Aaro Salo

Improvement Suggestions for a Software Organisation on Developer Time-Tracking and Effort Estimation Practices

Metropolia University of Applied Sciences

Master of Engineering

Industrial Management

Master's Thesis

12 May 2025

Abstract

Author:	Aaro Salo
Title:	Improvement Suggestions for a Software Organisation on Developer Time-Tracking and Effort Estimation Practices
Number of Pages:	62 pages + 2 appendices
Date:	12 May 2025
Degree:	Master of Engineering
Degree Programme:	Industrial Management
Professional Major:	Data Driven Business
Supervisors:	Dr. Thomas Rohweder, Principal Lecturer Sonja Holappa, M.A., Senior Lecturer

The objective of this thesis is to build a proposal for improving the effort estimation and time-tracking practices in the case company in order to improve planning accuracy by aligning hour-reporting and planning practices. In today's technological society, companies producing software must reach exceeding levels of quality and efficiency and without proper planning of software development this is not possible in complex and grand software projects as seen in telecommunication industry.

The thesis consists of four discrete steps: current state analysis, conceptual framework development, proposal generation and validation. Current state analysis collects and discusses the current state of affairs in the company by analysing the strengths and weaknesses of current processes. Also referred to as Data 1, the information is collected through interviews and analysis of company's internal documents. In the second step, relevant literature research provides a framework to establish three categories to improve within the company: robust processes, clearly-defined roles and company culture. In the third step, Data 2 is collected through another round of interviews using the previous steps as a baseline. The output is the generation of the initial proposal to improve company processes by combining both company-specific intel and relevant academic literature. Lastly, the validation steps contains the validation of the initial proposal into the final proposal through Data 3 collection, in which the initial proposal is improved upon with relevant company informants.

The concrete output of this thesis is the list of recommendations to improve developer time-tracking and effort evaluation practices. The recommendations contain implementing the proposed process improvements within a certain department of the case company.

Keywords: time tracking, effort evaluation, hour logging, planning

Preface

It has always been my goal to continue studying after completing my technical degree. The Industrial Management programme at Metropolia University of Applied Sciences felt like a great opportunity to further develop my skills in management and leadership. The journey to this point has not been the easiest, but it has certainly been a rewarding one.

First and foremost, I would like to thank Dr. Thomas Rohweder for his continuous support and clear communication throughout the writing process. I am also grateful to Sonja Holappa for her generous insight, shared with dedication, during the thesis work. My sincere thanks go to the entire degree programme staff and fellow students for the courses, discussions, and valuable lessons along the way.

I would also like to thank my manager, Farah Salah, for enabling me to pursue these studies alongside my work.

Finally, I want to thank all my friends and family for their support – especially my wife, Tuuli. Without your understanding and encouragement, I would not have made it through.

Aaro Salo

Espoo

8 March 2025

Contents

Abbreviations

3GPP: 3rd Generation Partnership Project

APO: Area product owner

EE: effort estimation/evaluation

FOT: Feature owner team

FS: Feature screening

FS0/FS2: Checkpoints used in introducing a new feature

FS0EE: FS0 effort estimate or FS0 effort evaluation

FS2EE: FS2 effort estimate or FS2 effort evaluation

L-APO: Lead area product owner

LPO: Local product owner

PO: Product owner

RAN: Radio access network

SW: Software

SWC: Software component

XP: Extreme programming

xPO: Common term for any level of product owner (LPO, APO, L-APO, PO)

1 Introduction

Planning is a universal concept in the context of companies and work. Each process, which is aimed to produce a valuable output, requires planning in order to complete the task efficiently. Regardless of the content or the type of the work, defining a set of steps and their order to complete the task is required. Additionally, if multiple people are working on different steps, the duration and load of each step should be known in order to reserve sufficient capacity over a specific timespan. Companies need planning in order to be predictable to their customers and their shareholders. Accurate planning enables companies to compare themselves to their competitors, create roadmaps for future operations and, hence, producing decision-making capabilities in the leadership of a company.

Especially large software companies live and die by the quality and amount of the produced software. In large software projects, in which a great number of software engineer work on common code bases, accurate planning is a mandatory requirement. The competition is usually fierce so new features must be delivered in relatively tight schedules to maintain a company's competitiveness. However, simultaneously, the quality of the software must be high enough so the product being delivered to customers is of satisfactory level.

As the software of today's ever-developing technological society become more and more complex, the importance of planning becomes greater. The market is demanding better, faster and more reliable technology, to which software companies need to answer to by producing better products, at the core of which planning is. Multiple frameworks have been developed to make planning and execution processes in software more accurate. This indicates that organisations are aiming to find the best approach.

1.1 Business Context of the Case Company

The case company is a global business-to-business technology innovation leader working in the field of data networks. The company was founded in 1865 in Finland operating in over 130 countries today. Over 86 000 people work in the company globally. Their strategy revolves around digitalisation and connectivity, which are allegedly expected to have a critical role in solving some of the world's greatest challenges including stalled productivity, climate change and unequal access to opportunity. In their own words, the ambition of the company is to bring solutions to the market that aid in the digitalisation of physical industries and cities, helping them decarbonise and increase efficiency, productivity and safety.

Mobile Networks is a business unit in the field of wireless communication for mobile devices. The radio solutions, which this business unit delivers to the market, enable customers with the possibility to set up wireless networks. The purpose of the products is to provide both hardware and software which enable wireless high-speed, low-latency and reliable data transmission for millions and millions of people.

1.2 Business Challenge, Objective, and Outcome

Radio access networks (RAN) are complex pieces of hardware and embedded software which require careful specification and planning. In addition, customers – usually mobile operators – request more and more complex functionalities so they can serve their own customers better. Consequently, creating software in this market is both very competitive and challenging due to the complex nature of wireless data transmission. It is a highly-regulated field combining country legislation for radio frequency permissions and is also under heavy standardisation by 3rd Generation Partnership Project (3GPP), which is an institution mandating how the wireless connectivity has to operate.

In order to build the software for hard real-time applications in a competitive and regulated field requires dedicated hardware to carry out heavy processing and a

large headcount of software and hardware designers to realise the RAN products. Especially in a software development setting, the different components of software development need to be properly aligned. In other words, dependent software components need to be implemented in a dedicated order so they can be tested and verified to be working. In a project containing hundreds or thousands of software developers, the planned work would in an ideal situation be like clockwork. However, many parameters may cause software development to be delayed: erroneous planning, lack of resources, incorrect specifications, unidentified impact, human error, but even on a ground-level the planning may be impossible to be carried out in the first place. The organisation has realised the developer time bookkeeping is not cohesive and precise in all software teams.

The objective of this thesis is to build a proposal for improving the effort estimation and time-tracking practices in the case company in order to improve planning accuracy. The suggestions will help the software teams align their hour-reporting and planning practices, which will improve developer bookkeeping of used time resources, thus, providing better visibility for the implemented software features. This in turn will aid to achieve more accurate planning for future features, whose hour estimates may be based on historical data. The provided suggestions should, thus, align the practices within the business unit in scope and ultimately improve performance.

1.3 Scope and Outline of the Thesis Report

This thesis focuses on two dimensions of software feature planning: effort evaluation and developer time tracking.

This thesis contains seven chapters. Followed by the first chapter, which is the introduction of the thesis, Chapter 2 details the project plan containing both the project plan and used data collection methods. Chapter 3 describes the current state of feature planning in the case company including analyses of time-tracking and effort evaluation practices. Subsequently, Chapter 4 discusses relevant literature for improving planning processes and Chapter 5 provides a proposal for

the practical improvements in the case company. Finally, Chapter 6 explains the validation of the proposal including feedback and Chapter 7 recapitulates the outcome of the thesis with future steps and author’s self-evaluation.

2 Project Plan

In Chapter 1, the business context, business challenge, objective and outcome, and scope and outline of the thesis were introduced. This chapter explains the chosen research approach, then outlines the research design and finally specifies the data collection methods.

2.1 Research Approach

Research may be carried out in multiple different ways. There are two high-level approaches: pure research and applied research. The two types of research differ clearly in the context they are applied in and how they are executed. Usually, the purpose of pure research is to expand knowledge of processes in general, while disregarding the effect of practical goals, which might ultimately prevent the discovery of more general and pure ideas [1]. Instead, relatively minor attention is paid to practical applications.

Furthermore, pure research is tightly associated with an academic agenda, as this form of research is aimed primarily at an audience with an academic background. The main goal of pure research is to tackle problems and enhance our understanding across different fields. Research findings are usually shared through academic platforms such as conferences and journals. The practical impact is typically indirect, affecting practices through higher education, consulting by researchers, or non-academic publications. [2, pp. 8–9]

Applied research, on the other hand, aims to find a solution to an existing and immediate problem. Such research may be carried out by academics, practitioners or consultants and market research agencies. The main goal in applied research is to address a practical problem. In a business domain, applied research is usually employed when practical enhancements of processes are considered using often qualitative methods, which consist of, for instance, interviews [3].

In this thesis, applied research is utilised and the methods chosen are qualitative. The business challenge at hand, as detailed in Chapter 1, is a practical matter to improve ground-level software feature planning by harmonising effort evaluation and developer time-tracking practices in the case company. The outcome of this research is generated through utilising triangulated data at different stages of this study and the conceptual framework stemming from literature research.

2.2 Research Design

This thesis is divided into four parts: a current state analysis, literature review, proposal building and validation. The above order is chronological and is further depicted in Figure 1.

Source	Stage	Outcome
Data source 1 Interviews for planning personnel, team managers and developers (within Competence Area)	Current state analysis How do different geographical sites execute effort estimation and time-tracking?	Uncovering misalignments between sites and documentation of pros and cons of current planning practices
Literature on software development frameworks	Study Reading and extracting information on how different software development frameworks do planning	Conceptual framework for improved planning practices
Data source 2 Discussion with Lead Area Product Owner	Proposal Generate a proposal how to align effort estimation and time-tracking within teams and sites	Initial improvement suggestions to harmonise planning practices
Data source 3 Feedback from L-APO developers	Validation Feedback from planning personnel, developers and Competence Area project lead	Final improvement suggestions to harmonise planning practices

Figure 1: Thesis Research Design

As depicted in Figure 1, current state analysis is conducted as the first step, which uncovers the current strengths and weaknesses of the effort evaluation and time-tracking practices in the case organisation. Further information is extracted from internal company documentation and guidelines on how such processes should and should not be conducted. In addition, interviews are carried out by discussing with various stakeholders. The stakeholders are product owners on multiple levels and software developers who carry out the time-tracking in

practice. More information regarding product ownership and its meaning will be disclosed in Chapter 3.2. As the case company is a global corporation, the interviewees are chosen so that geographically all software teams are covered and, consequently, any potential misalignment between teams is found.

The second stage of the study is a literature review as shown in Figure 1. It takes place after the current state analysis and utilises the findings of the current state analysis to find solutions to the weaknesses revealed in the time-tracking and effort evaluation processes of the case company. The outcome of the second stage is a conceptual framework for improving key processes.

The penultimate stage is building the initial proposal to improve the processes. The recommendations generated are created to improve upon the key weaknesses that are found in the first and second steps. Co-creating the recommendations with the interviewees utilises the conceptual framework generated in the second stage.

The fourth and final step is the validation stage where feedback from product owners and software developers from multiple geographical sites is collected. The initial proposal will be refined and the final proposal is given.

2.3 Data Plan

This thesis uses two main sources of data: interviewing relevant personnel at the case company and internal documents and guidelines. The data sources are shown in Figure 2.

As the basis for the current state analysis, data source 1 consists of interviews with product owners and software developers on multiple geographical sites. In addition, company internal documentation is utilised in order to uncover existing guidelines for time-tracking and effort evaluation practices.

The interviewees function as informants also for data source 2. However, one key difference is also interviewing Lead product owners for proposal building to ensure

	Content	Source	Informant	Timing	Outcome
Data source 1 **Current state analysis**	• Status of current planning practices • Pros and cons	• Company internal hour tracking tools • Feature screening process overview • Interviews	Area Product Owners Software Developers (Multiple geographical sites)	Jan-Apr/2024	Summary of current status of planning practices
Data source 2 **Proposal**	Definition of proposed aligned practices	Interviews with planning personnel	Lead Area Product Owner Area Product Owners (Multiple geographical sites)	Jul-Aug/2024	Initial proposal for improvement suggestions
Data source 3 **Validation**	Validation of aligned practices	Discussion with project lead, planning personnel and developers	Lead Area Product Owner Area Product Owners Software Developers (Multiple geographical sites)	Oct-Nov/2024	Final improvement suggestions

Figure 2: Thesis Data Plan

validity of the proposal on a higher level of the software project.

As depicted in Figure 2, the final data source consists similarly of interviews with project leads, product owners and software developers. All interviews are conducted either in online calls or face-to-face meetings.

The next chapter describes the current state analysis in more detail and provides insight to time-tracking and effort evaluation practices in the case company by outlining the key strengths and weaknesses.

3 Analysis of Current Feature Planning Practices

In this chapter, the current state of time-tracking and effort evaluation practices is discussed. The practices are clearly defined and explained, and further divided into strengths and weaknesses based on interviews, company tools, and documentation. The data used in the Current State Analysis (CSA) were presented in Chapter 2.

This chapter first gives an overview of the current state based on data shown in Figure 2. Then, a general description of planning practices in a software engineering project is discussed, followed by an analysis of time-tracking and effort evaluation practices.

3.1 Overview of Data 1 Collection

The current state analysis is based on interviews with personnel from different geographical sites in a specific software component of the business group and internal documentation of the case company. Choosing one particular software component in the business group was a decision made to maintain a reasonable scope for the thesis.

As stated in Figure 1, data for the current state analysis was extracted from the company's internal documentation and interviews with relevant planning personnel within the software component. Interviews were conducted one-on-one either physically or remotely through private calls. The interviewees were chosen so that the sample of interviewed people would provide a reliable foundation for the study.

3.2 Overview of Current Organisation Structure

As shown in Figure 3, in sizeable organisations such as the case company of this thesis, the software projects are very large. The software project in question is the development of an embedded software repository that contains the code for

telecommunication hardware that is sold to customers together with the software. Customers of the case company, most commonly teleoperators, use these embedded software and hardware to provide wireless Internet access. Their customers can connect wirelessly using user equipment, such as mobile phones, in exchange for a subscription fee for the Internet service. The software is divided into multiple software components. Software components are essentially teams that are responsible for different parts of the code repository, which are conveniently split into separate entities to simplify the structure of the codebase. One example split could be uplink (UL) and downlink (DL): downlink refers to data transmission from a base station located in a cell tower to a user device, and uplink refers to data transmission in the opposite direction [4]. In this example, there would be separate DL and UL teams, and these teams would be parts of two different software components. In other words, each individual software team is practically tied to work within a specific component in the software. The software teams in the case company are located in multiple geographical sites.

The scope of the CSA – and essentially the whole thesis – concentrates on the practices within one particular software component consisting of multiple geographical sites, each with multiple software teams. The scope has been defined so that the number of applicable software teams is manageable, but also so that a meaningful sample of the software project is considered. The size of the discussed software component includes some hundreds of software developers, line organisation, and project organisation.

Time tracking is of great importance in companies that bill a customer directly according to used hours, but within software houses that develop software for sale, similar practices are not absolutely necessary. Developer time tracking may seem unnecessary if the link to company strategy is not carefully considered, and the necessity is a factor that arose also during the execution of the interviews. The main purpose of developer time tracking is to evaluate the cost of features within the software project and, even on a higher level, to evaluate the cost of a product.

Figure 4 shows how the relevant data points are utilised. The line organisation

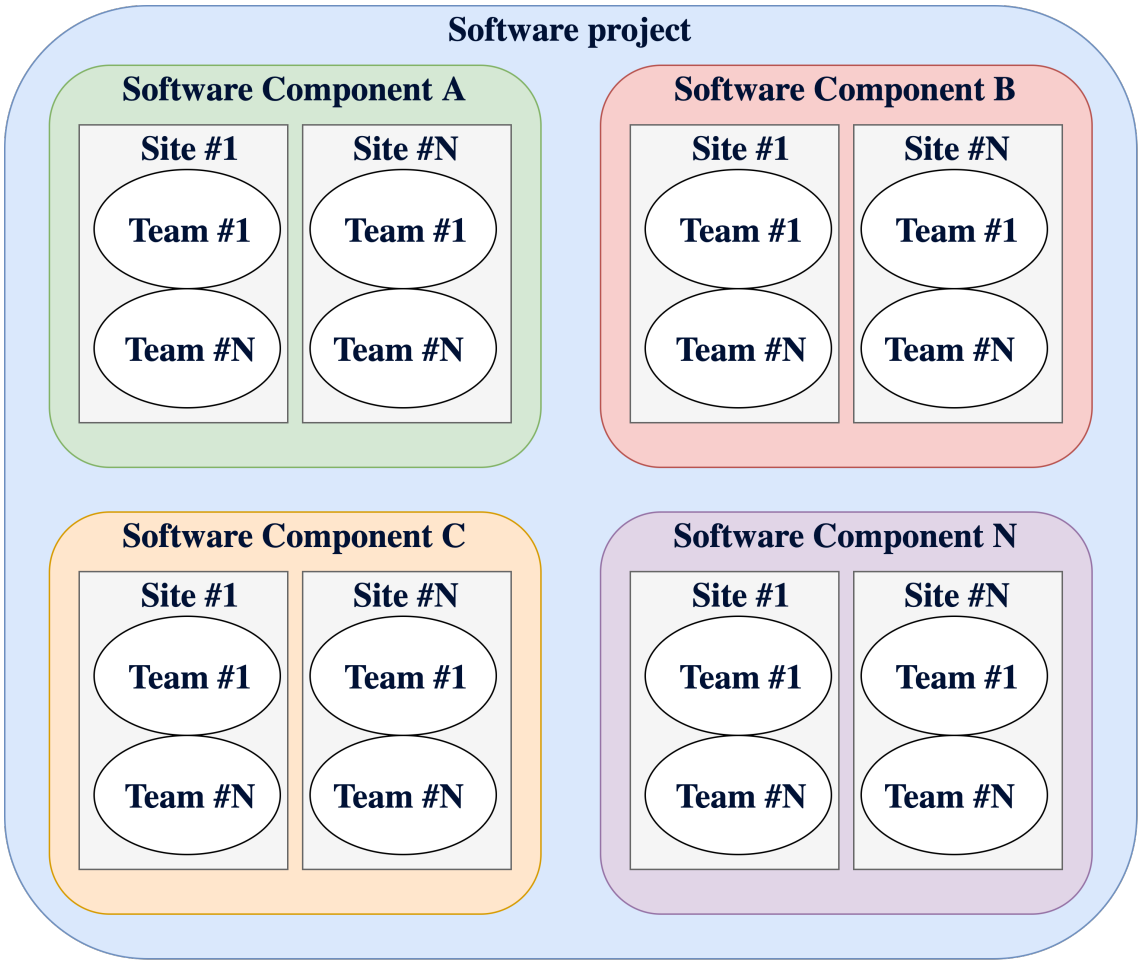

Figure 3: How different teams map between geographical sites and software components in a software project.

consists of software teams managers and administrative personnel; software teams consist of developers, local product owners, and scrum masters; and the project organisation consists of local, area, and lead area product owners and planning personnel. Local product owners are team members who represent the project organisation within software teams.

The company sets priorities for new features that will be implemented within the software. The priority depends on multiple factors, such as marketability, importance to customers, assumed performance increases, complexity, and cost. Ultimately, the meaningful metric from which the cost of a feature is derived is developer time. More specifically, it is the time that software engineers use to implement the said feature. This is, of course, a simplified view for multiple reasons, e.g. there also being maintenance and administrative costs, but this is

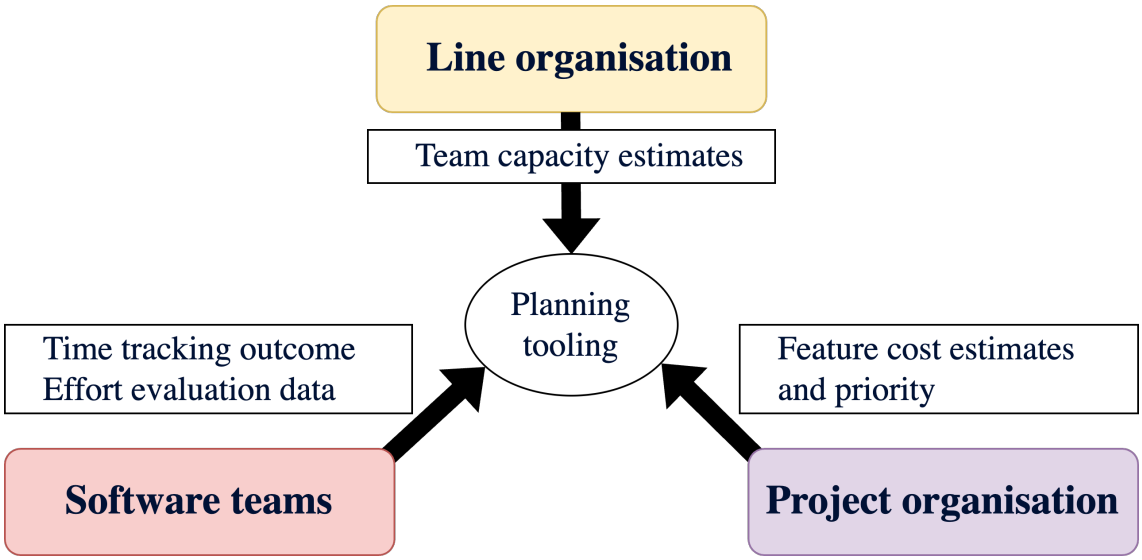

Figure 4: Structure of the planning data collected in the organisation.

one of the tools the company utilises to keep track of project costs, which can then be used to make business decisions regarding future products and profitability projections. In this tooling, only software developer time is tracked; project organisation time consumption is a static cost and not followed separately.

In Chapter 2.3, the relevant stakeholders in this study were introduced: software developers, local product owners (LPO), area product owners (APO), and the lead area product owner (L-APO) are all in key positions when considering time-tracking and effort estimation practices. Figure 5 further elaborates on how LPOs, APOs and L-APOs are mapped against geographic sites and software teams. Within the software component in the scope of this thesis, four geographical sites are considered, each with approximately six to ten software teams, which usually consist of six to ten team members with the inclusion of an LPO. An APO is usually responsible for one to three software teams, and each geographical site appoints an L-APO looking over the whole site. In addition, each software component appoints a global L-APO, who follows the work of the whole software component. The PO is responsible for multiple software components.

The interviewees for this thesis consist of the global L-APO, two APOs, and two LPOs. All four geographical sites have been covered when choosing the interviewees. The rationale in choosing the interviewees was to cover all levels of

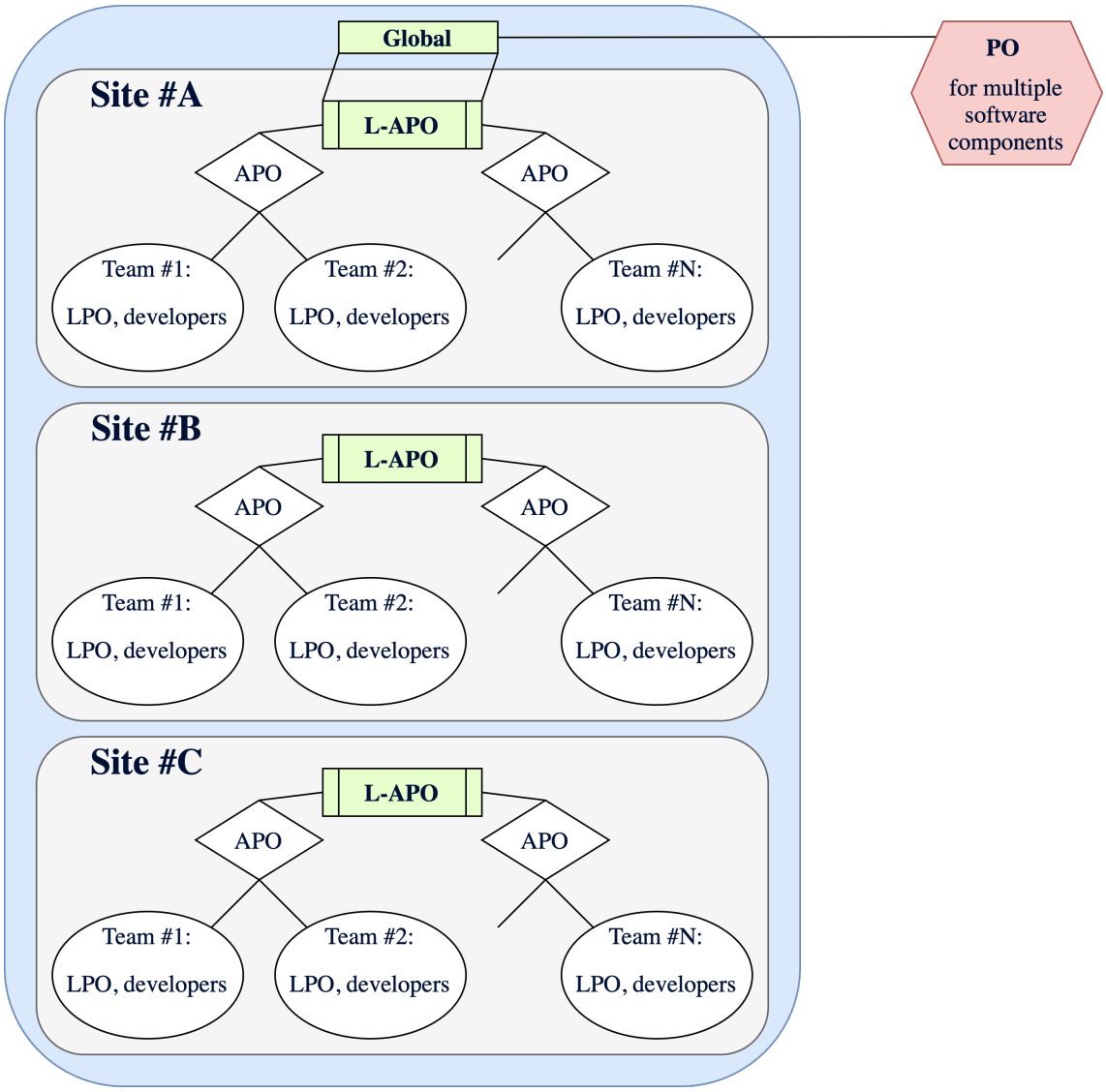

Figure 5: Depiction of project organisation in relation to software teams within one software component.

the project organisation who are required to deal with time-tracking and effort evaluation practices while covering all geographical locations of the software component in question. As depicted in Figure 4, software teams provide the time-tracking information. Furthermore, since software teams work under the jurisdiction of the product ownership hierarchy as shown in Figure 5, all product owners are stakeholders. Line organisation members were not interviewed as they do not take part in time tracking or effort evaluation directly. From the effort evaluation process point of view, the relevant stakeholders are, as before, software teams and the project organisation. Consequently, during the CSA, all interviewees were queried regarding both processes.

The interviewees were asked regarding basic information about time-tracking and effort evaluation practices in the case organisation, for example, why these processes are required in the first place, who is responsible for driving them, and what kind of practices the interviewees have seen in the case organisations. Appendix 1 includes the question battery containing the full list of the interview questions.

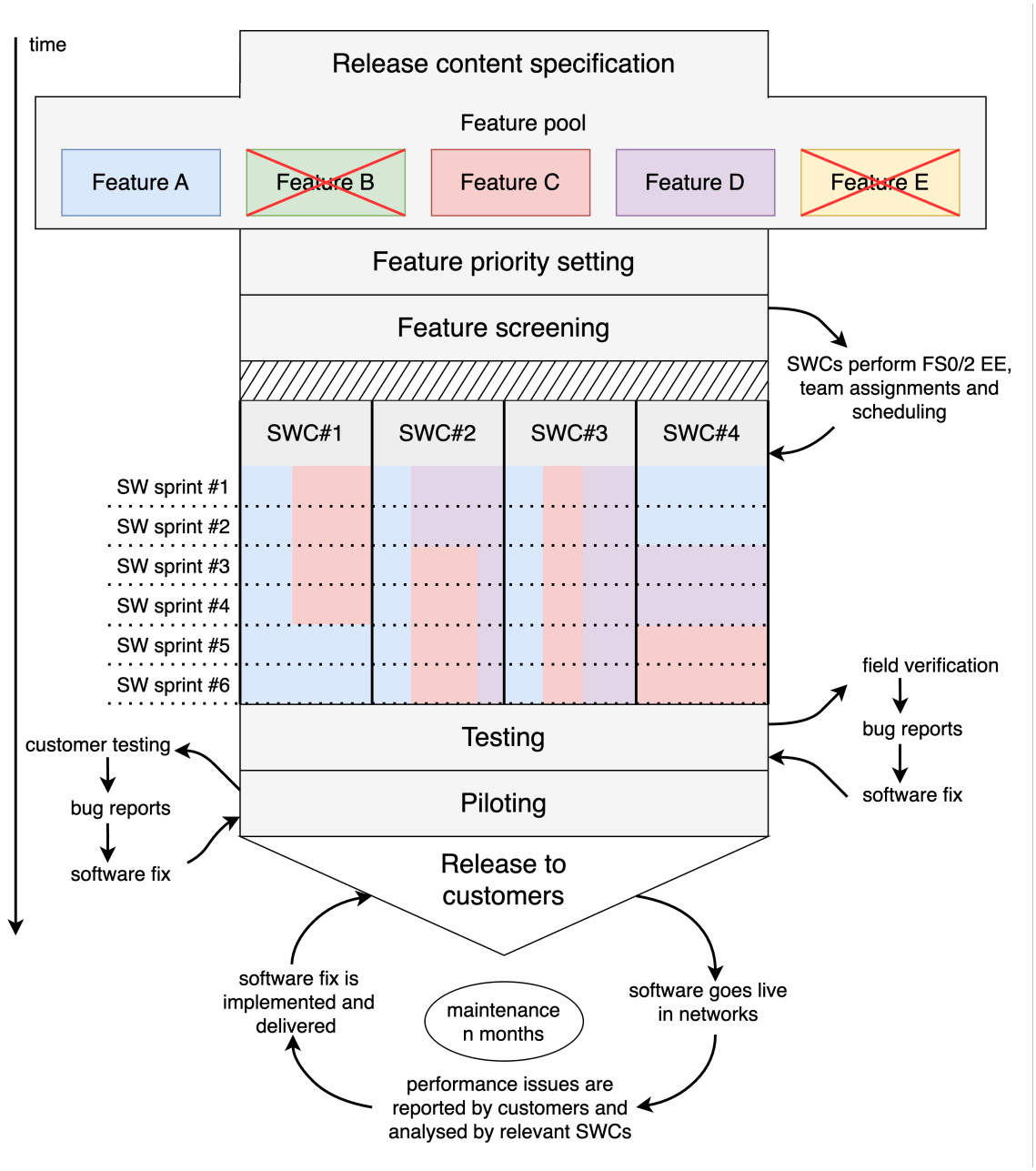

Figure 6: Figure showing a simplified view of the software release process. This mock-up release has a SW development period of six sprints.

Figure 6 shows where in the release process effort evaluation and time-tracking

are utilised. Effort estimation takes place before the actual software implementation, during which time-tracking is conducted. Software releases are parallelised, which means that when software of a release is being implemented, the content specification and feature screening may be on-going for the next releases, and testing and piloting on-going for the previous releases. Software fixes refer to fault management, which will be discussed briefly in Chapter 3.3.1.

Software Components (SWC) work in parallel during the implementation period. Depending on the feature, SWCs may be interdependent or independent of one another; the colours in Figure 6 aim to convey this: the area of the rectangles indicate the full capacity of SWCs to implement software height being time and width being an abstract indication of developer availability or capacity. Depending on the dependency relations and resourcing, each SWC needs to plan the timeframe, during which the implementation for each feature will be executed. Depending on the EEs provided during feature screening each feature's EE for SWCs will be different: some SWCs work on the features in parallel while some work in series .

For the afore-mentioned reasons, maintaining the schedules and time-tracking for future reference are needed so that the software for the release is ready on time. In case one SWC fails to deliver in the promised schedule, it may cause an cascade effect resulting dependent SWCs to also having to delay their implementation work ultimately causing the need to either postpone the whole release or push the delayed features to the next release. This further underlines the importance of accurate planning.

3.3 Description of Current Time-Tracking and Effort Evaluation Practices

There are multiple frameworks in the industry how time tracking and planning are executed in software houses. One of the popular frameworks is scrum. Scrum divides time into sprints, which in the case organisation last 14 days – from Wednesday to the Tuesday in two weeks [5, pp. 13–14]. A mock-up mapping of sprints to calendar time is shown in Figure 7. Current state analysis reveals that

while all teams execute both time tracking and effort evaluation, the practices in doing so are not cohesive.

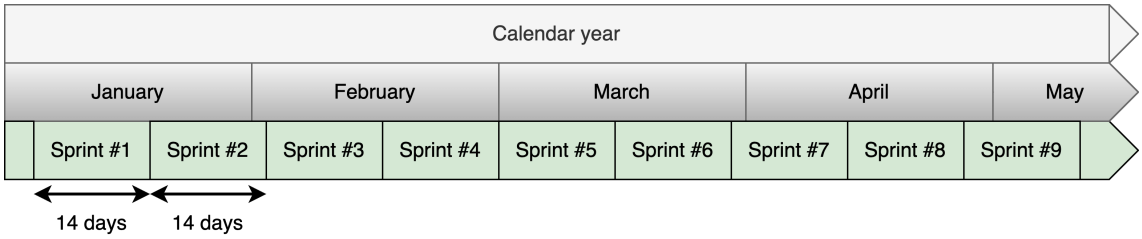

Figure 7: Mapping of sprint - the relevant planning time frame - to calendar time.

3.3.1 Description of Current Time-Tracking Practices

Each software team and site reports developer hours within a sprint. According to the interviews conducted, the tracking is done at minimum once a sprint, i.e. biweekly. In some teams, used time is reported once a day, in others more than once a week and in some once in two weeks. In all teams, time tracking is performed in Jira, which is a platform provided by the company Atlassian. According to the company itself, the tool is "*the #1 agile project management tool used by teams to plan, track, release, and support great software with confidence*" [6].

Every team in the case company works on multiple types of work, which may be roughly divided into two categories: backlog and non-backlog work. Backlog work is work, in the context of which time is used into new software features, internal improvements and support activities. As the name suggests, backlog work is shown on the project backlog, which is a sprint-specific listing of work to be completed within the time frame of one sprint. Software developers are required to log their used working hours when consuming time on backlog efforts. Backlog tasks are usually planned before the sprint starts according to basic agile practices. [7, pp. 14–15]

Non-backlog work, on the other hand, is work that either cannot be efficiently planned such as fault management or work that project management has decided that used working time is not tracked. Fault management could be bugs reported

by customers and the nature of the work is strongly reactive, resulting in the organisation not being able to efficiently plan such activities in advance: depending on the priority of the issue, which customer is reporting it and expected lead time to solve the issue, fault management work is required to constantly work on the most pressing problem. The amount of non-backlog work fluctuates from team to team as a function time. During the writing of this thesis, the approximated ratio between backlog and non-backlog work is 70:30. In Figure 6, it is shown that fault management is work outside of feature software development, however, consuming developer resources from the same pool due to the parallelised releases.

This thesis focuses on planning practices within the context of software planning. Non-backlog work is excluded from further discussion and is assumed to be of minimal relevance in this context. The reason why this conclusion may be drawn is that the impact of backlog and non-backlog work to time-tracking practices is independent: reserving a software developer's time into non-backlog activities only scales down the need to track developer time for other activities decreasing the impact of differing practices within the organisation but not removing it completely. From now on, it is assumed that all developers in all teams work 100% in context of backlog work. In reality the capacity scales from team-to-team depending on the backlog and non-backlog work ratio.

There were three key differences found during the current state analysis: stakeholder for logging responsibility, amount of hours logged per developer per unit time and frequency of time-tracking events. In most teams, it is the software developers themselves who carry out the tracking, but in few of the teams, time tracking is carried out by the LPO. No documentation was found or official guidance reported to be given, when investigating who should execute this process.

The greatest difference, however, are in the amount of tracked time. The amount of hours logged per time unit per developer differs between sites and teams. In order to report the amount of hours logged properly, one has to first understand

underlying basic concepts. Firstly, in all of the geographical sites the general working time in one sprint is 7.5 hours a day for 10 weekdays, which accumulates into 75 hours per normal sprint – assuming 10 working days in a two-week time frame. However, the case organisation accepts the fact that it is unrealistic to expect 7.5 hours of efficient work time per day. The reason for this is that in the context of scrum or agile, there are mandatory sprint events or meetings called *ceremonies*. The ceremonies, which developers attend in the case organisation depend on the software team in question, but include and are not limited to the daily, the planning meeting, and the retrospective decreasing the working time during a day. In addition, as in any organisation, there are info sessions, result calls, and training sessions, which further decrease the effective working time.

As reported by the interviewees, the case organisation expects that 5 to 7.5 hours per day are logged, but the exact figure depends on multiple variables. For most of the interviewees, the most common daily amount of hours tracked was 6 hours per day, which corresponds to 60 hours per sprint. In addition, as each team has been given high autonomy to agree upon their preferred working style when planning and implementing software features, the current state analysis uncovered that there is no exact value used by every team.

During the investigation of company documentation, no concrete documentation on time-tracking practices was found for the software component that is investigated in the context of this thesis. One unofficial guidance was found in the documentation archive of another software component. As a reference, this documentation proposed that each day should correspond to 7 hours per day or 70 hours per sprint of logged work time.

During the interviews, the global L-APO suggested that there is *informal guidance* within the relevant software component of 6 hours per day. By informal guidance, the conveyed message of the interviewee was that 6 hours per day is a general rule of thumb, which may be applied within software teams and it is not a mandated or officially documented figure.

Example	Time (hours)				Percentage
	Per Day	Per Sprint	Per Quarter	Per Year	
#1	5	50	300	1300	-
#2	5.5	55	330	1430	+10%
#3	6	60	360	1560	+20%
#4	7	70	420	1820	+40%
#5	7.5	75	450	1950	+50%

Table 1: Visualisation of the difference in reported hours per day causes in the long run; quarter is assumed to be 6 sprints and year 26 sprints in this table.

Table 1 shows how the differences in daily tracked developer time accumulates over time. Hour or two in a day may not seem like a great difference, but over time the differences accumulate. The reported daily logged developer times per interviewees have been chosen as the examples for the table. The table assumes that all weekdays are working days and that there are no holidays within the year for simplicity. However, the deviation of 50 % between the greatest and smallest numbers remain regardless. In absolute terms, in the most radical case, the difference of logged hours between two software developers working full time for one year is 650 hours.

If the deviation is further expanded over multiple software developers – say per team consisting of five or ten software developers – the output of a team logging 5 hours per day over a year will consume 7,500 and 13,000 hours and a team logging 7.5 hours per day over year will consume 9,750 and 19,500 hours respectively, which clearly shows that the deviations accumulate over time causing a great discrepancy of the apparent efficiency of a software team.

The final difference in time-tracking practices was frequency of logging. The lowest frequency reported was once per sprint and the highest frequency was once per day. Inherently, it makes no difference what the frequency of logging is as long as human error of forgetting is excluded. Further analysis shall be done in Chapter 3.4.

3.3.2 Description of Current Effort Evaluation Practices

Effort evaluation may be roughly divided into two different categories: new feature effort evaluation before feature is approved for development and refining the initial effort estimates when feature work has been assigned to a team and the implementation is planned.

When a new feature is drafted and proposed, there are various steps in the process that are taken. In the interest of keeping this thesis compact, only the relevant parts of the process are discussed. The feature is assigned a system release, business priority and all the affected software components execute two levels of effort evaluation: FS0 and FS2. The abbreviation FS comes from feature process checkpoints from the term “feature screening”. Both of these checkpoints precede the *Approved for Dev* decision, after which implementation team assignment, Feature Owner Team (FOT) meetings and planning are triggered. Mapping against software development may be seen in Figure 6.

In both checkpoints, each of the software components triggers an evaluation to estimate needed efforts to implement the feature in question. The estimate is always a singular value, whose unit is an hour. The estimate is specific to each software component and may range from zero to tens of thousands of hours. FS0 is the first checkpoint and usually this estimate is still quite crude. Depending on the nature of the feature, FS2 effort estimation (FS2EE) is requested after FS0EE, but the timespan may be anything between few weeks and few months. FS2EE is expected to be a more refined estimate compared to FS0EE. After the feature has been approved for development, FS2 effort estimates are used to reserve capacity from software components, which then maps to the resource tool used by the case company. The effort reservations are compared against team capacities and they are followed on multiple levels: team, department, site, software component and whole software project.

In FS0 and FS2 effort evaluation phases, the L-APO requests some individual experts or teams to do the evaluation. On a high level, the evaluation in driven by

L-APO, but it may also be delegated to APOs or directly to skilled technical experts. According to the information obtained through interviews, both the process and ways of working within teams are quite established and there are no radical differences. Essentially, L-APO, APOs, and technical experts utilise existing information from completed features and combine that information with the specification documents available for the new feature and evaluate the needed efforts. If possible, initial team assignment is made to gain more accuracy. If not, then the reserved effort will be labelled “unassigned” remaining unplanned for the time being.

The second type of effort evaluation is the refinement of FS2EE when feature implementation is started. Each software component assigns software teams to implement required functionality in the context of a new feature. In this process, the teams use the FS2EEs as an input to their plans. Essentially, the work is split into more manageable chunks which may be planned in a time frame mandated by the feature implementation schedule. In this phase, the singular FS2EEs values are split into parts and more technically-informed evaluation is executed. The dependencies between software components are discussed within the Feature Owner Team, which is a virtual team consisting of a FOT leader hosting the meetings and representatives from each affected software component. Additionally, if multiple software teams are assigned work from a single feature, in addition, dependency and schedule planning meetings are needed also within the software component. If only one team has been assigned to a feature, usually, one team can handle the software-component-specific content planning.

In the FS2EE phase, the evaluations become more granular and the work is planned with more resolution in time as well. The work is scheduled according to the release planning guidelines. In the latter phase, with the information of dependencies and team capacity, the work is planned in time so that consumed development time is not necessarily linear but can instead fluctuate on a sprint-to-sprint basis.

Figure 8 shows an example of the output from a resource management tool in the

case company. This example denotes the capacity, time spent and estimated efforts, both in hours, as a function of sprints. The implication of the example figure is that software development is currently in Sprint #2. Furthermore, it seems that the team is overbooked for Sprints #3, #6, #7, #8 and #9. The dark red time spent bars represent time-tracking data. Light red bars, on the other hand, represent efforts estimates: FS2EEs, more granular, refined feature estimates, and also FS0EEs if team assignment has been already carried out.

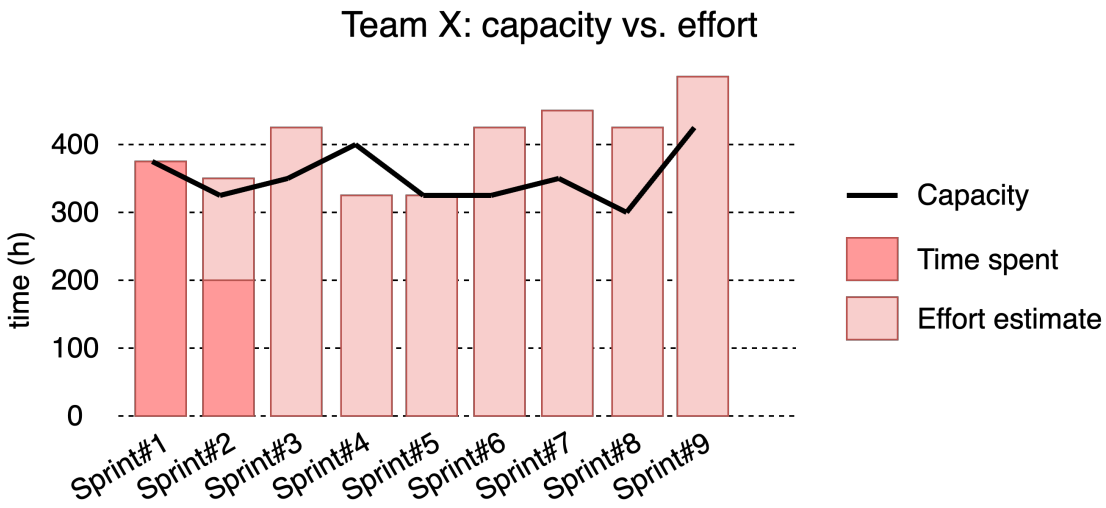

Figure 8: Example mock-up figure representing data generated by the company resource management tool: the capacity is compared to planned and consumed effort for each team, department, site and software projects is seen in such figures.

3.4 Analysis of Time-Tracking Practices

During the execution of the CSA, it was noted that across the investigated software component, the time-tracking practices are mainly not harmonised. Practices differ across the sites and teams. Table 2 summarises the analysis of time-tracking practices.

Organisational differences	Strength	Weakness
LPOs carry out time tracking on behalf of the members in some teams	LPOs logging consumes less time and logging is more reliable	LPOs logging may cause worse resolution of used time
Variations in the amount of tracked time per unit time between teams	None	Comparison of used time between teams becomes compromised
Frequency of logging is not harmonised between teams	More team autonomy and teams may decide on best approach	Less frequent logging may lead to inaccurate time-tracking data

Table 2: Strengths and weaknesses of time-tracking practices in the case organisation.

Most notably, the amount of hours logged per unit time and responsibility-bearer of logging efforts vary greatly. The amount of hours logged per day, assuming full time used on backlog items, varies between 5 to 7.5 hours. Additionally, in some of the teams it is the team LPO who does the logging, while in most teams it is the software developer themselves who perform time tracking.

Next, the strengths and weaknesses of the listed differences will be evaluated in Chapters 3.4.1 and 3.4.2.

3.4.1 Strengths in Time Tracking

In general, in the software component in question, teams are allowed to build their practices quite freely. This means that also in the context of time-tracking activities, there are bound to be differences, as not all practices have been defined by the case company to be executed in a certain way.

Team autonomy certainly has its benefits and when considering the time-tracking practices and their difference within the software component Table 2 lists the main benefits. The most common time-tracking stakeholders are the developers themselves. However, as nothing prevents the teams from deciding that the logging of hours is centralised to the LPO, some teams have opted to adopt this approach. Consequently, such practices were reported in the interviews. It seems

that the practices are usually harmonised between teams on one site, but the practices for the stakeholder who performs logging, then again, differ from site to site. If one person does the logging, it is more probable that each team member's used time is reported. In cases when developer performs time tracking personally, there is always the risk of human error of forgetting. In addition, if one person performs this in a centralised manner, there is less total time used on dealing with the tools used for time tracking. It may be argued that there is more time to concentrate on actual work.

The second difference in the time-tracking practices of the case company is the amount of tracked time per unit time. There is no evident benefit to the amount of tracked time being different. According to the responses from the interviews, the differences constituting why amount of logged hours differs is due to unclear definition in the organisation what constitutes "backlog hours", i.e. time that is expected to be logged on backlog items. The case company also expects each developer to use time on competence development. If a developer is to study a feature before actual implementation, should this be considered competence development or actual working time under the feature in question? And, lastly, since the teams in the case organisation mostly follow agile and scrum-like ways of working, as mentioned in Chapter 3.3.1, each developer is expected to join scrum ceremonies which, depending on the team, may take up to 3-4 hours of working time per week. It seems to be unclear to some people in the organisation whether these hours constitute backlog work.

Another reported issue that relates to the previous two points is that there are personal differences between team members. Some are reportedly difficult to motivate about the importance of hour logging. Additionally, others are very conscientious about the work they do and they refuse to log hours for the time they sit in meetings – even if the team or site practices dictate that feature-related and scrum-related meetings are part of the feature work, these individuals choose not to log hours for the time used in meetings. Even with the direct request from the team LPO, interviews indicated that such occurrences take place.

The third and last difference is that teams and their members may choose the frequency of logging independently from other teams. The strength in this practice is that teams may decide on the practice that works best for them. If less frequent logging is accepted, less time may be used to deal with the tools.

3.4.2 Weaknesses in Time Tracking

Each of the three listed organisational differences in time tracking also have their downsides. For LPOs carrying out time tracking, the evident weakness is that LPOs probably do not know the amount of hours used by each developer even down to the hour. This means that either gross generalisation of a person's time is made or, alternatively, a team LPO would need to ask for detailed reports from each developer to log the hours accurately. In other words, either resolution in logging is lost or the time saving benefits of performing time tracking centrally are not gained.

The most crucial weakness is the fact that the amount of time logged per developer varies up to 50% depending on the reference point from team to team. In essence, this causes that the unit is different from adding up hours between teams. If one developer logs 5 hours per day in one team and the same developer would log 7.5 hours per day in another team for the same work done if they were to transfer team, comparing these two values with seemingly same unit – hours – does not make sense. As was shown in Table 1, the differences accumulate per team as the inspected time frame and team size grow. Basic mathematical principles indicate that addition cannot be performed on two numbers with different units. Yet, such calculations are executed when calculating team capacities and feature and product costs in the case organisation.

Another, weakness is that in some teams, the work is not refined to specific tasks. Instead, the work is tracked generally, e.g. "Feature X work during Sprint #N". Such practices are strictly not forbidden, but they are against the basic principles of agile development: "The product Backlog list is constantly updated with new and more detailed items, as well as with more accurate estimations and new

priority orders” [7]. Such practices make the logging of work easy, but in the process resolution and visibility is lost. There is no way to trace what was done in the context of a feature implementation if there is no backlog item corresponding to the work with sufficient information regarding the content.

Lastly, a less crucial weakness is caused by the frequency of time-tracking events. In an ideal world, the frequency does not matter, but in a real-world application humans are prone to errors. If the frequency of logging hours is sparse, it becomes easier to forget. There are various reports from the interviews that such forgetfulness sometimes takes place, which, naturally, leads into inaccurate reports from teams. This in turn, makes feature and product cost calculations less accurate.

3.5 Analysis of Effort Evaluation Practices

Feature effort evaluations are done in multiple stages: FS0, FS2 and during implementation. FS0EE and FS2EE are clearly defined by company processes and are, hence, harmonised. According to the conducted interviews, each site performs these evaluations similarly. One key reason for already harmonised processes is that FS0EE and FS2EE processes are practically completely centralised so that L-APO of the SWC takes care of all FS0EE and FS2EE queries. From there, L-APO delegates subtasks or parts of the estimation to experts or APOs on all sites.

More granular refinement of effort estimation is done after FS2 when the feature implementation is starting or on-going. Each team conducts these evaluations in the manner best suited for them. Most often, the teams follow the scrum ceremonies: planning, refinement and retrospective. During planning, next sprint is prepared by adjusting backlog item effort estimations, assigning work to developers and assigning work to a specific sprint. During refinement, the estimates and assignments may be changed according to team needs. The refinement meeting is usually held in the middle of the sprint. Lastly, in the retrospective meeting the team delves into the previous sprint and how it went.

However, not all the teams in all the sites refine their feature work on a task level. This means that, instead of creating backlog items with clear description of what kind of work will be done, general items only are shown on the backlog.

In addition, according to the data gathered from the interviews, there has been a trend where the FS2EEs are not sufficient to implement the feature. This means that, more often than not, the FS2EE are underestimated leading into delays in the implementation process because in reality more time resources are needed than originally reserved.

The company does have means to prevent such errors in estimation, although they are partially retroactive. In practice, in case the FS2EE deviates greatly from actual reported time spent, the process dictates that an analysis is conducted on the reasons why this occurred. On paper, this could be seen as a risk that actual time spent would be either under-reported or over-reported on purpose in order to avoid the analysis forcing the team to explain their deviation to project management. However, according to the interviews conducted, no such occurrences of falsifying time-tracking data have been exposed.

3.5.1 Strengths in Effort Estimation

The most apparent strengths of the current effort evaluation practices are solid process structure and mostly harmonised practices between sites and teams. The features introduced follow a strict process and from effort evaluation point of view, each feature must provide FS0 and FS2 effort estimates before a feature is approved for development. All software components, sites and teams must follow the same process. This, by default, essentially forces harmonisation between teams.

Another reason is the fact that all FS0 and FS2 effort estimation queries for the software component in question go through the L-APO within the current process.

3.5.2 Weaknesses in Effort Estimation

There are, however, weaknesses in effort evaluation practices. Even though processes are quite harmonised, giving FS0EEs and FS2EEs is challenging with limited knowledge of the upcoming feature. Especially, enormous features are hard to evaluate and they are prone to deviations between FS estimates and actual time spent.

Additionally, some teams do not follow basic scrum principles where task-level refinement is expected. This further obscures the actual time spent on a specific feature if the traceability of completed work is not possible, thus undermining the principles of agile software development. Additionally, this reduces the accuracy of both time-tracking and effort evaluation practices. Lastly, it has been reported by some interviewees that there is a tendency of effort evaluations being underestimated resulting in re-planning needs causing either delays or remapping of content to later software releases.

3.6 Summary of Key Findings

The current time-tracking practices within the organisation vary significantly across teams and sites. Each team reports developer hours within a sprint. The frequency of time tracking ranges from once a day to once every two weeks, with some teams reporting time daily while others do so less frequently. Importantly, there’s a division of work into backlog and non-backlog categories, with backlog work being planned before each sprint and non-backlog work being more reactive and less planned.

Responsibility for time tracking differs between teams, with some having software developers themselves perform the task, while in others, it’s carried out by a designated team member such as the local product owner (LPO). The amount of time logged per unit time varies significantly, influenced by factors such as team habits, customs, and non-documented hearsay, with reported daily logged hours ranging from 5 to 7.5 hours.

Effort evaluation practices follow a structured process involving two levels of evaluation (FS0 and FS2) before a feature is approved for development. However, there are challenges, such as accurately estimating effort for large features and inconsistencies in task-level refinement across teams. Some teams do not follow the basic scrum principle of task-level refinement, which can lead to inaccuracies in both time tracking and effort evaluation.

Strengths of these practices include the flexibility for teams to adapt their own approaches and the structured process for effort evaluation, ensuring consistency across sites. However, weaknesses include discrepancies in time tracking and effort estimation, especially for large features, and inconsistencies in task-level refinement. Thus, undermining the accuracy of evaluations. The benefits and the drawbacks of current practices within the SWC in the case company are shown in Table 3. It is evident that while the teams are able to choose their apporaches freely, the weaknesses

Strengths	**Weaknesses**
Low process overhead	Inconsistent time-tracking across teams
High team autonomy	Teams may log hours based on estimates rather than actual work
Easy balancing of capacity and effort	Capacity estimates can become unreliable, leading to project failures
Flexible time-tracking practices	Poor customer visibility if deliveries are missed
Option for centralised logging reduces errors	Logging lacks detail or requires extra effort from LPOs
Less admin work with centralised logging	Hour tracking varies up to 50% between teams, making comparisons meaningless
Teams control logging frequency	Some teams log work too broadly, reducing traceability
	Infrequent logging leads to errors and inaccurate cost calculations

Table 3: Summary of strengths and weaknesses in current time-tracking and effort estimation practices according to CSA.

The next chapter presents ideas from literature which concern how time-tracking and effort estimation practices might be improved within the case company. The current state analysis conducted in this chapter will function as a basis point for the literature review and those findings will be processed further to discuss addressing the found weaknesses in Chapter 3.

4 Improvement Ideas for Planning in Literature

This chapter examines ideas and best practices related to the weaknesses identified in Chapter 3. Building on the analysis in the previous chapter, this chapter develops a conceptual framework for time-tracking and effort evaluation processes. Moreover, concepts from the literature on software development, particularly those related to time-tracking and effort evaluation, are synthesised to generate the conceptual framework.

4.1 Agile Release Planning

Software release planning using agile methodologies, often referred to as agile release planning, is a relatively new area of study. Its roots can be traced back to development practices that emerged in the 1980s and later evolved into agile methods in the 1990s [8, pp. 4–5]. The case organisation utilises agile release planning and software development practices whose origin may be traced to The Agile Manifesto. The Agile Manifesto, formulated in 2001 by a group of software developers, represents a key milestone that revolutionised software development practices. The manifesto emphasises a flexible, collaborative approach to project management, highlighting the significance of individuals and interactions, working software, customer collaboration, and responsiveness to change. By prioritising these values over traditional methods, the Agile Manifesto advocates for a more adaptive and efficient development environment. Its twelve guiding principles further elaborate on how to achieve these goals, promoting iterative progress, technical excellence, and continuous improvement. This approach has become a cornerstone of modern software development methodologies, fundamentally transforming how teams deliver value [9].

The manifesto essentially serves as a foundational baseline for all agile methodologies. Consequently, the case organisation has been, and continues to be, heavily influenced by the manifesto, as well as by the discussions and frameworks that have emerged from it over the last two decades. However,

literature on the topic of agile release planning is relatively scarce, with most related articles being case studies [10, p. 2]. Nevertheless, there is ample practical data available from organisations that have undergone agile transformations, and the case organisation is one such entity that has adopted agile practices in recent years.

One of the most critical aspects of agile release planning is that the plan should not be rigid once established. Instead, it is an iterative process wherein sprint-based development cycles provide feedback to teams about progress and upcoming priorities. As such, the plans necessitate continuous replanning and adjustment as needed [10].

4.2 Planning in an Organisation

For planning practices to be effective across a large organisation, it is the organisation's responsibility to define the key planning processes that each development team should follow. As Kantola et al. propose, the plan itself has no inherent value; rather, it is the instrument through which a team – and consequently the organisation – will produce value and achieve its goals [10, p. 2]. In other words, planning is a key foundation for a successful software development team and for a company that produces software.

Both effort estimation and time-tracking practices in the case company, as discussed in the CSA in Chapter 3, are not strictly defined but rather allow development teams significant autonomy. This practice extends across most of the problem areas discussed in the CSA. Generally speaking, the organisation prioritises team autonomy in the hope of optimising software development output. However, many researchers have reported findings that discourage excessive emphasis on autonomy and weak commitment to agile practices. For instance, Kantola et al. suggest that user stories – items in the product backlog which represent the smallest tasks to be completed by a team – are critical, as they express the requirements that must be fulfilled during software development [10, p. 2]. As mentioned in Chapter 3.4.2, some teams opt not to have clear tickets or

stories on the sprint backlog detailing the upcoming work clearly, but instead the backlog functions as purely keeping book of consumed developer time per sprint. Literature indicates that such practices are not desirable as the requirements become detached from the software development [11, pp. 33–34]. To deliver greater value, development teams must take the time to refine backlog tickets related to customer requirements.

Schwaber et al. also state that by having clear backlog of items with requirements to be fulfilled will also result into committing into definition of done. In essence, definition of done is a formal description of what a backlog item will accomplish for the software being developed. If the content of the ticket is a requirement to be fulfilled, within the definition of done of this item, the product will by design become more complete [11, p. 12].

Such requirement of backlog items are the core of Scrum and the organisation would need to establish the common ways of working between software teams that each developer and team shall adhere to by the principles of agile methodologies. The literature is not unanimous on the importance of implementing agile holistically; some proponents of agile argue that only by completely implementing the agile methodologies, the benefits of agile are achieved [12]. However, according to Bass et al., implementing agile methodologies fully is not possible in all types of companies at all [8, p. 5]. Furthermore, another study found that fewer than 15% of software development in enterprises or companies is conducted purely in traditional or *agile* approaches, indicating that implementing practices purely in an agile way is more of an exception than the norm [13, pp. 3530–3531].

Finally, across software development industry, Kuhrmann et al. found that, when asking project managers and developers what agile software development means in the first place, there is a substantial amount of confusion regarding the terminology. They found that many people believe they work in an agile way, even though they only use methodologies associated with agile [13, p. 3524].

4.3 Process Compliance in an Organisation

The case organisation in question utilises a combination of various development frameworks, with Scrum being the most prominent. However, a multitude of frameworks exists, among which the most commonly used are Lean, Scrum, Extreme Programming (XP), Scrumban, Kanban, and many others. Agile, on the other hand, is a method rather than a framework [11]. The case organisation employs a mix of frameworks, but the most prominent features are derived from Scrum, specifically sprint ceremonies and the roles within the software development organisation, such as product owners and development teams, as detailed in Chapter 3.2.

There is no one-type-fits-all framework for an organisation nor is there one that is superior to the others. As an example, it is suggested in some of the literature that larger organisations might need to choose a hierarchical requirements model, instead of a flat one where each requirement converts into a backlog item as agile release planning is a highly complex process, which consists of many inter-dependent activities [10, p. 2].

To reduce project risks and improve both productivity and product quality, the use of agile methods has been widely advocated [8]. This necessitates that the case organisation defines and enforces optimal practices in effort evaluation and time-tracking. As such, sticking to a set of principles in agile release planning is essential for getting the best results. However, this sticking to one set of principles is does not mean that the processes and practices shouldn’t be iterated upon. Agile planning involves deciding what work needs to be done, scheduling tasks, and managing resources while being flexible to changes. Using consistent principles helps organise these tasks better, making it easier to prioritise work and allocate resources. This consistency improves communication and reduces confusion, making sure everyone is on the same page. It also helps manage changes more effectively, leading to better efficiency and success in meeting project goals.

Lastly, the case organisation operates across multiple geographical locations creating a multicultural working environment. Different cultures may accept and implement company processes in varying ways. However, literature suggests that a multicultural work environment can be a source of increased productivity, provided that it is actively managed to ensure that developers remain compliant with established processes [14, pp. 71–72].

4.4 Roles and Responsibilities

There is plenty of literature on how clear role definitions in a team correlate with better task outcomes. Hackman and Oldham's 1975 article introduces the Job Diagnostic Survey, a tool designed to assess job characteristics and their impact on employee motivation and satisfaction. It emphasises how job design influences motivation through factors like skill variety, task identity, and autonomy, offering a framework for improving job efficiency [15].

4.4.1 Job Ambiguity and Satisfaction

Researchers have found a clear link between lack of role clarity and output and efficiency of a developer in a software engineering setting [16, pp. 16–17]. Role ambiguity occurs when team members are unclear about their responsibilities, tasks, or how their work aligns with others' roles. This issue is particularly detrimental in software development teams, as it can lead to significant performance problems. Studies consistently show that role ambiguity impairs team effectiveness and project outcomes.

For instance, Beecham et al. found that unclear roles often lead to increased errors and inefficiencies in software development projects. Ambiguity in role definitions can cause team members to duplicate efforts or overlook critical tasks, resulting in delays and increased costs. This confusion also fosters frustration and lowers morale, which further complicates the problem. [16]

Dingsøyr et al. emphasise that in agile teams, role ambiguity disrupts team

dynamics, hindering collaboration and communication. Clear roles are essential for maintaining alignment and ensuring that team members can effectively coordinate their efforts [17].

In general, literature on managing role ambiguity in agile teams and software development environments stresses the importance of clear role definitions for reducing conflicts and improving team performance. For instance, managing role ambiguity effectively contributes to better project outcomes by clarifying expectations and enhancing team cohesion.

Avoiding role ambiguity is crucial for fostering efficient, effective, and harmonious software development processes. Clear role definitions help mitigate errors, enhance team dynamics, and contribute to successful project completion.

4.4.2 The Impact of Product Owners on Software Project Outcomes

Research and studies on agile methodologies suggest that the role of a product owner is in key role in determining how successful a development team is [18].

According to Judy et al., the product owner is the cornerstone in maximising the value of the output of the development teams and that POs should have ownership of project priorities, execution of the plan and to maintain a clear vision of what their teams are required to produce [19].

In order to set clear practices for effort evaluation and time tracking, the stakeholders and responsibilities need also be defined properly. As discussed in the CSA, it seems that especially for time-tracking, it is not clear who is responsible for making sure the team carries out the tracking according to the process.

Moreover, many researchers highlight the product owner's importance in optimising a software team's output. According to Sverrisdottir et al., the product owner is "responsible for the financing of the project during its life-cycle and he puts forward the requirements and objectives of the project. His most important role is to maximise the output of the team, and the output for each task" [18,

pp. 260–261]. Product owners also play a key role in the overall software development process, as they facilitate communication between the customer and development teams. They are responsible for developing and maintaining the product backlog, which contains the user stories that define the project's requirements [8].

The literature consistently agrees that the product owner role must be clearly defined. Given the significance of the product owner's role, having a clear definition of the tasks expected of them is paramount.

4.5 The Impact of Cognitive Biases on Effort Estimation and Planning

Psychology has investigated planning-related biases for several decades, revealing a variety of cognitive distortions that can significantly affect our ability to make accurate predictions. These biases operate subconsciously, influencing our judgments and decisions without our awareness. In this discussion, three of the most prominent biases that impact our planning will be explored in the context of software development: the time-saving bias, the planning fallacy, and the optimism bias. Each of these biases contributes to a skewed perception of how long tasks will take and how likely we are to achieve our goals, often leading to less effective planning and decision-making.

They are a crucial consideration in effort estimation for software development due to their potential to distort project planning and resource allocation. Effort estimation involves predicting the time and resources required to complete various tasks, foundational for creating realistic schedules, budgets, and milestones.

4.5.1 Time-Saving Bias

Research by Svenson et al. revealed that people often incorrectly estimate time savings when increasing speed, due to employing the proportion heuristic [20]. In essence, it considers the speed difference as a proportion of the initial speed, leading to errors such as overestimating the benefits of minor speed increases in

already optimised processes and undervaluing significant improvements in less efficient areas. For instance, in a study, conducted participants believed that increasing speed from 70 to 110 km/h saved more time than from 30 to 40 km/h, while the latter actually saves more time [21, pp. 99–101]. Intuitively, one might assess that as

$$\frac{110}{70} \approx 1,57 > \frac{40}{30} \approx 1,33, \tag{1}$$

it would naturally make sense for the prior speed increase to save more time. However, if we consider an arbitrary distance – say 30 km – increasing speed from 30 to 40 km/h will save 15 minutes while increasing speed from 70 to 110 km/h will save a bit over 9 minutes. This bias has been observed across various contexts, including software development [22].

In software development, agile methodologies emphasise iterative development and frequent reassessment. Misestimating time savings can lead to improper task prioritisation, where efforts are focused on areas with minimal actual gains, neglecting more impactful opportunities for improvement. This can result in underestimated project timelines, leading to missed deadlines and cost overruns. Furthermore, time-saving bias can affect team morale and stakeholder expectations. Overestimations can create unrealistic expectations, leading to disappointment and mistrust when predicted gains are not realised. Conversely, undervaluing substantial improvements can demotivate teams, as their significant efforts may appear less impactful.

For the reasons explained above, both recognition and mitigation of time-saving bias through regular task reassessment ensures more accurate effort estimations. This leads to better project planning and efficient resource allocation. Additionally, improved overall performance in software development projects may be achieved. Specifically, in the context of evaluating the amount time required for a software development task, research suggest that there is a requirement to consider cognitive biases when making decisions as contributing factors to such

inaccuracies [21, pp. 107–108].

4.5.2 Planning Fallacy

The planning fallacy is a cognitive bias that leads individuals to underestimate the time, costs, and risks of future actions while overestimating the benefits. This phenomenon, first identified by psychologists Daniel Kahneman and Amos Tversky in 1979, reveals a fundamental flaw in human prediction and planning [23]. People tend to be overly optimistic, assuming that everything will go as planned without accounting for unforeseen obstacles.

Research has shown that individuals consistently expect to finish their tasks earlier than they actually do [24, pp. 377–380]. Buehler at al. have studied that this optimistic bias can be replicated and fewer than half of the participants completed their tasks within their predicted time frames. What the research seems to indicate is that people in charge of planning often construct narratives focusing on how the future would unfold, rarely considering their past experiences with similar tasks or potential problems [24, pp. 377–380].

The literature suggests that people tend to disregard personal characteristics, others' experiences, and deadlines, focusing instead on specific details of their plans. This aligns with the planning fallacy, where people generate forecasts based on specific scenarios rather than considering broader information. In addition, Buehler et al. have shown that in cases when delays occur, the people in charge of planning would often attribute their delays to unique, non-recurring circumstances, diminishing the relevance of past failures to current predictions [24, pp. 377–380].

In software development setting specifically, Shmueli et al. study the planning fallacy, which manifests as time underestimation, scope overload, and over-requirement. Over-requirement in this context is defined as extra development efforts for no good reason. The issues often stem from overly optimistic time estimates, leading developers to include more features within a

project's scope. There are two mechanisms in play when considering how to prevent falling prey to planning fallacy: considering past or historical data systemically from similar already finished software implementations and software developers who perform the implementation not being the actors doing the estimation as well. The findings suggest that both mechanisms together have an additive effect, improving decision-making more than either mechanism alone. However, these interventions do not entirely eliminate biases. The study emphasises the need for developers to use historical data for more realistic planning and for managers to recognise the propensity of developers to overestimate what can be achieved in a given timeframe. [25]

4.5.3 Optimism Bias

Optimism bias was first extensively studied by psychologists Neil Weinstein and his colleagues in the 1980s. Weinstein's research highlighted how individuals tend to perceive themselves as less likely to experience negative events compared to others, a phenomenon that came to be known as ”optimism bias” [26].

Optimism bias is a cognitive distortion where individuals overestimate the likelihood of positive outcomes and underestimate the probability of negative ones. This significantly impacts project management practices. Also prevalent in software development industry, this bias leads to unrealistic scheduling and budgeting, which can undermine project success [27, pp. 380–382]. As noted by Flyvbjerg et al., optimism bias often results in overly ambitious timelines and cost estimates, which are not grounded in empirical evidence or historical data [28].

In software development, optimism bias can cause several critical issues. First, it can lead to underestimation of project timelines [23]. Developers and project managers may assume that tasks will proceed more smoothly than they do in reality, leading to delayed delivery and missed deadlines. Secondly, optimism bias contributes to budget overruns [29, pp. 4–5]. By underestimating the resources required, projects may face financial shortfalls, impacting their overall viability and success. Finally, this bias can result in scope creep, where the project scope

expands without adequate adjustments to time and budget [24]. The tendency to overlook potential risks and challenges means that software projects might continue to grow beyond the original plans, straining resources and leading to compromised quality.

Despite the recognition of these issues, the mitigation strategies for optimism bias, such as reference class forecasting, have not been extensively validated for software projects. Research suggests that while this method shows promise in engineering contexts, its application to software development remains largely untested [28]. Therefore, further empirical research is essential to confirm the effectiveness of these techniques in the software development domain and to develop tailored strategies to address the specific challenges posed by optimism bias in this field.

4.6 Conceptual Framework

Based on the literature reviewed in the previous sections, a conceptual framework is formed. Figure 9 combines three different categories: processes, roles and culture.

This conceptual framework outlines three critical areas essential for the success of agile software teams: robust processes, clearly-defined roles, and company culture. Robust processes involve the establishment of a clear definition of agile methodologies suitable for software teams. They also include the implementation of harmonised backlog practices for ticket creation and time tracking.

The framework emphasises the importance of clearly-defined roles to avoid role ambiguity. Additionally, it highlights the need for clear product owner role definitions to ensure streamlined team operations.

Lastly, the framework addresses the role of company culture by advocating for the integration of cultural diversity into organisational processes. It also focuses on mitigating biases related to time estimates. Together, these elements provide a

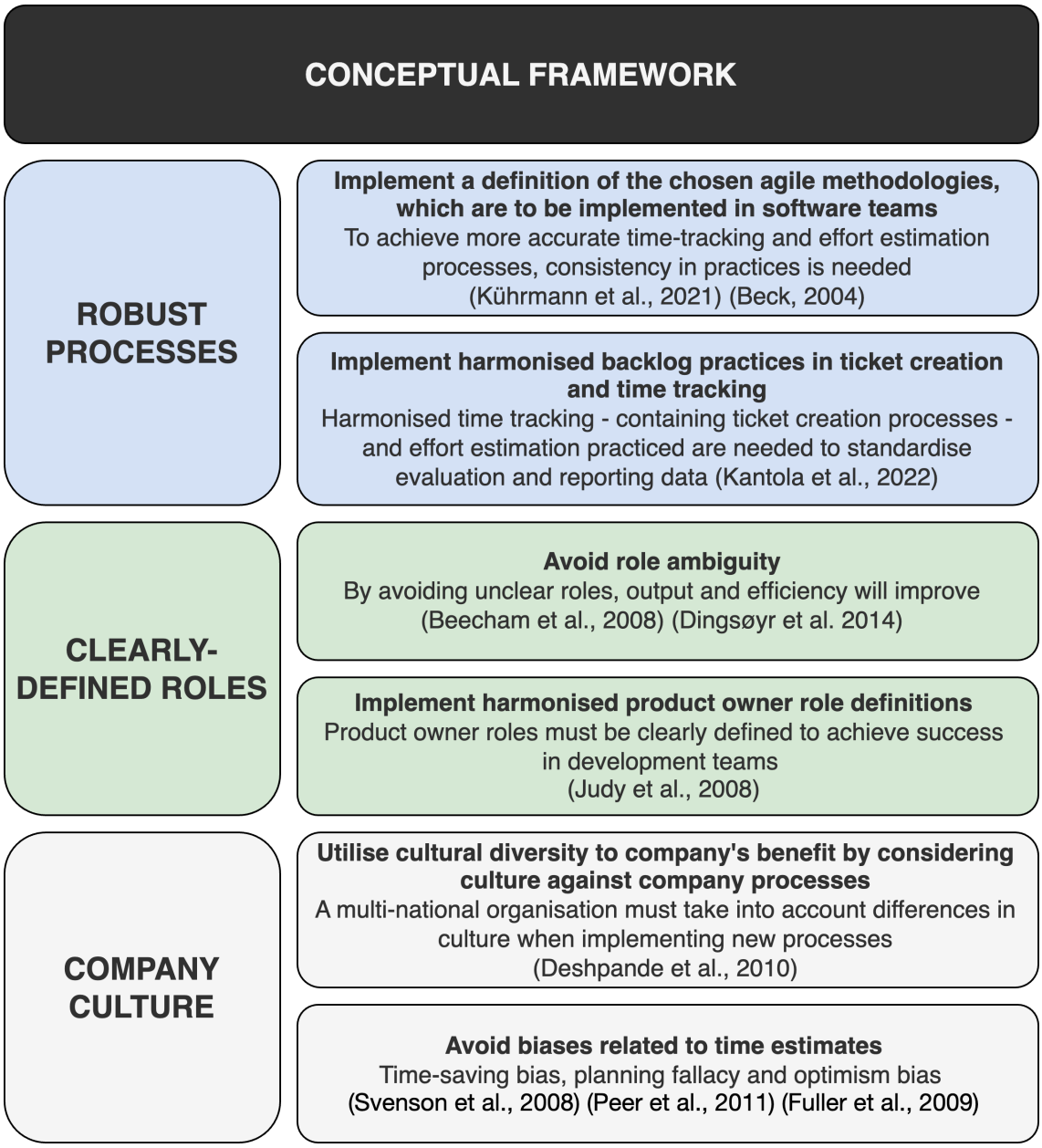

Figure 9: The conceptual framework of this thesis.

comprehensive foundation for fostering effective teamwork and project success in agile environments.

In Chapter 5, the conceptual framework is used to generate a proposal for the case organisation on how effort estimation and time-tracking practices may be improved. The conceptual framework is tailor-made to address the weaknesses of current practices.

5 Proposal of Improvements to Ground-Level Organisation Practices

Chapter 5 establishes the initial proposal to improve effort estimation and time-tracking practices within the case organisation. The proposal builds on the conceptual framework generated in Chapter 4, while incorporating input from internal company informants as detailed in Chapter 2.3.

5.1 Overview of Data 2 Collection

As stated in Figure 1, data which was used to generate the proposal is extracted from a co-creation workshop with the case company informant – the software component Lead APO – where the findings from the CSA in Chapter 3 and the conceptual framework from Chapter 4.6 is used as a basis point for discussion. The workshop consists of a face-to-face meeting where the current state analysis findings were discussed thoroughly and their implications were addressed.

The initial proposal aims to harmonise effort estimation and time-tracking processes to enhance overall efficiency and accuracy across the whole software component that is scoped within this thesis. This involves reinforcing existing protocols, addressing role ambiguities, and fostering a culture that supports the adherence to existing processes while also adapting potential new processes into the loop. The primary focus to improve upon the processes of effort estimation and time tracking is on ensuring that all hours are logged correctly and that information flow is optimised to prevent miscommunication and inefficiencies. Finally, in the context of time tracking, accuracy of logged effort during feature implementation will be improved.

5.2 Proposal for Improving Company Processes

Currently, the FS0EE and FS2EE practices are keenly followed and updated by project management who are also responsible for improving the process flows of

other software components in the development unit. However, there is significant room for improvement in the flow of information from project management to software teams and their individual software developers. During the co-creation workshop with the company informant, it was established that this rigidity of information flow is mainly a role-definition-related issue. Additionally, to harmonise the number of hours logged per day, clear instructions and process definitions are necessary. The co-creation workshop revealed that while set processes exist, they are neither enforced nor known by all relevant stakeholders, such as APOs and LPOs. For example, there is a requirement to log hours twice per week and to maintain three levels of backlog items for tasks (software component, epic, and story level), but these processes are not followed on all geographical sites. Furthermore, there is no cohesive process on how much hours per day should be logged. Admittedly, due to the division between backlog and non-backlog work, the line between what should be categorised as working hours to be logged is unclear to many teams.

To address these issues, it is crucial to clearly define and enforce the logging frequency of twice per week. As an output of the co-creation workshop, it is proposed that both frequency of logging and clear basic rules for amount of hours per unit time logged need to be introduced. Additionally, it is necessary to ensure that all backlog items are categorised into the three specified levels. This remedies issues in poor transparency to the details of feature implementation. Although, generally, all sites implement agreed features to a high standard, from a consistency point of view, traceability in Jira through story-level item listing would be beneficial if and when software bugs appear later in testing or customer reports.

Improving communication channels is also essential to ensure all stakeholders are aware of and adhere to these processes. This will help create a more consistent and transparent workflow, reducing discrepancies in logged hours and improving overall project management. In practice this means clear requirement for APOs to relay information from PO and L-APO and project management to SW teams. As each APO is responsible for one or more SW teams and each SW teams has their own team APO, this improvement would guarantee that all information is relayed.

In the current way of working, such requirement is not present and it is not clear whose responsibility it is. The information is assumed to be relayed by the APO but it could be relayed to LPO, who would further relay the information to the team adding one more possible node of failure. Having the APO be alone responsible for the information sharing would make it certain that information is shared.

5.3 Proposal for Improving Role Definitions

Literature and the co-creation workshop both highlight the pivotal role of product owners in funnelling information within a software organisation. As discussed in Chapter 4.4.2, in some organisations overworked POs may cause the development to grind to a halt to all intents and purposes. However, the case organisation of this thesis has opted for multi-level PO roles to distribute responsibility and workload, which has led to worsened information flow and unclear processes. This role ambiguity has been identified as a significant barrier to efficient communication and process adherence.

To mitigate these issues, it is important to address the abundance of processes that are currently not enforced. One potential solution is the implementation of a central process owner or a centralised process bank. The first solution would be a person who is the owner of a set of processes and that they are correctly implemented in a specific geographical location, software component or whole development unit. The second solution would be a company information source – such as a website or a document – containing xPO role related processes that require adhering to. This would ensure that all processes are documented, accessible, and enforced consistently across the organisation. Additionally, enhancing competence development through targeted, process-centric trainings, targeted to xPOs, would be beneficial. Such training would help LPOs, APOs and other stakeholders understand the importance of their roles and the processes they need to follow.

Lastly, regular reminders about the importance of clear communication and information flow are also necessary to reinforce these practices. This could, in

essence, signify improved onboarding to the role when a new xPO starts in their role.

5.4 Proposal for Improving Company Culture

Due to a lack of clarity in the time-tracking process, the tendency to fail to log time accurately often occurs, which leads to a feedback loop. In other words, less time is perceived as necessary for tasks, when new features are estimated in FS0 and FS2 stages, if historical effort data of older features is used as an input. To ensure all hours are logged, including those spent in technical meetings, developers need regular reminders of the importance of comprehensive time-tracking. As there is no one clear amount of time per unit time that should be marked by software developers, CSA uncovered that multiple practices exist. The simple proposal is to set a constant value for all developers within the software component to log.

Another aspect to be considered when renewing and reinforcing the company processes is as follows: given the organisation's multiple geographical sites and diverse cultures, any radical changes to processes should be carefully considered to avoid creating friction. It is essential to provide clear reasoning backed by data when proposing process changes to ensure buy-in from all stakeholders. Prioritisation of processes is, hence, required from higher project management.

Implementing a step-wise approach to process harmonisation is recommended to prevent abrupt transitions that could disrupt workflows. This gradual implementation would allow teams to adapt to new processes incrementally, reducing resistance and improving overall adoption. Furthermore, improving management culture by consistently explaining the rationale behind process updates and enforcing adherence to these processes is vital. Ensuring that all sites and teams are aligned is crucial. This alignment will help create a more unified and efficient organisation.

An additional finding during the workshop is that due to the geographical location differences between sites, the way new processes are adopted also differs across

the sites. Ensuring clear communication when introducing new processes and enforcing existing ones properly, would prevent inter-site differences from forming.

Category	Improvements
Robust Processes	Enhance information flow from project management.
	Standardise daily and weekly hour logging practices.
	Maintain three hierarchical levels for backlog items (CA, epic, story).
	Require developers to log hours directly into story-level items.
	Ensure APOs communicate new and existing processes to SW teams directly and effectively.
	Improve work visibility by including detailed descriptions and logged hours in backlog items.
	Strengthen communication by emphasising the importance of APO messaging.
	Enforce at minimum a twice-per-week logging schedule.
	Eliminate unnecessary processes and focus on streamlining and enforcing only the essential ones.
Clearly-defined Roles	Clearly define all roles within the SWC to avoid ambiguity.
	Clarify ownership of central processes.
	Prohibit logging of hours on behalf of another person/developer.
	Assign responsibility for logging hours to the developers who consume them.
	Provide role-specific training for xPOs when assuming new responsibilities.
	Define the APO role more clearly.
	Provide targeted training for APOs and LPOs on process compliance.
Company Culture	Foster a mindset among developers to log all work types, including coding, reviews, meetings, and discussions.
	Implement processes gradually, supported by clear reasoning.
	Increase awareness of biases and their impact on work through trainings.
	Introduce new processes incrementally to avoid overwhelming the team.

Table 4: Initial proposal to company processes in order to enhance time-tracking and effort estimation practices.

Table 4 contains the categorically separated initial proposal items. The categories are based on same categorisation as conceptual framework in Chapter 4: robust processes, clearly-defined roles and company culture. Each of the improvement proposals aims to improve either time-tracking or effort estimation practices.

5.5 Summary of the Initial Proposal

The proposed improvements focus on enhancing effort estimation and time-tracking accuracy by reinforcing existing protocols, addressing role ambiguities, and fostering a supportive culture. By clearly defining and enforcing processes, improving communication channels, and providing targeted training, the organisation can achieve more accurate time-tracking and effort estimation. The proposed changes aim to create a more cohesive and efficient workflow, ultimately leading to better project outcomes and improved overall efficiency. The main points of the initial proposal are summarised in Table 4. Within Table 4 the improvements proposed in the initial proposal are categorised into three categories in accordance to the conceptual framework.

In this chapter, the data from current state analysis in Chapter 3 and the conceptual framework generated in 4 were used to generate the initial proposal of Chapter 5 with informants from the case company. In Chapter 6, the initial proposal will be validated and improved.

6 Validation of Improvement Suggestions for Effort Estimation and Time Tracking Practices

In this chapter, the validation of the initial proposal is discussed. The discussion begins with an overview of the proposal validation process, continues into changes and refinements made during the validation stage, and concludes by presenting the list of final process steps designed to enhance the effort estimation and time-tracking processes.

6.1 Overview of Data 3 Collection

The implementation of the proposals in practice lies outside the scope of this thesis. This limitation is due to the restricted timeframe allocated for the writing process, coupled with the inherently slow pace at which large organisations are able to adapt and implement significant process changes. As an alternative to direct implementation, the proposal items were discussed with a key stakeholder: the software component Lead Area Product Owner (L-APO), to collect constructive and actionable feedback on the proposed improvements, as stated in Figure 1.

The validation process involved a combination of a face-to-face meeting and follow-up message exchanges with the L-APO. During the meeting, both the conceptual framework outlined in Chapter 4.6 and the initial proposals described in 5 were presented. This enabled the relevant stakeholder to review, evaluate, and comment on the feasibility and relevance of the proposals within the context of the software component’s processes. Subsequent message exchanges served to clarify any outstanding points of feedback, ensuring that refinements to the proposals were well-informed and targeted.

Overall, the feedback on the proposed improvements was positive, as the proposals were directly tailored to address the issues identified during the current state analysis (CSA). The company informant confirmed that similar issues had been observed internally, further validating the relevance of the findings. In

addition, suggestions for refinement and minor additions were incorporated into the final proposal.

6.2 Feedback on Improvement Proposals for Time-Tracking and Effort Estimation Practices

The feedback received on the initial proposal highlighted the overall strength and applicability of the suggested improvements. The care and thoroughness exercised during the co-creation workshop, detailed in Chapter 5, were reflected in the minimal number of corrections or significant updates needed. This reaffirmed the alignment of the initial proposal with the needs and expectations of the organisation.

Despite the positive reception, a few areas for refinement emerged during the validation process. Primarily, the wording of some proposed improvements was adjusted for clarity to ensure a shared understanding among stakeholders. Additionally, the final proposal was restructured for better readability and presented in a table format that summarised all suggested improvements, facilitating easier review and reference.

The technical enchancements to the improvements consist of highly encouraging re-estimation of tasks during feature development, requiring developers to log directly into story-level items and prohibiting logging of hours on behalf of another person or developer.

Category	Improvements
Robust Processes	Enhance information flow from project management.
	Standardise daily and weekly hour logging practices.
	Maintain three hierarchical levels for backlog items (CA, epic, story).
	Ensure story-level items include clear descriptions of work.
	Require developers to log hours directly into story-level items.
	Ensure APOs communicate new and existing processes to SW teams directly and effectively.
	Improve work visibility by including detailed descriptions and logged hours in backlog items.
	Strengthen communication by emphasising the importance of APO messaging.
	Encourage effort re-estimation during feature development.
	Enforce at minimum a twice-per-week logging schedule.
	Establish a process repository for easy reference.
	Eliminate unnecessary processes and focus on streamlining and enforcing only the essential ones.
Clearly-defined Roles	Clearly define all roles within the SWC to avoid ambiguity.
	Clarify ownership of central processes.
	Prohibit logging of hours on behalf of another person/developer.
	Assign responsibility for logging hours to the developers who consume them.
	Provide role-specific training for xPOs when assuming new responsibilities.
	Define the APO role more clearly in order to reduce the risk of information flow bottlenecks.
	Provide targeted training for APOs and LPOs on process compliance.
Company Culture	Foster a mindset among developers to log all work types, including coding, reviews, meetings, and discussions.
	Implement processes gradually, supported by clear reasoning.
	Increase awareness of biases and their impact on work through trainings.
	Introduce new processes incrementally to avoid overwhelming the team.
	Avoid the introduction of incomplete or insufficiently developed processes to ensure that all processes are fully enforced and followed.

Table 5: Updated proposal to company processes in order to enhance time-tracking and effort estimation practices after Validation phase. Updated and added improvements are highlighted in green.

Table 5 is based on Table 4 and contains the categorically separated final proposal items with the items highlighted in green denoting the changes that were made during the validation process. The categories are based on same categorisation as conceptual framework in Chapter 4: robust processes, clearly-defined roles and company culture. Each of the improvement proposals aims to improve either time-tracking or effort estimation practices.

6.3 Final Proposal

The final set of improvements aimed at enhancing time-tracking and effort estimation practices is presented in Table 5 with updates done during Validation highlighted in green. These improvements are categorised into three key areas, as identified in the literature review in Chapter 4: robust processes, clearly defined roles, and company culture. Each category reflects a critical aspect of the overarching improvement strategy, with the proposed changes working together to establish a more efficient and transparent framework for effort estimation and time tracking.

In this chapter, the initial proposal from Chapter 5 was used as input and refined into the final proposal through a co-creation workshop. The final chapter, which is presented next, outlines the conclusions of this study, proposes recommendations for next steps and concludes with a self-evaluation of the project.

7 Conclusions

This chapter is the final chapter of this study containing an executive summary with the relevant information regarding the business challenge, objective, current state analysis findings, conceptual framework and suggested improvements in prior chapters. The credibility of the study is discussed and the thesis concludes with closing words.

7.1 Executive Summary

The objective of this thesis was to build a proposal for improving the feature planning process to enhance planning accuracy. The suggestions were generated in order to align software teams' hour-reporting and planning practices, enabling better developer bookkeeping of used time resources and providing increased visibility for implemented software features. This improved transparency would harmonise the practices between the teams and provide unity in bookkeeping and estimation, which would form a solid basis for easier project management and improved software development through better predictability, more accurate planning and clearer information on implemented changes. As a consequence, more accurate future feature planning would be possible and understanding how much efforts past features consumed and what the code changes on a more detail-level were during implementation. In the first stage of the study, the focus of the thesis was decided to be on two dimensions of software feature planning: effort evaluation and developer time tracking.

In the second stage, the project plan was established and the research approach and design were presented with the inclusion of a data plan. This thesis has drawn from two primary sources of data: interviews with relevant personnel and internal company documentation. Interviews with area product owners and developers across multiple geographical sites served as the foundation for the current state analysis and the proposal's development. Internal documentation was utilised to uncover existing guidelines related to time-tracking and effort

evaluation practices. Product owners and developers were consulted during the current state analysis and proposal building steps to ensure its validity across the whole software component.

The third stage was the current state analysis, which revealed significant variability in time-tracking practices and effort evaluation across teams and sites. They differed in tracking frequency, ranging from daily to biweekly, and responsibility for time logging was noted to vary between developers and local product owners. Furthermore, discrepancies in practices, such as task-level refinement and the refinement of work into backlog and non-backlog categories, contribute to inaccuracies in time tracking and effort evaluation. Another key finding was that the amount of time logged per day was significantly different between teams, sites and other parts of the organisation causing at worst – seemingly – a 50-percent deviation in the output of the teams. Lastly, it was uncovered that in certain teams task-level refinement was not done as dictated by basic Scrum and Agile principles. While the flexibility in team practices and the structured process for effort evaluation were seen as strengths, these mentioned inconsistencies were analysed to be undermining planning accuracy. These findings provided the foundation for exploring literature-based improvements in the subsequent chapters.

In the fourth stage, the conceptual framework was built from relevant scientific literature available in the context of the current state analysis topics. The framework focused on three critical areas: robust processes, clearly-defined roles, and company culture. Robust processes included harmonised backlog practices and clear definitions of agile methodologies. Role clarity was emphasised to reduce ambiguity, with particular attention to product owner responsibilities. Company culture was addressed by integrating cultural diversity into processes and mitigating biases in effort estimates and planning. Together, these elements were used to create the conceptual framework, which became the second part of the foundation for the proposal generation. Other considerations examined during this stage were researching Agile methodologies and the impact of cognitive biases that might impact effort estimation and planning in a software organisation.

The fifth and penultimate stage was creating the proposal for the case organisation. The proposed improvements aimed to enhance effort estimation and time-tracking accuracy by strengthening existing processes, clarifying roles, and fostering a supportive culture. The proposal underlined a clear and enforceable set of practices, improved communication, and targeted training as key components. Specific recommendations included encouraging task re-estimation during development, requiring direct logging at the story level, and prohibiting proxy time logging. These changes were proposed in order to create a cohesive and efficient workflow, aiming to improve project outcomes through cohesion in time-tracking and effort evaluation practices.

In the sixth and final stage – the validation phase – the proposal was revisited with key stakeholders within the organisation. The proposal was positively received, though refinements were made. Key technical enhancements included encouraging re-estimation during development, requiring developers to log hours directly, and prohibiting logging on behalf of others. These refinements ensured alignment with stakeholder needs and supported the overall goal of improving planning accuracy. The validation was carried out as a combination of a meeting and discussions to find further constructive feedback which could be transformed into actionable items to improve the proposal. Lastly, based on the input, the final proposal was formed through the updates detailed above.

7.2 Recommendations for Next Steps

The next steps in implementing the proposed improvements involve agreeing on the implemented changes into the organisation systematically over time. Whether the organisation chooses to utilise all the improvement points is up to the project management and managers of the software component. Efforts are recommended to focus on refining processes, defining roles more precisely, and fostering a supportive company culture. Robust processes should be reinforced by improving information flow, clarifying roles, standardising practices, and ensuring consistent and detailed logging at all backlog levels.

Enhanced communication and targeted training for key roles, such as APOs and LPOs, would drive compliance and understanding of new and existing procedures. Clearly-defined roles would eliminate ambiguities, ensure proper ownership of tasks, and establish direct accountability for hour logging. A key aspect in introducing these changes is the gradual approach. The organisation should not update everything at once but either gradually over time. This approach in introducing process changes would help maintain team cohesion, supported by efforts to address biases and foster a culture of comprehensive work documentation. Through these steps, the organisation would build a strong foundation for sustained operational improvements.

Due to important nature of planning, the organisation is recommended to further look into the biases impacting project management and planning personnel. This thesis was not able to make a deep-dive, but rather concentrated on a few meaningful ones to show the impact in a planning setting. Cognitive biases have been researched for several decades in psychology and in the scope of this study, there was no possibility to study them widely and thoroughly to form a all-encompassing breakdown of each and every cognitive distortion affecting people’s ability to make accurate predictions in the context of planning work for a software organisation.

7.3 Self-Evaluation of Thesis Project Credibility

The thesis addresses a relevant business challenge, specifically the non-cohesive and inaccurate time-tracking and effort estimation practices. Feedback and the outcomes of the proposal support the relevance of the CSA findings. The selected literature is relevant to the conceptual framework, even though the limited availability of literature in software development settings posed some challenges. The solution is highly relevant to a specific department of the company, but its extension to other areas would require additional investigation and adaptation to specific practices.

The structure of the thesis is logical, with each chapter building upon the previous

one. There is clear progression from problem identification to the solution proposal. The project objectives were met, and the research design and outcomes align closely.

The evidence trail of the thesis is solid, and the choices made in building the thesis are grounded in a logical mindset. CSA findings and interpretations are based on sufficient data. However, the generation of the proposal could have been improved further through a broader audience. The limited availability of literature in software development settings slightly constrained the project, but the conceptual framework was successfully built from a less software-development-centric point of view.

The findings, solutions, and interpretations are data-driven and transparently documented, ensuring that similar results could be expected in other studies, albeit with some variations depending on context. Improvements to time-tracking and effort estimation practices are highly targeted, ensuring consistent outcomes. The practical value of the thesis was demonstrated, as some practices were adopted during the writing process itself, reinforcing its reliability.

The author’s proposal regarding the priority of the to-be-implemented improvements are focusing on harmonising time-tracking practices across teams and geographical sites. Following this, backlog practices of how work is tracked needs to be harmonised. Thirdly, during software development re-estimation of planned work needs to be focused on. Then, definition of roles and their responsibilities are to be clarified next in order to improve communication and information flow and lastly awareness of cognitive biases would be recommended to be investigated and awareness would be spread.

As stated by Patton, in qualitative research careful analysis is key, and Patton investigates enhancing the quality and credibility of qualitative analysis and lists three points [30, pp. 1189–1191].

1. Rigorous techniques and methods in gathering and analysing qualitative data through triangulation

2. Trustworthiness and competence of the qualitative researcher
3. Belief in the value of qualitative research, through purposeful data collection

Patton argues that triangulation is a required method in qualitative research as no one single method is adequate in solving or explaining rivaling explanations. In practices a combination of interviewing, observing and analysing documents are all needed methods in qualitative research as using only one of these methods would leave leave a study vulnerable to potential errors in conducting only a subset of the three mentioned methods [30, pp. 1192–1194].

In order to reach a credible outcome within this study, triangulation method was utilised. As explained in Chapter 2.1, the collected data combined interviewing relevant stakeholders, observing company practices and studying company's internal documents. Interviewees were chosen such that all geographical sites were covered and project organisation personnel were interviewed so that all 3 levels were covered: LPO, APO and L-APO.

Credibility could have been improved with a wider co-creation workshop when generating the proposal. Only the lead area product owner was consulted during this phase. In hindsight, the study might be benefitted if more voices were heard in this stage, but at least – if only one person was to be co-operated with – the best choice was taken.

In terms of the relevance of the study, the thesis addresses a relevant business challenge, the current state analysis was able to extract the problem areas within the case company, and the solution is relevant to the software component in which the study was conducted.

From logical point of the structure of the thesis follows a cohesive approach. The approach was instructed by Metropolia University of Applied Sciences and the approach focused on a logical and structurally valid setting for the thesis: defining the problem and an objective, understanding the problem and generating a framework to solve the problem, and finally creation of the solution and its

verification.

7.4 Closing words

Companies producing software for their customers must continuously find new ways to one-up the competition. In larger projects, the importance of planning accuracy becomes more pronounced, which is the reason that this thesis was written to remedy issues relating to issues in planning. The study conducted in this thesis outlines a solution containing multiple improvements to make planning of software development more accurate. Reaching such a practical objective and providing tangible improvement suggestions was very rewarding. I hope that – even though the generated proposal is as such only applicable to a relatively small, restricted department of the case company – the company is able to utilise the proposal and improve their processes accordingly and improve their software development output.

References

1 Calvert, Jane. 2006. “What’s special about basic research?” In: Science, Technology, & Human Values 31.2, pp. 199–220.

2 Rose, Susan; Spinks, Nigel & Canhoto, Ana Isabel. 2014. Management research: Applying the principles. Routledge.

3 Kananen, Jorma. 2013. “Design research (applied action research) as thesis research: A practical guide for thesis research”. In: JAMK University of Applied Sciences.

4 Holma, Harri; Toskala, Antti & Nakamura, Takehiro. 2020. 5G technology: 3GPP new radio. John Wiley & Sons.

5 Schwaber, Ken. 1997. “Scrum development process”. In: *Business Object Design and Implementation: OOPSLA’95 Workshop Proceedings 16 October 1995, Austin, Texas*. Springer, pp. 117–134.

6 Corporation, Atlassian. 2024. Jira project management tool description. Online. <https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software>. Visited on 04/13/2024.

7 Abrahamsson, Pekka; Salo, Outi; Ronkainen, Jussi & Warsta, Juhani. 2017. “Agile software development methods: Review and analysis”. In: arXiv preprint arXiv:1709.08439.

8 Bass, Julian M. 2015. “How product owner teams scale agile methods to large distributed enterprises”. In: Empirical software engineering 20, pp. 1525–1557.

9 Manifesto, Agile. 2001. “Manifesto for agile software development”. In.

10 Kantola, Konsta; Vanhanen, Jari & Tolvanen, Jussi. 2022. “Mind the product owner: An action research project into agile release planning”. In: Information and Software Technology 147, p. 106900.

11 Schwaber, Ken & Sutherland, Jeff. 2011. “The scrum guide”. In: Scrum Alliance 21.1, pp. 1–38.

12 BECK, K Andres C. 2004. Extreme Programming Ex-plained. Ed.

13 Kuhrmann, Marco; Tell, Paolo; Hebig, Regina; Klünder, Jil; Münch, Jürgen; Linssen, Oliver; Pfahl, Dietmar; Felderer, Michael; Prause, Christian R; MacDonell, Stephen G, et al. 2021. “What makes agile software development agile?” In: IEEE transactions on software engineering 48.9, pp. 3523–3539.

14 Deshpande, Sadhana; Richardson, Ita; Casey, Valentine & Beecham, Sarah. 2010. “Culture in global software development-a

weakness or strength?" In: *2010 5th IEEE International Conference on Global Software Engineering*. IEEE, pp. 67–76.

15 Hackman, J Richard & Oldham, Greg R. 1975. "Development of the job diagnostic survey." In: Journal of Applied psychology 60.2, p. 159.

16 Beecham, Sarah; Baddoo, Nathan; Hall, Tracy; Robinson, Hugh & Sharp, Helen. 2008. "Motivation in Software Engineering: A systematic literature review". In: Information and software technology 50.9-10, pp. 860–878.

17 Dybå, Tore; Dingsøyr, Torgeir & Moe, Nils Brede. 2014. "Agile project management". In: Software project management in a changing world, pp. 277–300.

18 Sverrisdottir, Hrafnhildur Sif; Ingason, Helgi Thor & Jonasson, Haukur Ingi. 2014. "The role of the product owner in scrum-comparison between theory and practices". In: Procedia-Social and Behavioral Sciences 119, pp. 257–267.

19 Judy, Ken H & Krumins-Beens, Ilio. 2008. "Great scrums need great product owners: Unbounded collaboration and collective product ownership". In: *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*. IEEE, pp. 462–462.

20 Svenson, Ola. 2008. "Decisions among time saving options: When intuition is strong and wrong". In: Acta psychologica 127.2, pp. 501–509.

21 Peer, Eyal. 2011. "The time-saving bias, speed choices and driving behavior". In: Transportation Research Part F: Traffic Psychology and Behaviour 14.6, pp. 543–554.

22 Fuller, R; Gormley, M; Stradling, S; Broughton, Patricia; Kinnear, N; O'Dolan, C & Hannigan, B. 2009. "Impact of speed change on estimated journey time: Failure of drivers to appreciate relevance of initial speed". In: Accident Analysis & Prevention 41.1, pp. 10–14.

23 Tversky, Amos & Kahneman, Daniel. 1974. "Judgment under Uncertainty: Heuristics and Biases: Biases in judgments reveal some heuristics of thinking under uncertainty." In: science 185.4157, pp. 1124–1131.

24 Buehler, Roger; Griffin, Dale & Ross, Michael. 1994. "Exploring the" planning fallacy": Why people underestimate their task completion times." In: Journal of personality and social psychology 67.3, p. 366.

25 Shmueli, Ofira; Pliskin, Nava & Fink, Lior. 2016. "Can the outside-view approach improve planning decisions in software development projects?" In: Information Systems Journal 26.4, pp. 395–418.

26 Weinstein, Neil D. 1980. “Unrealistic optimism about future life events.” In: Journal of personality and social psychology 39.5, p. 806.

27 Prater, James; Kirytopoulos, Konstantinos & Ma, Tony. 2017. “Optimism bias within the project management context: A systematic quantitative literature review”. In: International Journal of Managing Projects in Business 10.2, pp. 370–385.

28 Flyvbjerg, Bent. 2008. “Curbing optimism bias and strategic misrepresentation in planning: Reference class forecasting in practice”. In: European planning studies 16.1, pp. 3–21.

29 Lovallo, Dan & Kahneman, Daniel. 2003. “Delusions of success”. In: Harvard business review 81.7, pp. 56–63.

30 Patton, Michael Quinn. 1999. “Enhancing the quality and credibility of qualitative analysis.” In: Health services research 34.5 Pt 2, p. 1189.

1 Interview Questions of the Current State Analysis

Following questions were asked from the interviewees in the current state analysis stage of the thesis.

1. Why is time tracking conducted in the case organisation?
2. Why is effort evaluation conducted in the case organisation?
3. Who is responsible for driving time tracking?
4. Are you aware of any official guidance how much time should be logged per day/week/sprint?
5. Is logging of hours consistent in your team?
6. Basic rule of thumb is to log 6 hours per developer per day. Is this a good starting point in your opinion?
7. Have you heard of this being the official guidance?
8. What is the resolution of time-tracking: full hours, half hours, quarter hours or something else?
9. How often do developers in your team log hours?
10. What kind of practices have you seen?
11. Do you need to remind SW developers to log hours?
12. Are effort estimations accurate currently?
13. How could this process be developed in your opinion?

Additionally, free-form discussion was had concerning similar topics that came up naturally in the interview process. The angle of some question varied slightly depending on what role the interviewee has. Not all questions were relevant to all roles.

2 Interviewee Replies from Data Collection

Following are transcribed and simplified answers from the interviewee process. The transcribed contents are not word-to-word as the interview was not recorded down to each word. The answers are based on notes that have been stylised.

2.1 Interviewee #1

In Interviewee #1's department and teams, there are various approaches to time-tracking: in some teams, events are recorded per day, in others per week or even per sprint. It was reported that the interviewee's view is that daily logging is best. In Interviewee #1's department, time-tracking was earlier harmonised to be 6 hours per day. Since then time-tracking has been on sufficient enough level in all teams. This was not always clear before. In some departments, some developers log 5 hours per day while others log 7 hours. Now it is more consistent, and the improvement is clearly visible. It seems clear that teams should be harmonised in their time-tracking practices. From time to time, reminders need to be done by the LPO.

In terms of effort evaluation, after a feature is completed, it is necessary to check if there is any deviation between estimated and recorded efforts. Mostly, effort is underestimated, the reason being that there is always too much optimism in the plans. An expert gave FS2EE, but when it came to ”normal” developers, the hours used were significantly underestimated. Communication and management create overheard in this process. Features are often underestimated. There is also a need to balance work between teams. Software work should be assigned according to capacity. A high-level view is missing from long-term plan, it seems. The long-term view is the most problematic one.

2.2 Interviewee #2

Interviewee #2 discussed the resource management tool and the progress of planned work. They noted that there are very different practices on different sites. Some teams do not engage in story-level item creation. Essentially meaning that work logs correspond to very simple work descriptions. Some developers do not log their activities as required. It is unclear who is responsible for ensuring that guidelines are followed. Previously, it was more evident that these guidelines were adhered to, but with the rapid growth of the organisation, adherence at higher organisational levels has diminished. There is a concern that efforts are being logged less frequently than they should be. Although a guideline exists, it is unclear where the guideline is recorded, and there is no culture of properly advertising it.

The deviation between estimated effort and actual spending is mainly due to overly optimistic estimates or additional work that has not been taken into account in feature screening phase. Many hours are not logged accurately. A concrete issue with planning is that the feedback time of tools is excessively high, which should be improved as it disincentivises stakeholders from doing proper planning. Business decisions rely on this information, and logging activities also serve as a protection mechanism for teams.

2.3 Interviewee #3

The approaches seem quite harmonised, with 65 % of the time dedicated to development and fault management. This equates to 40 hours multiplied by 0.65. Logging is done by LPOs, who also bear the responsibility. At least once every sprint time-tracking is performed. Some sites do it every day, but here it is done once per sprint. The understanding in Interviewee #3’s department is that a guideline of 6 hours per day exists as there is recollection of such process. It is not clear where, though. This is considered a reasonable approach.

In general, it seems that time-tracking does not need to be considered too heavily

on this site. The question arises: how can LPO log for developers? The answer depends on the situation. There are 4 main development teams, and APOs do not interfere with their way of working; it is up to the LPO to decide. The Scrum master is also a contact point for information. It is important to keep track of fault management hours, and capacity is reserved for fault management work. Additionally, Interviewee #3's site wants to know the exact amount of hours used for fault management work.

There is a separate Jira project for EE evaluation and fault management, which has been in use for some years. Not everyone thinks it is useful, but personally to Interviewee #3, it is likable. The company system that visualises time-tracking and effort estimation data is not an optimal system. While it works, it has flaws. One major con is the delay, which is a big issue for planning as it can take multiple days to reflect. However, it is good that this tool really combines the data, but the way of implementation and introducing changes is problematic. The format is not user-friendly, and it is not trustworthy due to time-delay before changes are showed in the tooling.

2.4 Interviewee #4

Interviewee #4 mentioned that the official working hours – that are logged – are 30 hours per week, and this is adhered to. However, teams now create their own rules. The guidance that was provided by the interviewees came as a surprise (6 hours per day). Interviewee #4 found that this should be harmonised.

The interviewee has worked in other departments as well. At the beginning of the previous project, the enforcement of rules began. Although the practices were mostly similar, there were differences in activeness. Interviewee #4 finds the company system that visualises time-tracking and effort estimation data not particularly useful on a personal level, as it is mainly used for reporting. It requires effort to keep the data updated. The engineering of estimations for capacity is likened to a ”weather forecast - and then it gets changed retroactively,” leading to information being lost. Interviewee #4 also mentioned the use of a separate Excel

sheet and noted that time-tracking should be completed by Monday afternoon before the Sprint exit.

2.5 Interviewee #5

Interviewee #5 mentioned that they are not so familiar with the topic and rely on expertise of effort estimation. There is a deviation from original estimates, and historical data from previous features is used to help with effort estimation.

The responsibility to log hours falls on the LPO, with one LPO per team. Each team LPO does the time-tracking, as asking each individual to do this would cause overhead. LPO logging is the preference for Interviewee #5, but both ways have their benefits and drawbacks. The granularity of stories is coarser, and hours are logged once a sprint or at most once a week. There is a problem with tooling, as there are many emails indicating that Jira data is not up-to-date. In Interviewee #5's teams/site, logged hours should be at least 5 hours a day. There has not been seen guidance, but approaches have been harmonised. The APO does not need to follow this. The company system that visualises time-tracking and effort estimation data is considered a good tool for decision-makers, but accurate data is not in Jira. Features are not yet assigned to teams, and a minimum of 6 months should be the standard to make load balancing easier. Effort estimation data is inaccurate, with usually too little effort estimated for features. Estimation accuracy is key and needs to be good.