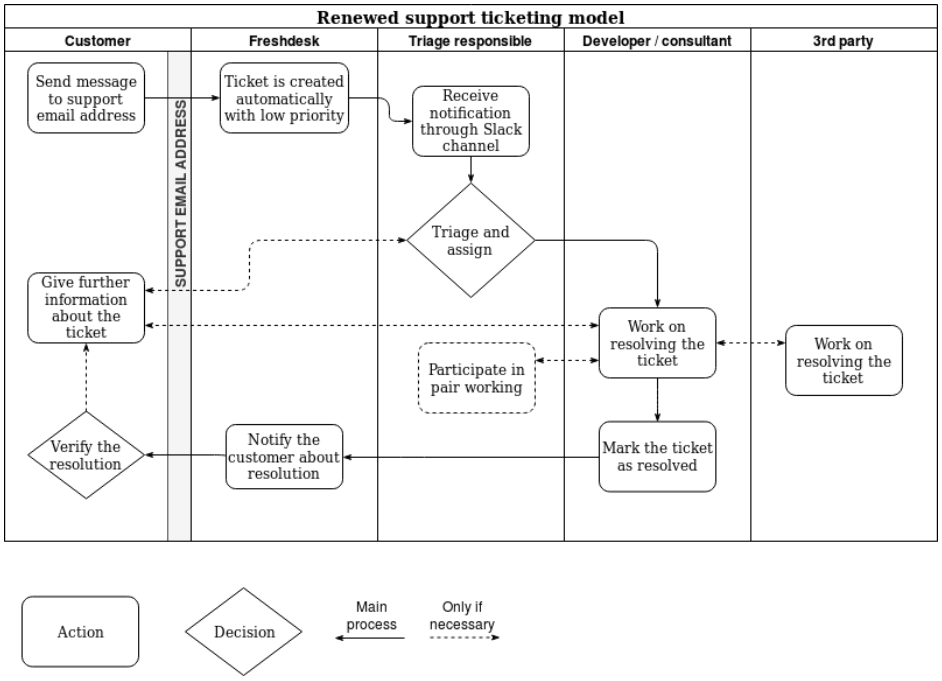

Figure 9.1: Renewed process for handling low, medium and high priority incidents

Pair working was also added as a part of the triage process to further improve knowledge sharing. The main goal of combining pair working with the triage responsibility was to share knowledge between the executives and other employees and therefore reduce dependability on key persons, which was considered a significant threat for scalability. The intended implementation of pair working was that small and well-defined maintenance tasks could be solved by working as a pair so that the person normally responsible for handling the ticket acts as a navigator and the other person acts as a driver and actually does the tasks at hand, while being instructed by the navigator. To lower the barrier of finding a pair, it was agreed that the person with the triage responsibility on that day should be the first candidate to be considered, but of course in some cases it is more suitable and meaningful to pair with someone else and therefore this was not strictly enforced. Additionally a pair working guide introducing the methodology was created to clarify the practicalities, since all of the participants were not fully familiar with the principles beforehand.

An additional solution to improve knowledge sharing and prevent knowledge loss over time was to improve the quality of informal documentation. Initially the documentation available was usually either heavy delivery doc-

umentation, delivered to the customer together with a new release, or a few most likely outdated sentences on a README file alongside the codebase. A general README file template was created to tackle this issue and lower the barrier of writing at least a few more sentences about the intended use case and development instructions while developing the project itself. The template was considered a useful solution since as discussed in the second workshop, writing a README file is easily forgotten and considered too troublesome, when there are no general guidelines about its contents easily available. The template was considered easy to implement with a potential of significant improvements and therefore it was selected for the implementation phase. The template was stored on the company's GitHub account for easy accessibility and version control.

Another solution to tackle the dependability on key persons and lower the barriers between different teams was to have project presentations on a regular basis. The main goal of the presentations was to share general knowledge about business cases without going deeply into technical details. The presentations were intended to be kept short and informal to ensure that they would not become a mandatory burden for anyone. To ensure informality, relaxed atmosphere and a wide audience, it was agreed that these presentations would be added as an occasional part of the weekly Friday meetings that were highly casual and usually without any predetermined agenda. Therefore a short 10 minute presentation would not take much time from casual chatting and could act as a "topic of the day" -like inspiration, although it was not necessarily meant to be like that.

9.1.3 Effectiveness of the selected solutions

The purpose of the third research question was to analyze how well the implemented solutions managed to address the challenges identified by RQ1. This effectiveness was measured by using various metrics presented in section 5.1. These metrics formed a base for the goals of the implementation phase, which were prioritized into three categories: primary, secondary and optional. Next each of the goals with different priority levels are evaluated based on the results of the implementation phase presented in section 8.1.

Primary goals

The primary goals that were the main focus area of this study were aiming to address the challenge of knowledge sharing, which was considered as one of the main challenges of the organization. The challenge was especially sharing knowledge between the executives and other employees since due to

company's background, the CTO and COO handled most of the maintenance tasks in the initial maintenance process. The overall goal was to share the maintenance responsibility and skills so that the workload would be more equally distributed and the executives would have more time available for new development and management tasks.

The first goal of decreasing the gap between the executives and others at 3+ level (G1) directly addressed the problem of maintenance tasks piling up on the CTO and COO. The gap was initially 6.3 projects for juniors and 8.8 for seniors, but during the implementation phase, the executives actually increased their knowledge at 3+ level, while others remained unchanged (see table 8.1). Therefore this goal was not fulfilled as the gap actually increased rather than being decreased. The reason for this is that the executives, namely the CTO and COO, had quite many projects initially on level 2, which were then increased to level 3 during this study. The common denominator for the projects with increased knowledge levels was that they were all older legacy projects that are not familiar for many employees in the company. The use of outdated technologies also introduced further challenges with knowledge sharing on legacy projects, since they were developed years ago by the CTO and employees who are no longer working at the company. Therefore the current employees are not familiar with the used technologies, which limits the possibility of knowledge sharing. This is an issue emphasized by Richardson and von Wangenheim (2007), which could become increasingly problematic in the future, however it is not a significant challenge at QOCO yet, although it is still a thing to be aware of.

Despite the increased gap on 3+ level, it is also notable that the executives actually lost knowledge on level 5, while juniors managed to increase their knowledge (see table 8.1). This is highly positive as the gap on level 5 knowledge was also large between the executives and others and these changes confirm that the solutions have actually managed to share the responsibility and the most detailed knowledge about newer projects. By analyzing the responses in a more detailed manner, it is evident that this change is indeed caused by newer projects still in active development mode. As the projects are being developed further, the complexity usually increases and keeping knowledge up to date requires continuous effort. Therefore the fact that the gap between the executives and juniors has decreased on level 5 knowledge is a sign of successful knowledge sharing between the executives and juniors. However, it is unclear whether this was caused by the implemented solutions or not, but at least they have most likely had a positive impact on this development.

The second primary goal of increasing persons per project on 3+ level (G2) reflected the diversity of the skill sets. As the time frame of this study

was limited, it was agreed that the focus should be on increasing persons with 3+ knowledge, because it represents all employees with enough knowledge to do at least minor tasks on the project. Initially the amount was 3.2 for an average project and the goal was to increase it to 4.0 during the implementation, which was quite ambitious. The amount was actually increased to 3.6 (see figure 8.1), which is only half of the intended improvement, but still a significant positive change. Therefore this goal was only partially fulfilled, but the direction of the development is definitely right and should continue in the future as well. It is still notable that this increase is mainly explained by the increase on the executives' knowledge, which was not exactly the intended outcome of this study and therefore the future development should focus more on increasing the knowledge of other employees as well.

The third and fourth primary goals were related to lowering the significance of knowledge sharing barriers. First of all, tight schedule and multitasking (G3) are quite closely related to each other, and they were initially considered as the main barriers for sharing maintenance knowledge. The most significant effect was on tight schedule as its significance score decreased by 5 (see section 8.1.1), although it was still considered as the most significant barrier. The effect on multitasking was not that strong, but it also managed to decrease its score, which fulfills G3. However, multitasking still remains a challenge as all teams have actually increased their multitasking factor during the study (see table 8.2), even though the overall feeling was that the triage process has decreased the interruptions and multitasking. The situation is especially worrying on the Airline 1 development team, which has a significantly high multitasking factor compared to other teams. This is a challenging managerial issue, which should be addressed soon to prevent it from getting even worse.

As described previously, multitasking could be a sign of inadequate portfolio management (Vähäniitty et al., 2010), but its root cause in QOCO's context was not extensively studied in the scope of this study. Tight scheduling on the other hand is a typical issue of small companies with limited resources (Kajko-Mattsson and Nyfjord, 2009), which is in line with the fact that it remains the most significant knowledge sharing barrier also at QOCO even after this study. Since this is the case, further improvements on the situation require a more detailed analysis on the root causes of both multitasking and scheduling issues as they are closely related to each other.

What comes to lack of informal documentation as a knowledge sharing barrier (G4), it was initially considered as the third most significant barrier, which was then addressed directly by implementing a README template. The goal itself was fulfilled as the significance of the barrier was lowered, but an even more interesting development was an increased demand for formal

documentation, which was a bit surprising. One explanation for this could be that actually the practice of triaging has raised a demand for documentation, which has been missing in many cases. Additionally there was a concern about managing multiple parallel documentations, which was discussed during the third workshop. Informal documentation with the README template could be seen as the first step towards more structured documentation practices as the documentation has mostly been nonexistent previously. Therefore the development on informal documentation is definitely good as it encourages discussion on documentation practices in a wider context as well.

Lack of documentation is a well known challenge of agile methodologies and it is usually solved by relying on informal documentation and tacit knowledge (Basri and O'Connor, 2011)(Ersoy and Mahdy, 2015)(Stettina and Kroon, 2013). However as it increases the risk of knowledge loss, which has been evident at QOCO as well, it is a positive development that this study has managed to raise discussion about documentation practices in general. Based on this study, it would be beneficial to include some light documentation to the development process, which is emphasized by Stettina and Kroon (2013) for example, as it would ease the transition between development and maintenance phases. However as noted in the third workshop as well, writing documentation to no one in particular is not especially motivating and decreases the documentation quality, which is also noted by Stettina and Kroon (2013). This an important challenge to be carefully considered while developing better documentation practices for QOCO in the near future.

Secondary goals

The three secondary goals were related to the process itself by addressing internally reported tickets, resolution times and the ratio of alerts. They represent the effectiveness of the solutions from process improvement point of view, which was not considered as critical as the primary challenges related to knowledge sharing, but still highly valuable. The changes in underlying metrics for these goals have been presented in section 8.1.2.

First of all, there was a goal to increase the ratio of alerts from initial 7 % to 10 % (G5) since it would reflect better customer service and an improved quality of incoming tickets. Distribution of ticket types after the implementation phase is presented in figure 8.3 and based on that graph, it is evident that this goal was not fulfilled. The ratio of alerts between January and April was only 2 %, which marks a significant drop from the initial ratio of 7 %. Also the absolute number of alerts decreased by 67 % compared to a 4-month average of 2018 and therefore the decrease in ratio can not be

explained by an increased total amount of tickets. One explanatory factor that mostly explains the decrease is the fact that most alerts in 2018 were caused by network connectivity issues on a single application. This has been apparently fixed as there has not been any more alerts related to that between January and April.

Secondly, it was intended that the ratio of internally reported tickets would decrease from initial 10 % to 5 % (G6) since it would mean that more tickets are reported directly to the support email address by customers rather than being forwarded by QOCO's employees. However there was also a challenge on this goal as an increasing ratio of internally reported tickets would also mean that an increasing amount of previously hidden tasks have been reported to the official process and it seems like that this is exactly what happened. The goal itself was not fulfilled as the ratio of internally reported tickets increased to 19 %, meaning that it almost doubled from the initial value (see figure 8.2). However, also the ticket count increased significantly during the implementation phase (see figure 8.4) and therefore it most likely means that an increasing amount of tickets are being actually reported to the support email address rather than being solved outside of the Freshdesk process. This is a good thing for now, but since the ratio is as high as 19 % currently, it is a clear point of improvement in the future since approximately one out of five newly created tickets are not initially reported to the support email address by the customers.

The last process improvement goal was to solidify the average resolution time below 30 hours during the implementation phase (G7). As can be seen from the figure 8.4, the average resolution time for low priority tickets varied between 12 and 51 hours between January and April, and between 12 and 37 hours during the implementation in March and April. Therefore it can be argued that the goal is not exactly fulfilled, but it is still worth noting that the average resolution time during the implementation phase in March and April was 24 hours, which is below the goal. Also an average resolution time of 12 hours in April marks a record low of the observation period while the ticket count has significantly increased and therefore the solutions have proved their worth even though the goal was not exactly fulfilled. This development was thought to be mainly due to improved assignment practices with the triage responsibility. As initially new tickets were waiting for "someone" to take action on them, the triage process ensures that the newly reported tickets are directly assigned to the person responsible for solving them, which decreases the overall resolution time as can be observed from the data. It is notable that the time period is still too short to claim that the average resolution time has been solidified and therefore it needs to be confirmed in the future when there is more data available.

Optional goals

In addition to the primary and secondary goals, there were also two optional goals that were considered as nice additions, but not necessary the ones to be focusing on. First of all, it was considered positive if the ratio of low priority tickets was decreased from initial 94 % to 90 % (G8), since it was thought that the priorities were not used correctly in Freshdesk and some higher priority tickets could be misclassified as low priority as it was the default for new tickets. However this goal was not fulfilled since the ratio of low priority tickets remained unchanged at 94 % after the implementation phase. Therefore it is arguable that there are most likely not that much misclassifications as the process has been better structured and there has been more concern towards the correct usage of priority levels. It can also be explained by a coincidence since the time period is only four months and higher priority tickets are still quite rare, which could mean that the ratio could increase with a longer time period, but there is no data to confirm that in the scope of this thesis. Either way, as this goal was considered optional, it is not that critical to continue monitoring in the future.

The second optional goal was to keep the total amount of tickets between 40 and 50 monthly tickets (G9). This would mean that the implemented solutions do not introduce more incidents that require maintenance actions, although an increase in ticket amount could also be explained by more tickets being actually reported to the official process. This goal has definitely not been fulfilled as the average amount of tickets in March and April was approximately 70 (see figure 8.4), which exceeds the goal by a great margin. As already argued above, this is mostly due to an increasing amount of previously hidden tickets being reported to the support email address, which causes the total amount to increase even though the actual workload remains the same. This hypothesis is also confirmed by an increased ratio of internally reported tickets and a more detailed look into the source data, which confirmed that the tickets were not directly caused by the implemented solutions. It is also worth noting that keeping the ticket amount below a certain limit does not necessarily reflect the effectiveness of the development or maintenance processes, since an increasing amount of different projects and a growing business should generally result in an increase in support tasks as well. With this in mind, the exact amount of tickets is not something to worry about, although it is still good to check the underlying reason if rapid growth is experienced in the future.

Goal	Priority	Result
G1	Primary	Not fulfilled. The gap actually increased on 3+ level
G2	Primary	Partially fulfilled. The amount of persons at 3+ level increased from 3.2 to 3.6
G3	Primary	Fulfilled. The significance of tight schedule decreased by 5 points and multitasking by 1 point
G4	Primary	Fulfilled. The significance decreased by 1 point
G5	Secondary	Not fulfilled. The ratio decreased from 7 % to 2 %
G6	Secondary	Not fulfilled. The ratio increased from 10 % to 19 %
G7	Secondary	Partially fulfilled. The average of March and April was 24 hours
G8	Optional	Not fulfilled. The ratio of low priority tickets was unchanged
G9	Optional	Not fulfilled. The average of March and April was roughly 70 tickets per month

Table 9.1: Summary of the results

Implications and conclusions

To conclude the answer to the third research question, the goals and their outcomes are gathered to the table 9.1 for easier evaluation. Judging by the goals it can be stated that this study has been successful, although it is clear that there is still a lot of work to be done in the future. As three of the primary goals were either fulfilled or partially fulfilled, it can be argued that the implemented solutions seem to have successfully addressed the identified challenges and they have managed to improve the internal processes. This naturally does not imply that these solutions are universally suitable for all small software companies, but it definitely encourages further studies on the subject.

Based on this study, I would argue that the rotating triage responsibility is a viable option to be considered in small software companies that are strug-

gling to transform ad hoc maintenance processes into more structured ones. The clear benefits of it are that it will encourage knowledge sharing and communication between teams, but also generally improve the efficiency of the maintenance process by shared responsibility. As all of the issues addressed in this study were quite typical for a small software company growing rapidly (Hasan and Chakraborty, 2011), the rotating triage responsibility should be applicable to other similar companies as well. Also since most of the literature on triaging practices focuses on automatic triaging in large companies (Hu et al. (2014) for example), it would be scientifically interesting to implement a similar solution in another context to get further insight on its applicability on a more general level.

Based on these findings, combining triage responsibility with pair programming or pair working is not exactly necessary from the process point of view, although it encourages knowledge sharing and communication. This study confirms that implementing pair programming in an actual case is quite difficult. The main challenges on pair working were considered to be its resource intensity and task selection. These findings are in line with the previous research on the subject, such as Lui et al. (2010), Plonka et al. (2012) and Spohrer et al. (2013), and therefore the suitability of pair working for small and highly specialized companies is questionable.

The implications on the informal documentation highlight the contradiction on README files, since they should be written when they are needed the least. As discussed, the documentation should be added as a part of the handover practices, even if using agile methodologies (Stettina and Kroon, 2013), which should be mutually agreed by the team. A problem of motivation and creating good quality content still remains, although the template has improved the quality significantly. The documentation practices of the case company will require further iterations in the near future, but it seems like the template has started the discussion, which is a good first step of the continuous improvement.

On the process improvement point of view QOCO is also a typical small company with a high commitment on process improvement, but lacking the resources and methodology to implement it (Basri and O'Connor, 2010b). As a key takeaway from this study, the company has implemented one iteration using continuous improvement methodology and as a result, has a better understanding of its present challenges and possible methods to address them. This study managed to raise discussion about internal process improvement and existing challenges, which marks an important change in the mindset of the whole organization. The success of the process improvement in the context of this study encourages the use of similar methods in other cases as well.

9.2 Threats to validity

To be able to understand the trustworthiness of this study, it is important to understand the precautions taken in the research design and during the research process. Validity of this study directly evaluates the trustworthiness of the presented results and therefore before making any further implications, it should be addressed from different perspectives. Runeson et al. (2012, p. 71-72) divide validity into four categories: *construct validity, internal validity, external validity* and *reliability*. Each of these aspects and precautions taken to improve them are discussed below.

9.2.1 Construct validity

Construct validity refers to interpretation issues in measures under research, including interview questions and selected metrics for example (Runeson et al., 2012). This study utilized several data collection methods and therefore there are many opportunities for interpretation issues. First of all, the executives were interviewed several times during the study, but the interviews were unstructured rather than being structured and standardized as a qualitative study method. During the interviews it was ensured that both the researcher and the executives understood the topics in the same way by asking more detailed questions and ensuring that both parties had a similar understanding on the subject. It is also notable that the researcher had a long background in the company and there was already a trust relationship and common understanding of the topics before the study, which reduces the risk of misunderstandings. Even though the interviews were not audio recorder, the discussed topics were on a general level and rather than relying on single quotes from these discussions, the study relied on overall understanding. Therefore it can be argued that the risk of interpretation issues on interviews is low.

Similarly, workshops as a main data collection method for qualitative data is subject to interpretation issues. For example discussed topics could have been understood differently by different employees, but this was addressed by creating an open environment of discussion, which encourages to ask for clarifications if something is unclear. These clarifications and confirmations on discussed topics were quite common during the workshops and judging by the outcomes, it can be stated that the risk of interpretation issues was also well handled.

In addition to the workshops and interviews there was also a survey for quantitative data analysis about knowledge sharing. The risk of interpreta-

tion issues on survey is much higher than in previous methods since it was answered individually, although there was a possibility to ask for clarifications if questions were unclear. However, this does not entirely remove the risk of misunderstandings and different interpretations and it is possible that the respondents understood for example the difference between skill levels differently. To lower this risk, the survey was used to measure the initial state and the outcome after the study. By keeping the survey exactly the same between the measurements and analyzing the overall change rather than each individual response separately, the risk of misinterpretations does not have that large impact on validity of the results. It is still worth noting that the population of the survey was very small, which increases the impact of individual misinterpretations.

One additional data collection method used was analyzing data exports from ticket and time tracking systems. As the data export analysis relies on selected metrics, it is important to consider how well these metrics actually represent the studied topic. This was addressed by evaluating 12 software maintenance metrics from literature and 10 custom defined metrics tailored for the case company's context. The metrics were analyzed together with the executives, which resulted in selecting seven most suitable ones to be used. Because of this careful evaluation, it can be argued that the selected metrics most likely represent the studied topic quite well and therefore the overall construct validity is high.

9.2.2 Internal validity

Internal validity refers to explaining causality and analyzing the factors outside this study that could have affected the results (Runeson et al., 2012). What comes to this study and its results, it can not be explicitly stated that the results occurred purely because of the actions taken in the scope of this study. Because there was no control group that could be used as a reference point, the results could be entirely caused by external factors and not this particular study. To tackle this issue this study utilized multiple data sources for triangulation and did not rely on a single data source on the implications. Also the results and their interpretations were discussed together with the executives to find possible alternative explanations for them. This reduces the risk of external factors affecting the results, but does not entirely remove it. Therefore it can be concluded that there is no solid proof of high internal validity, but the necessary precautions have been taken to ensure that it would be as high as possible.

9.2.3 External validity

External validity analyzes the applicability of the results on a more general context (Runeson et al., 2012). It is clear that this study is extremely context dependent and a similar study in another context could yield entirely different results. Also the time period was quite short meaning that the results should not be used to draw any long term general implications, although they could be used as a reference point for other studies on the subject. More generalizable results to define a general theory would require additional studies in similar contexts. To improve repeatability of this study, the research process has been documented in detail including used methods, the context and practicalities. Therefore the study itself should be repeatable, but the results might be entirely different. It can be argued that the external validity of this study is low due to high dependency on its context, but the possible actions to improve repeatability have been taken.

9.2.4 Reliability

Reliability of a study refers to its dependence on the researcher (Runeson et al., 2012). As the researcher had worked at the case company for over a year already before starting the study, the objectivity is questionable. However, the history at the case company could also be seen as a positive aspect since driving organizational change is much easier as an already agreed member of the community, rather than being newly introduced. This most likely affects the results of this study, because organizational change is a complex outcome of social interactions after all and another researcher outside the case company, or even inside of it, could have entirely different results. Despite of this, repeatability of the study has been improved by documenting all the data collection and analysis methods in detail, although using unstructured interviews is not optimal in this sense. The implications of the data analysis during different phases of the study were also discussed with the executives to find alternative explanations, which reduces the risk of biased analysis of the researcher, although does not remove it entirely. The interview and workshop topics have been documented, together with the used Think-Pair-Share methodology, but there could still be open questions about the details of them, which reduces repeatability. Also the study itself is no longer entirely repeatable in the same organization because it has already been conducted once and resetting the context back to the initial setting is no longer possible. Therefore it can be concluded that reliability of this study is questionable, but the research design and methodology have been documented to allow repeating a similar study in a different context.

9.3 Future research

As discussed already in the previous section, this study is highly context dependent and should not be used as the only source of general implications. To get a more general understanding of software maintenance challenges in small organizations and ways to solve them, a similar study could be repeated in multiple different organizations. Then this study could act as a single data point in a larger multiple case study, which then could yield more generalizable results about the challenges and the applicability of the solutions of this study. As a wider scope for future studies, a suggestion based on this study is to further research maintenance processes in general and their relationship to development workflow, especially in the context of small organizations. This could also yield more suggestions about the best practices on how to effectively manage an increasing amount of maintenance tasks together with active development.

Another important aspect that did not fit to the scope of this study was handover practices in agile organizations, both small and large. The agile methodology usually handles mostly the management of development tasks, but does not have much attention on maintenance tasks. It is almost like the software is forgotten after it has been released, which is far from reality. As agile methodologies do not generally emphasize extensive documentation, handover from active development to maintenance is risky and potentially results in knowledge loss. This is even the case when there is no separate maintenance team and the project is handed over to the development team itself by shifting it to maintenance mode. Therefore it is suggested that agile handover practices should be studied more as it is an important phase in software development in general, but it has not been studied extensively in the agile context.

Also the suitability of pair working for small companies raised questions that were left unanswered in this study. Judging by the results it seems that pair working is inherently unsuitable for small companies as the need for resources is in contradiction with the fact that lack of resources is one of the key problems of small organizations. The case is especially difficult in highly specialized companies where also the task selection and pairing become problematic. The benefits are also more applicable for larger companies as they focus on improved knowledge sharing and code quality. Therefore a further confirmatory study could be arranged to test this hypothesis on different sized companies.

Another important aspect especially for the case company, but also in a more general context, is to have further studies on how to detect software

faults before they become an issue. The natural answer to this is by improving monitoring, but the tricky part is detecting the right events while not reporting an extensive amount of false positives. This was identified as an interesting and important topic during the initial interviews of this study, but it was not advanced any further due to vague definitions of responsibilities and the limited time period. Therefore it remains to be solved in the case company, but it would also be interesting to research further in other cases as well.

What comes to the specific context of the case company, I would suggest that the suitability of different agile methodologies in a highly regulated field of aviation industry could be further evaluated. Also the recent development towards implementing different standards to gain competitive advantage could be another topic for further research since highly standardized processes are not in line with the agile ideology and the literature argues that small organizations do not generally like to implement standards. Therefore their implementation needs to be carefully planned and there could be another research assignment on a topic related to this. In addition to entirely new topics, it is also worth noting that this study was only the first iteration of continuous process improvement that could be further implemented in the future, since without continuous improvement, the outcome of this study will most likely also face issues after a while.

Chapter 10

Conclusions

The starting point of this thesis was the case company with an overall feeling that the maintenance process was not working as efficiently as it should be. The company had grown rapidly during the past year almost doubling its size, which introduced scalability issues on previous ad hoc processes that had been the dominant form of maintenance before adopting a more structured approach in 2017. Despite this, the process still heavily relied on key personnel, namely the CTO and COO, who both have a long background in the company. The main challenge of the maintenance process was identified to be knowledge sharing, which caused most of the tasks being assigned to the CTO and COO. As this practice was not scalable and began to slow down new development, it was considered the main challenge to be addressed by this thesis. In addition to this, there was also a significant amount of maintenance work occurring outside the official process in different informal channels including email and instant messaging applications, which caused a further difficulty to sharing knowledge on different tasks.

The approach to solve this problem was to evaluate different solutions together with the consultants and developers to encourage participation and co-creation of solutions. As a result of the workshops, it was agreed that three solutions would be implemented to improve knowledge sharing and the maintenance process in general. A rotating triage responsibility combined with pair working was considered to be the main solution, which was supported by project presentations during casual Friday meetings and a README template for better quality informal documentation. The triage responsibility was assigned to a group of ten persons, which covers most of the organization, although not everyone. The main goal for the triage responsible was to ensure that incoming tickets are of sufficient quality and then assign it forward to the person responsible for solving it, which would improve the overall quality of the tickets and divide the workload of handling newly cre-

ated tickets more evenly.

The solutions were tested during a seven week implementation phase, after which they were evaluated using the selected metrics and a feedback workshop for data collection. Based on this evaluation, it can be stated that the solutions managed to address the identified issues, although there is still work to be done in the future. All of the solutions were decided to be continued after the implementation phase with minor changes, which reflects their suitability to the case company. Especially the main solution of rotating triage responsibility combined with pair working worked quite well and could be applicable to other similar companies as well.

In addition to the solutions, a key takeaway for the case company is also the methodology for iterative continuous improvement. As there is still a lot of work to be done, this study merely acts as the first iteration of improvements by raising discussion about various challenges and introducing the methodology of monitoring existing processes and improving them in an iterative manner. The remaining challenges should be addressed in a similar way in the future to ensure that the well started process does not end up in a standstill.

Bibliography

Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. Cloud monitoring: A survey, 2013. ISSN 13891286.

Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. Kanban in software development: A systematic literature review. In *Proceedings of the 39th Euromicro Conference Series on Software Engineering and Advanced Applications*, pages 9–16. IEEE, 2013. ISBN 9780769550916. doi: 10.1109/SEAA.2013.28.

Muhammad Ovais Ahmad, Pasi Kuvaja, Markku Oivo, and Jouni Markkula. Transition of software maintenance teams from scrum to Kanban. In *Proceedings of the 49th Annual Hawaii International Conference on System Sciences*, volume March, pages 5427–5436. IEEE, 2016. ISBN 9780769556703. doi: 10.1109/HICSS.2016.670.

Sanne Akkerman and Arthur Bakker. Boundary Crossing and Boundary Objects. *Review of Educational Research*, 81(2):132–169, 2011. ISSN 08919976. doi: 10.3102/0034654311404435.

Ian Allison. Organizational factors shaping software process improvement in small-medium sized software teams: A multi-case analysis. In *Proceedings of the 7th International Conference on the Quality of Information and Communications Technology*, pages 418–423. IEEE, 2010. ISBN 9780769542416. doi: 10.1109/QUATIC.2010.81.

Jorge Aranda. Playing to the strengths of small organizations. In *Proceedings of the 1st Workshop on RE in Small Companies*, pages 141–144, 2010.

Richard Baskerville, Jan Pries-Heje, and John Venable. Soft design science methodology. In *Proceedings of the 4th international conference on design science research in information systems and technology*. ACM, 2009. ISBN 9781605584089, 1605584088. doi: 10.1145/1555619.1555631.

Shuib Bin Basri and Rory V. O'Connor. Understanding the perception of very small software companies towards the adoption of process standards. In *Proceedings of the European Conference on Process Improvement*, pages 153–164, Berlin, 2010a. Springer. ISBN 3642156657. doi: 10.1007/978-3-642-15666-3_14.

Shuib Bin Basri and Rory V. O'Connor. Organizational commitment towards software process improvement: an irish software VSEs case study. In *Proceedings of the International Symposium on Technology*, pages 1456–1461. IEEE, 2010b. ISBN 9781424467167.

Shuib Bin Basri and Rory V. O'Connor. Knowledge Management in Software Process Improvement: A case study of very small entities. In *Knowledge Engineering for Software Development Life Cycles: Support Technologies and Applications*, chapter 15, pages 273–288. Information Science Reference, Hershey, 1. edition, 2011. ISBN 9781609605094. doi: 10.4018/978-1-60960-509-4.ch015.

Dane Bertram, Amy Voida, Saul Greenberg, and Robert Walker. Communication, collaboration, and bugs: The social Nature of Issue Tracking in Small, Collocated Teams. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 291–300. ACM, 2010. ISBN 9781605587950. doi: 10.1145/1718918.1718972.

Ena Bhattacharyya. Communicative Competence Requirement in Technical Oral Presentation in Engineering Education: Stakeholder Perceptions in a Malaysian Context. *Journal of Applied Sciences*, 11(7):1291–1296, 2011.

Harry N. Jr. Boone and Deborah A. Boone. Analyzing Likert Data. *Journal of Extension*, 50(2):1–5, 2012. ISSN 1876-0821. doi: 10.1016/j.jfma.2016.04.007.

CMMI Product Team. CMMI for Development, Version 1.3: Improving Processed for Better Products and Services. *Software Engineering Institute*, 2010. ISSN CMU/SEI-2010-TR-033. doi: CMU/SEI-2010-TR-033ESC-TR-2010-033.

David Coghlan. Action research: Exploring perspectives on a philosophy of practical knowing. *Academy of Management Annals*, 5(1):53–87, 2011. ISSN 19416520. doi: 10.1080/19416520.2011.571520.

Giulio Concas, Maria Ilaria Lunesu, Michele Marchesi, and Hongyu Zhang. Simulation of software maintenance process, with and without a work-in-

process limit. *Journal of software: Evolution and Process*, 25(12):1225–1248, 2013. ISSN 20477481. doi: 10.1002/smr.1599.

Kimiz Dalkir. *Knowledge management in theory and practice*. Routledge, 2013. ISBN 9780080547367. doi: 10.4324/9780080547367.

Robert M. Davison, Maris G. Martinsons, and Ned Kock. Principles of Canonical Action Research. *Information Systems Journal*, 14(1):65–86, 2004.

K A Demir. A Survey on Challenges of Software Project Management. In *Proceedings of the 2009 Conference on Software Engineering Research and Practice*, pages 579–585, 2009. doi: 10.1088/0004-637X/693/1/1029.

Jean-Marc Desharnais and Alain April. Software maintenance productivity and maturity. In *Proceedings of the 11th International Conference on Product Focused Software*, pages 121–125. ACM, 2010. ISBN 9781450302814. doi: 10.1145/1961258.1961289.

Owen Doody and Maria Noonan. Preparing and conducting interviews to collect data. *Nurse Researcher*, 20(5):28–32, 2013. ISSN 1351-5578. doi: 10.7748/nr2013.05.20.5.28.e327.

Steve Easterbrook, Janice Singer, Margaret Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to Advanced Empirical Software Engineering*, chapter 11, pages 285–311. Springer, London, 2008. ISBN 9781848000438. doi: 10.1007/978-1-84800-044-5_11.

I. Burak Ersoy and Ahmed M. Mahdy. Agile Knowledge Sharing. *International Journal of Software Engineering*, 6(1):1–15, 2015.

Tor Erlend Fægri. Improving general knowledge in agile software organizations: Experiences with job rotation in customer support. In *Proceedings of the 2009 Agile Conference*, pages 49–56. IEEE, 2009. ISBN 9780769537689. doi: 10.1109/AGILE.2009.69.

Kaniz Fatema, Vincent C Emeakaroha, Philip D Healy, John P Morrison, and Theo Lynn. A survey of Cloud monitoring tools: Taxonomy, capabilities and objectives, 2014. ISSN 07437315.

Finder.fi. QOCO Systems Oy - Y-tunnus: 2292619-7 - Yritystiedot, taloustiedot, päättäjät & hallituksen jäsenet, 2019. URL `https://www.finder.fi/Sovellukset+ja+ohjelmistot/QOCO+Systems+Oy/Espoo/yhteystiedot/2350890`. Accessed 14.1.2019.

Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering and beyond: trends and challenges. In *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, volume 14, pages 1–9. ACM, 2014. ISBN 9781450328562. doi: 10.1016/0146-6380(83)90023-2.

Shahla Ghobadi and Lars Mathiassen. Perceived barriers to effective knowledge sharing in agile software teams. *Information Systems Journal*, 26(2): 95–125, 2016. ISSN 13652575. doi: 10.1111/isj.12053.

Supriya Gupta, Vandana Bhattacharya, and Madhumita Singha. Pair Programming "Potential Benefits and Threats". *International Journal of Advanced Computer Research*, 3(1):108–113, 2013. ISSN 22497277.

Jo E Hannay, Tore Dybå, Erik Arisholm, and Dag I.K. Sjøberg. The effectiveness of pair programming: A meta-analysis, 2009. ISSN 09505849.

Geir K Hanssen, Aiko Fallas Yamashita, Reidar Conradi, and Leon Moonen. Maintenance and agile development: Challenges, opportunities and future directions. In *Proceedings of the 2009 IEEE International Conference on Software Maintenance*, pages 487–490. IEEE, 2009. ISBN 9781424448289. doi: 10.1109/ICSM.2009.5306278.

Raza Hasan and Suranjan Chakraborty. Investigating Software Maintenance Challenges in Small Organizations. In *Proceedings of the America's Conference on Information Systems*, 2011.

Josune Hernantes, Gorka Gallardo, and Nicolás Serrano. IT Infrastructure-Monitoring Tools. *IEEE Software*, 32(4):88–93, 2015. ISSN 07407459. doi: 10.1109/MS.2015.96.

Eduardo Herranz, Ricardo Colomo-palacios, and Ricardo Colomo-palacios. Gamification as a Disruptive Factor in Software Process Improvement Initiatives Gamification as a Disruptive Factor in Software Process. *Journal of Universal Computer Science*, 20(6):885–906, 2014.

Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. Effective Bug Triage based on Historical Bug-Fix Information. In *Proceedings of the 25th International Symposium on Software Reliability Engineering*, pages 122–132. IEEE, 2014. ISBN 978-1-4799-6032-3. doi: 10.1109/ISSRE.2014.17.

Shohei Ikeda, Akinori Ihara, Raula Gaikovina KULA, and Kenichi Matsumoto. An Empirical Study of README contents for JavaScript Packages. *IEICE TRANSACTIONS on Information and Systems*, 102(2):280–288, 2019. ISSN 0916-8532. doi: 10.1587/transinf.2018EDP7071.

Kei Ito, Hironori Washizaki, and Yoshiaki Fukazawa. Handover Antipatterns. In *Proceedings of the 5th Asian Conference on Pattern Languages Programs of Programs*, 2016.

Mira Kajko-Mattsson and Jaana Nyfjord. A model of agile evolution and maintenance process. In *Proceedings of the 42nd Annual Hawaii International Conference on System Sciences*, pages 1–10, 2009. ISBN 9780769534503. doi: 10.1109/HICSS.2009.21.

Mehran Karimzadeh and Michael M Hoffman. Creating great documentation for bioinformatics software. *Briefings in Bioinformatics*, 19(4):693–699, 2017.

Mark Kasunic. Designing an Effective Survey. Technical report, Carnegie-Mellon University Pittsburg Software Engineering Institution, 2005.

Daniël Knippers. Agile Software Development and Maintainability. In *Proceedings of the 15th Twente Student Conference on IT*, Enschede, 2011. University of Twente, Faculty of Electrical Engineering, Mathematics and Computer Science.

Xabier Larrucea, Rory V. O'Connor, Ricardo Colomo-Palacios, and Claude Y Laporte. Software process improvement in very small organizations. *IEEE Software*, 33(2):85–89, 2016. ISSN 07407459. doi: 10.1109/MS.2016.42.

Jung Chieh Lee, Yih Chearng Shiue, and Chung Yang Chen. Examining the impacts of organizational culture and top management support of knowledge sharing on the success of software process improvement. *Computers in Human Behavior*, 54:462–474, 2016. ISSN 07475632. doi: 10.1016/j.chb.2015.08.030.

Guillermo E León de la Barra, De Barra, Mario B León de la Barra, and Ana M Urbina. Creative Problem Solving Workshops for Engineering Students. In *Proceedings of the 27th Annual Conference. Frontiers in Education: Teaching and Learning in an Era of Change*, volume 3, pages 1428–1430. IEEE, 1997. ISBN 0780340868.

Johan Linåker, Sardar Muhammad Sulaman, Rafael Maiani de Mello, and Martin Höst. *Guidelines for conducting surveys in software engineering v. 1.1.* Lund University, 2015. ISBN 0167-2789. doi: 10.1016/j.physd.2006.09.040.

Danielle Livy. 15 ITSM ITIL Metrics for Incident & Service Management Success, 2017. URL `https://blog.samanage.com/it-service-management/`

15-itsm-itil-metrics-for-tracking-incident-management-success/. Accessed 24.1.2019.

Kim Man Lui, Kyle Atikus Barnes, and Keith C.C. Chan. Pair Programming: Issues and Challenges. In *Agile Software Development*, chapter 7, pages 143–163. Springer, 2010. ISBN 978-3-642-12575-1. doi: 10.1007/978-3-642-12575-1_7.

Lotte S Lüscher and Marianne W Lewis. Organizational Change and Managerial Sensemaking: Working Through Paradox. *Academy of Management Journal*, 51(2):221–240, 2008. ISSN 00014273. doi: 10.1103/PhysRevLett.108.187601.

Frank Lyman. Think-Pair-Share: An expanding teaching technique. *MaaCie Cooperative News*, 1(1):1–2, 1987.

Phil Maguire, Rebecca Maguire, Philip Hyland, and Patrick Marshall. Enhancing Collaborative Learning Using Pair Programming: Who Benefits? *AISHE-J: The All Ireland Journal of Teaching & Learning in Higher Education*, 6(2):1411–14125, 2014. ISSN 20093160.

Sharan B. Merriam. Introduction to Qualitative Research. *Qualitative Research in Practice: Examples for discussion and analysis*, pages 3–17, 2002.

Deepti Mishra and Alok Mishra. Effective communication, collaboration, and coordination in eXtreme programming: Human-centric perspective in a small organization. *Human Factors and Ergonomics In Manufacturing & Service Industries*, 19(5):438–456, 2009a. ISSN 10908471. doi: 10.1002/hfm.20164.

Deepti Mishra and Alok Mishra. Software process improvement in SMEs: A comparative view. *Computer Science and Information Systems*, 6(1): 111–140, 2009b. ISSN 18200214. doi: 10.2298/CSIS0901111M.

Philipp Offermann, Olga Levina, Marten Schönherr, and Udo Bub. Outline of a design science research process. In *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, page 1, 2009. ISBN 9781605584089. doi: 10.1145/1555619.1555629.

Leon J Osterweil. Software processes are software too, revisited: An invited talk on the most influential paper of ICSE 9. In *Software Process Improvement*, pages 545–553. ACM, 2011. ISBN 9781118156667. doi: 10.1109/9781118156667.ch12.

Adrian Paschke and Elisabeth Schnappinger-Gerull. A Categorization Scheme for SLA Metrics. *Service Oriented Electronic Commerce*, 80(14): 25–40, 2006. ISSN 18632351.

Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–78, 2007. ISSN 01956108. doi: 28/5/914[pii].

Laura Plonka, Helen Sharp, and Janet Van Der Linden. Disengagement in pair programming: Does it matter? In *Proceedings of the 34th International Conference on Software Engineering*, pages 496–506, 2012. ISBN 9781467310673. doi: 10.1109/ICSE.2012.6227166.

Laura Plonka, Helen Sharp, Janet Van Der Linden, and Yvonne Dittrich. Knowledge transfer in pair programming: An in-depth analysis. *International Journal of Human Computer Studies*, 73(1):66–78, 2015. ISSN 10959300. doi: 10.1016/j.ijhcs.2014.09.001.

Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu, and David Lo. Categorizing the Content of GitHub README Files, 2018. ISSN 15737616.

Ita Richardson and Christiane Gresse von Wangenheim. Why are small software organizations different? *IEEE Software*, 24(1):18–22, 2007. ISSN 07407459. doi: 10.1109/MS.2007.12.

Andreas Riege. Three-dozen knowledge-sharing barriers managers must consider, 2005. ISSN 13673270.

Hugh Robinson, Judith Segal, and Helen Sharp. Ethnographically-informed empirical studies of software practice. *Information and Software Technology*, 49(6):540–551, jun 2007. ISSN 09505849. doi: 10.1016/j.infsof.2007.02.007.

Robin Ruefle, Ken Van Wyk, and Lana Tosic. New Zealand Security Incident Management Guide for Computer Security Incident Response Teams (CSIRTs). Technical Report May, 2013.

Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009. ISSN 13823256. doi: 10.1007/s10664-008-9102-8.

Per Runeson, Martin Höst, Austen Rainer, and Björn Regnell. *Case Study Research in Software Engineering*. Wiley, Hoboken, 1. edition, 2012. ISBN 9781118181034. doi: 10.1002/9781118181034.

Sharon Ryan and Rory V. O'Connor. Development of a team measure for tacit knowledge in software development teams. *Journal of Systems and Software*, 82(2):229–240, 2009. ISSN 01641212. doi: 10.1016/j.jss.2008.05.037.

Sharon Ryan and Rory V. O'Connor. Acquiring and Sharing Tacit Knowledge in Software Development Teams: An Empirical Study. *Information and Software Technology*, 55(9):1614–1624, 2013.

Mary Luz Sánchez-Gordón and Rory V. O'Connor. Understanding the gap between software process practices and actual practice in very small companies. *Software Quality Journal*, 24(3):549–570, 2016. ISSN 15731367. doi: 10.1007/s11219-015-9282-6.

Rouzbeh Sarrafieh. Pair programming, 2017. URL `https://gist.github.com/rouzbeh84/4bafc9fe4fe02edf506d11997c4674b0`. Accessed 9.3.2019.

Jin Shao, Hao Wei, Qianxiang Wang, and Hong Mei. A runtime model based monitoring approach for cloud. In *Proceedings of the 3rd International Conference on Cloud Computing*, pages 313–320. IEEE, 2010. ISBN 9780769541303. doi: 10.1109/CLOUD.2010.31.

Alberto Sillitti, Giancarlo Succi, Ilenia Fronza, and Jelena Vlasenko. Does Pair Programming Increase Developers Attention? *Industrial Track of ESEC/FSE2011, Szeged, Hungary*, 2011.

Riitta Smeds, Rita Lavikka, Miia Jaatinen, and Antero Hirvensalo. Interventions for the co-creation of inter-organizational business process change. In *Proceedings of the International Conference on Advances in Product Management Systems*, pages 11–18. Springer, 2015. ISBN 9783319227580. doi: 10.1007/978-3-319-22759-7_2.

Ian Sommerville. Software Evolution. In *Software Engineering*, chapter 9, pages 234–260. Addison-Wesley, 9th edition, 2011.

Kai Spohrer, Thomas Kude, Christoph T Schmidt, and Armin Heinzl. Knowledge creation in information systems development teams: the role of pair programming and peer code review. In *Proceedings of the 21st European Conference on Information Systems*, 2013. ISBN 9781479973675. doi: 10.1145/2597073.2597117.

Kai Stapel, Eric Knauss, Kurt Schneider, and Matthias Becker. Towards Understanding Communication Structure in Pair Programming. In *Proceedings of the International Conference on Agile Software Development*, pages 117–131, Berlin, 2010. Springer.

Christoph Johann Stettina and Egbert Kroon. Is there an agile handover? An empirical study of documentation and project handover practices across agile software teams. In *Proceedings of the International Conference on Engineering, Technology and Innovation & IEEE International Technology Management Conference*, pages 1–12. IEEE, 2013. ISBN 9781467373838. doi: 10.1109/ITMC.2013.7352703.

Muhammad Sulayman and Emilia Mendes. A systematic literature review of software process improvement in small and medium web companies. In *Proceedings of the International Conference on Advanced Software Engineering and Its Applications*, pages 1–8, Berlin, 2009. Springer. ISBN 3642106188. doi: 10.1007/978-3-642-10619-4_1.

Wenying Sun. *The True Cost of Pair Programming: Development of a Comprehensive Model and Test*. PhD thesis, University of Kansas, 2011.

Rebecca Tiarks. What Maintenance Programmers Really Do: An Observational Study. In *Proceedings of the Workshop Software Reengineering*, pages 36–37, 2011.

Haridimos Tsoukas. A Dialogical Approach to the Creation of New Knowledge in Organizations. *Organization Science*, 20(6):941–957, 2009. ISSN 1047-7039. doi: 10.1287/orsc.1090.0435.

Jarno. Vähäniitty, Kristian Rautiainen, and Casper Lassenius. Small software organizations need explicit project portfolio management. *IBM Journal of Research and Development*, 54(2):1:1–1:12, 2010. ISSN 0018-8646. doi: 10.1147/JRD.2009.2038747.

Maarten van Gompel, Jauco Noordzij, Reinier De Valk, and Andrea Scharnhorst. Guidelines for Software Quality version 1.1. *CLARIAH Task*, 54, 2016.

Zhining Wang and Nianxin Wang. Knowledge sharing, innovation and firm performance. *Expert Systems with Applications*, 39(10):8899–8908, 2012. ISSN 09574174. doi: 10.1016/j.eswa.2012.02.017.

Christof Wecker. Slide presentations as speech suppressors: When and why learners miss oral information. *Computers & Education*, 59(2):260–273, 2012. ISSN 03601315. doi: 10.1016/j.compedu.2012.01.013.

Daniel M Wegner. Transactive Memory: A Contemporary Analysis of the Group Mind. In *Theories of Group Behavior*, pages 185–208. Springer, New York, 1987. doi: 10.1007/978-1-4612-4634-3_9.

Laurie Williams. Pair Programming. In *Encyclopedia of Software Engineering*, volume 2. Wiley, 2010.

Stuart Wray. How pair programming really works. *IEEE Software*, 27(1): 50–55, 2010. ISSN 07407459. doi: 10.1109/MS.2009.199.

Robert K. Yin. Case study research design and methods third edition. *Applied social research methods series*, 5, 2003.

Franz Zieris and Lutz Prechelt. On knowledge transfer skill in pair programming. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '14*, pages 1–10. ACM, 2014. ISBN 9781450327749. doi: 10.1145/2652524.2652529.

Appendix A

Survey structure

This is the structure for the knowledge sharing survey conducted at the beginning of February and at the end of April. The survey was created with Google Forms.

Notation:
This is a question
This is an additional text from the survey
These are clarifications for documenting purposes

What is your role?
Select the most suitable one

- *Junior developer*
- *Senior developer*
- *Junior consultant*
- *Senior consultant*
- *Executive*

Work experience in years at QOCO?
Integer scale from 0 to 10

How familiar are you with these projects?
List of 34 projects in random order, each evaluated with a 6-point scale:

0. *I do not know the business case*
1. *I am familiar with the business case*
2. *I am familiar with the business case and general architecture*

3. *I can do some minor tasks*
4. *I can do major tasks*
5. *I know this project in detail*

Do you think following barriers are a challenge at QOCO?
Knowledge sharing barriers are things that disturb or prevent sharing of knowledge. Think especially in the context of QOCO Systems.

5-step Likert scale (*Strongly disagree, Disagree, Neither agree or disagree, Agree, Strongly agree*) was used to evaluate 20 knowledge sharing barriers in random order:

Lack of formal documentation
Lack of informal documentation
Lack of comments in code
Lack of trust
Messy and complex code
Lack of informal communication
Knowledge hoarding
Difference in experience levels
Tight schedule
Lack of motivation
Used communication tools
Multitasking
Complex domain (aviation industry)
Working in different locations
Difference in age
Individual communication skills
Strong code ownership
Physical work environment
Organization culture
Strong organizational structure

Appendix B

README template

```
# qoco-readme

### Table of Contents
- [qoco-readme](#qoco-readme)
    - [Table of Contents](#table-of-contents)
  - [Business case](#business-case)
  - [Development](#development)
    - [Prerequisites](#prerequisites)
    - [Setting up local development environment](#setting-up-local-development-environment)
    - [Building](#building)
    - [Running or Usage](#running-or-usage)
    - [Deploying](#deploying)
  - [Environments](#environments)
  - [Support](#support)
  - [Related / dependent projects](#related--dependent-projects)
  - [Known limitations](#known-limitations)


## Business case

The business case of the project is briefly described here. What does this project do? Why does it exist?

This README template is meant to improve informal documentation of projects. It is meant to be used as a reference material about
necessary information required for README. It can be modified to fit the context of different projects, but the template itself
should be kept on a general level to ensure suitability for different use cases.

## Development

### Prerequisites
A list of prerequisites for this project and instructions on how to get them.
* [A Github account](https://github.com/)
* [Markdown editor](https://code.visualstudio.com/)
* [Git](https://git-scm.com/)

### Setting up local development environment
Steps needed to setup the local development environment.

Clone repository:
‘‘‘git clone git@github.com:QOCOSystems/qoco-readme.git‘‘‘

### Building
Steps needed to build or install the project. Additional information about build tools can be presented.

This project is a simple markdown file, so no additional build steps are required.

### Running or Usage
Steps needed to run or start the project. Additional information about runtime parameters etc. can be presented.

This project can’t be run, it is meant to be copied to the root folder of a project in a default branch so that it is the first
thing that developers see when they open the repository. The template acts as a starting point and reference material to create
a project specific README and after copying it should be modified to fit the specific context.

### Deploying
Steps needed to deploy the project to dev/preprod/prod environments.

This project is hosted on GitHub and it is deployed by creating a pull request to master branch (see [Contributing](CONTRIBUTING.md))
```

```
## Environments

A list of environments this project is deployed to. Also a description of artifacts or architecture overview can be added.

| Stage | Environment | Artifacts          |
|:------|:------------|:-------------------|
| Prod  | Github      | README.md template |

## Support

Who knows something about this project? Where to find more documentation?

This template was first created as a part of master's thesis by Markus Tyrkkö.

## Related / dependent projects

List of projects that use the API provided by this project for example, or are otherwise dependent or related to this one.
Also a brief description about how they are related to this would be nice to have. An overview of architecture can also be added here.

* [Markdown Cheatsheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)
  * A cheatsheet for markdown syntax

## Known limitations

A list of known limitations or issues. This might not be necessary as issues could be tracked using other issue tracking methods,
but if there is something that should be mentioned in README already, it could be added here.
```