Final thesis

Knowledge management and throughput optimization in large-scale software development

by

Henrik Andersson

LIU-IDA/LITH-EX-A–15/025–SE

2015-06-22
Final thesis

Knowledge management and throughput optimization in large-scale software development

by

Henrik Andersson

LIU-IDA/LITH-EX-A–15/025–SE
2015-06-22

Supervisor: Kristian Sandahl, Linköpings universitet
Helena Gällerdal

Examiner: Johannes Schmidt, Linköpings universitet
Abstract

Large-scale software development companies delivering market-driven products have introduced agile methodologies as the way of working to a big extent. Even though there are many benefits with an agile way of working, problems occur when scaling agile because of the increased complexity. One explicit problem area is to evolve deep product knowledge, which is a domain specific knowledge that cannot be developed anywhere else but at the specific workplace. This research aims to identify impediments for developing domain specific knowledge and provide solutions to overcome these challenges in order to optimize knowledge growth and throughput.

The result of the research shows that impediments occur in four different categories, based on a framework for knowledge sharing drivers. These are people-related, task-related, structure-related and technology-related. The challenging element with knowledge growth is to integrate the training into the feature development process, without affecting the feature throughput negatively.

The research also shows that by increasing the knowledge sharing, the competence level of the whole organization can be increased, and thereby be beneficial from many perspectives, such as feature-throughput and code quality.
Acknowledgement

This report is a master thesis at the program Master of Science in Information Technology at the Institute of Technology at Linköping University. The study was conducted at a software development company in Linköping.

I would like to thank all people involved in the study, including all employees at the department at the company, the interviewees and especially my supervisor Helena Gällerdal. I would also like to thank my supervisor Kristian Sandahl and my examiner Johannes Schmidt at the Department of Computer and Information Science for their time and support.

Henrik Andersson
Linköping, June 2015
## Contents

1 Introduction  
  1.1 Background  
  1.2 Metrics and internal investigation  
    1.2.1 Numetrics  
    1.2.2 Internal investigation  
  1.3 Aim of study  
    1.3.1 Problem definition  
  1.4 Limitations  

2 Theory  
  2.1 Earlier studies at the company  
    2.1.1 Agile in large-scale  
    2.1.2 Ways of working with knowledge sharing  
    2.1.3 Sustaining and developing expertise in project-based organizations  
  2.2 Related work  
    2.2.1 Agile in large-scale  
    2.2.2 Building knowledge in cross-functional teams  
    2.2.3 Complexity  

3 Method - Case study design  
  3.1 Case study research  
  3.2 Rationale behind chosen method  
  3.3 Research questions  
  3.4 Data collection  
    3.4.1 Interviews  
  3.5 Data analysis procedure  
    3.5.1 Interview recording and summary writing  
    3.5.2 Coding the summaries  
    3.5.3 Weighting the outcome  
  3.6 Validity procedure  
    3.6.1 Construct validity  
    3.6.2 Internal validity
3.6.3 External validity ............................................. 23
3.6.4 Reliability .................................................. 23
3.7 Conducting research ethically .............................. 23

4 Case study ...................................................... 24
  4.1 The organization ........................................... 24
  4.2 Initial focus group meeting .............................. 25
  4.3 Pre-study .................................................... 25
  4.4 Data collection and analysis ............................. 25
  4.5 Second focus group meeting ............................. 27
  4.6 Third focus group meeting ............................... 28
  4.7 Classification into frameworks .......................... 28

5 Results and analysis .......................................... 30
  5.1 The organization ........................................... 30
  5.2 Result of first degree data collection .................. 30
    5.2.1 Impediments .......................................... 31
    5.2.2 Prioritized Impediments ............................. 34
    5.2.3 Current state ........................................ 35
    5.2.4 Knowledge-driver impediments ..................... 38
    5.2.5 Solutions ............................................. 45
  5.3 Suggestions for improvements ........................... 50
    5.3.1 People-related drivers .............................. 50
    5.3.2 Structure-related drivers ........................... 51
    5.3.3 Task-related drivers ............................... 52
    5.3.4 Technology-related drivers ....................... 52

6 Discussion ..................................................... 54
  6.1 Discussion ................................................ 54
  6.2 Evaluation of validity ................................... 55
    6.2.1 Construct validity .................................. 55
    6.2.2 Internal validity .................................... 56
    6.2.3 External validity .................................... 56
    6.2.4 Reliability .......................................... 57
  6.3 Ethical evaluation ........................................ 57

7 Conclusions ................................................... 58
  7.1 Conclusions ................................................ 58
  7.2 Future work ............................................... 59

A Questionnaire - Team members .............................. 65
B Questionnaire - Supporting roles .......................... 68
C Introduction letter ......................................... 71
D Interview introduction ...................................... 73
Chapter 1

Introduction

Today’s competitive environment with fast-paced development cycles and short time-to-market require companies to be flexible in their work, especially when the development process is market-driven (Karlsson et al., 2007). Agile methodologies have been a well-established truth as an approach to be flexible and deliver high-quality software. Agile methodologies are based on the Agile Manifesto\footnote{http://www.agilemanifesto.org} from 2001. The manifesto is addressed to software development in small teams and not directly applicable for large-scale developing firms. Research has shown that many of the supposed advantages with adopting an agile approach entails problems for large-scale firms, where it is a high overlap between the supposed benefits and problem areas (Petersen and Wohlin, 2009). In other words it seems like it is a matter of processes, where the processes do not work in the same way as in smaller companies. An example is when you have developers in several countries all over the world working on the same product, you can not stick to the agile manifesto to one hundred percent since the manifesto states that co-located teams and verbal interaction comes over documentation, processes and tools. One of the reasons behind that is fast knowledge sharing between the competences, which obviously is not always possible in companies with developing teams located all over the world, where the verbal interaction is limited. This leads to the difficulties with knowledge sharing in large organizations. The knowledge is further twofold. One part is the knowledge sharing in the cross-functional teams. This work setting tends to influence the developers in the teams to go broader in their knowledge because the team members have different expertise and are affecting each other (Bredin and Söderlund, 2011). The other part is to develop deep understanding about the product, which is developed by working with the product and its different subsystems for a long-time.
1.1. BACKGROUND

This problem is related to difficulties in adopting agile in large-scale because of the complexity, due to product size, number of people involved and number of parallel projects. The definition of large-scale is somewhat diverse. Moe and Dingsøyr (2014) tried to summarize and conclude a definition from practitioners. No definition is currently widely established but many parameters are discussed, such as over 50 developers and over 5 teams as well as coordination can’t be achieved from one Scrum Board and technical perspectives as larger and more complex products than a few teams can handle. However, in this research all those parameters are fulfilled and we can content ourselves with large-scale software development as an organization where the agile manifesto is not directly applicable.

The perspective of large-scale can be divided into two separate categories, a technical perspective and an organizational perspective. The technical perspective can be further divided into a domain specific competence area and a general technical competence area. General technical competence is related to knowledge you can gain anywhere, by working with similar tasks or techniques, in this case by working as a software developer. The domain specific area is where knowledge about the product and its different sub-systems are central. This is something you can’t improve anywhere but at the specific work-place. The essential aspect here is that team members’ knowledge is dependent on which type of task they get and where in the system they are working. Research has also shown that different organizational settings highly influence the competence development, where multi-project environments increase the risk of extreme workload which results in decreased time for reflection, learning and recuperation (Zika-Viktorsson et al., 2006).

1.1 Background

The company in the study is a software development company that has adapted to an organizational setting where feature-driven development is performed by cross-functional teams. The declared strategy by corporate level is to work according to Lean with an agile methodology in every cross-functional team. Lean is aimed to simplify and make clarity in the production where complex operations with many products and actors are a natural part of the daily work. The important factors in a Lean organization is a mindset of focusing on optimization for customer value, meanwhile finding a good balance to resource allocation in terms of not sub-optimising. (Modig and Åhlström, 2013) This transformation into Lean and Agile has not come without problems, because of the size of the organization as well as the complexity of the products. The department under investigation is responsible for one product in the corporate product
portfolio. The department is further divided into several units responsible for different subsystems of the product. They have identified that deep knowledge about the product is a major factor that influence the output of their projects, i.e. new released features. This is due to the complexity of the product with several subsystems that can be argued to be categorized at large-scale on their own (Moe and Dingsøyr, 2014). Every team is co-located but there are teams located at different sites in three countries. The purpose behind the cross-functional teams is that every team should be able to take responsibility for the whole development cycle of a feature.

1.2 Metrics and internal investigation

This subsection describes findings from internal metrics and investigations at the company. These can be used as hands on evidence for the current state of the organization.

1.2.1 Numetrics

Numetrics\(^2\) is an analytical tool for performance rating and benchmarking for software development firms. The tool is owned by McKinsey & Company and provides companies developing embedded software the possibility to apply analytics-based decision-making and estimations to improve productivity. The main output of the analytical tool is effort as a calculation of complexity divided by man hours. The exact algorithm is confidential but the different input data is known. The design complexity model is calculated from the technical characteristics of the product and includes variables such as lines of code, implemented requirements, test cases, programming language and type of middleware and hardware. This calculation gives a design complexity rating which can be used for different kinds of analyses.

Figure 1.1 shows the result from the complexity rating of delivered requirements to customers in 2014. As the tendency is that the effort increases when the complexity increases, the indication is that the productivity is dependent on the complexity in the product. Hence, the product knowledge is essential for productivity and to stay competitive.

Figure 1.2 shows the productivity of delivered customer requirements as a calculation of complexity. The result shows that the productivity tended to increase during 2014. No statistical significant relationship could be found between project size and effort needed, which further implies that the complexity in the product is a more accurate measurement for productivity than the size of a project.

\(^2\)http://www.mckinsey.com/client_service/semiconductors/tools_and_solutions
1.2. METRICS AND INTERNAL INVESTIGATION

Figure 1.1: Complexity Measurement

Figure 1.2: Productivity Measurement
1.2.2 Internal investigation

When feature development exceeds budget in terms of time and cost, the teams can demand a cost analysis on their performance. This subsection reviews the conclusion drawn from such an analysis as well as the implications. In this case there was a relatively small feature implementation which constitutes a good example of possible impediments in the feature flow.

Cost estimate

In this case, a relatively small feature should be implemented. The necessary competences were implementation of the feature and testing the feature in the system.

Cost analysis

Because of the way of working with cross-functional teams as the smallest building block, the feature was allocated to a full team which didn’t have any prior experience within the area.

The insufficient product knowledge about the area in the team demanded for external support which was not provided. The supposed contact for knowledge support in technical solutions was overloaded and couldn’t respond quickly. As an example the team spent much of their time on meetings, for discussing possible solutions for implementing the feature. A team at necessary competence level wouldn’t need any meetings. Anyway, the meetings are dictated by the established way of working, why it could be easily realized that there are problems with small features allocated to full teams.

The delivery mechanism creates a high cost because of technical dependencies in the system. Even small features require comprehensive testing and delivery to main track is cumbersome. Testing experience was insufficient in the team.

Implications

The outcome of the feature development process was that it exceeded budget several times. The implication is that a competence mismatch is expensive and affects many parts of the process which causes delays in the lead time. The lesson learned is that when a team without the right competence is allocated to a feature, the knowledge support must be provided in a decent way. Similar cost issues can be seen in other features
but becomes more visible in small features like this. One further implication is that product knowledge is related to task distribution and resource allocation, where the latter includes both team composition and external support. To allow for better feature-flow, investigations of how task distribution and resource allocation are handled is needed. Further, investigations of how these parameters affect the product knowledge and how that contributes to decreased lead time is an important concern.

1.3 Aim of study

The research aims to identify impediments with developing and sustaining deep product knowledge in the cross-functional teams. The impediments found are further investigated in order to find solutions that can improve the work execution to allow for deep product knowledge.

1.3.1 Problem definition

The study is addressed to answer these research questions:

- How can a large software development company secure deep product knowledge and long-term evolution in a feature-driven work environment with cross-functional teams?
  - How can the tasks best be distributed to the teams to secure product knowledge in the organization?
    - And also, how can it be ensured that the organization has knowledge within the whole product?
  - How can resource allocation of specialized capabilities be handled to secure feature growth and long-term product knowledge?

1.4 Limitations

The research is limited to only investigate teams working in a subsystem in one product of the corporate product portfolio. Hence, the product knowledge for the interviewees is limited to the subsystem and its surrounding interfaces. One further demarcation is that the study only collects primary data from team members and supporting roles to the teams from one of the departments.
Chapter 2

Theory

This section summarizes earlier research and will work as a basis for the analysis of the case study. The section is divided into two subsections covering different types of theory, which together will form the theoretical framework. The first subsection reviews earlier research conducted at the company in related areas. The second subsection outlines related theory and state of the art of the studied field of research.

2.1 Earlier studies at the company

Several studies have been conducted at the company, which are of relevance for this study. They cover topics as agile in large-scale and human resource management in cross-functional teams. A summary of the findings are presented in the following subsections.

2.1.1 Agile in large-scale

Petersen and Wohlin (2009) investigated how the state of the art about benefits and challenges in agile methods were applicable in large-scale software development. Petersen and Wohlin (2009) conducted a case study and compared the result with a recent literature review by Dybå and Dingsøyr (2008), which reviewed the current state of the art of agile methodologies in terms of supposed advantages and issues. Many of the supposed benefits from using agile was true also in large-scale software development, but they emphasize that while using small and coherent teams increase the control of a project, it raises new issues in coordination by the management level. They didn’t find any new advantages with agile development in large-scale, in comparison with the state of the art.
2.1. EARLIER STUDIES AT THE COMPANY

However, they were able to find that new issues arose. These issues could all be categorized as related to increased complexity when scaling agile.

Lagerberg et al. (2013) build their research upon earlier reported benefits from agile practices in software development. The research aims to contribute with empirical evidence of how agile practices impact large-scale software development, through a holistic multiple-case study comparing two projects; one with a classical waterfall methodology (except from a few agile practices in some teams) and one using a holistic agile methodology. The empirical data was compared to the findings from a systematic literature review which resulted in six focus areas of comparison. One of those is knowledge sharing, where no statistically significant difference were found between the two projects in necessary capabilities inside the teams to help each other and how much the project members got external support. What could be proven to be a statistical difference was that the implemented agile practices contributed to knowledge sharing. The agile practices were iteration planning meetings, retrospectives, and demos. The research also found that knowledge sharing between different functional roles could be statistically proven to be larger in the project where an agile methodology was used, which implicate that the team members are more likely to be able to contribute to one and others work.

In addition Lagerberg et al. (2013) reported that agile principles can be beneficial for large-scale software development firms even when agile is just partially implemented. One contradiction to prior research was also found; that documentation is important and cannot be fully replaced by verbal interaction in large-scale agile software development. The documentation is important mainly because it contributes to knowledge sharing between distributed sites.

Sekitoleko et al. (2014) investigated challenges with technical dependencies and their communication when agile practices are applied in large-scale software development. They found that the technical challenges in terms of interdependencies among both activities and artifacts have a major impact on the overall performance in large-scale software development. What is even more challenging is that there are two types of technical dependencies. Planned technical dependencies are those which are identified during the planning phase of a feature. Unplanned technical dependencies are those which occur during the implementation phase of a feature. Altogether they found five main challenges from their case study, related to the technical dependencies in large-scale agile software development; planning, task prioritization, knowledge sharing, code quality, and integration.

They also concluded that these challenges relate to each other to an extent that if one challenge is handled badly, it can affect all the other challenges negatively and create a vicious circle. What’s on the positive side though,
is that they found it possible to contribute to all challenges by starting to mitigate one of them. The researchers suggest that knowledge sharing is a good starting point for this and will enable companies which struggle with agile adoption to achieve the intended benefits.

The identified challenges provides a framework which can be used as an inventory of the current state of an organization to identify bottlenecks and that way focus resources to break the vicious circle. The occurrence of a challenge is also highly dependent on the presence of the other challenges. To exemplify this, a scenario is presented with its origin in the planning challenge.

To identify technical dependencies in software development would need a perfect plan. Such an optimal plan is not possible to create. However managers try to identify dependencies at a high level to distribute tasks to the teams. This planning challenge leads to unplanned dependencies that occur during the development process. The teams may then have to reprioritize their backlog to implement a component that another team is dependent on, or have to wait with implementing certain functionality until another team has delivered what they are dependent on. This task prioritization challenge contributes to extensive knowledge sharing between the teams to enable collaboration. The knowledge sharing challenge relates to difficulties such as bad teachers, laziness, lack of communicativeness and overloaded experts. This is a time-consuming challenge which often leads to hasty decision-making in the implementation, and thereby contributes to bad code quality. This is often because of the developers just trying to solve the dependency conflicts rather than focusing on maintaining good quality in the entire product. The code quality challenge then directly affect the integration challenge towards delivery to customer when the different features should be tested and merged into a latest system version. The result of this challenge is often delivery stops to the main branch, which in turn leads to more merge conflicts because of the isolation in different branches. (Sekitoleko et al., 2014)

2.1.2 Ways of working with knowledge sharing

Davies and Brady (2000) contribute to research about building knowledge in organizations developing complex products and systems (CoPS) by explaining how they can build necessary capabilities to compete successfully in new products and services. They exemplify CoPS with high-technology and high-value capital goods such as telecommunication systems. The research is built upon a conceptual framework from Chandler (1990) who highlights that knowledge sharing between different organizational levels is essential to be competitive in terms of competence. Chandler (1990) emphasize that strategic capabilities must influence the
2.1. EARLIER STUDIES AT THE COMPANY

Functional capabilities to build the right knowledge that is in line with the organization's overall objective. Davies and Brady (2000) found out that this framework is obsolete, especially in project-based organizations where the project activities are essential for developing CoPS. They modified the framework to also include project capabilities, so that the necessary knowledge development can be integrated into the daily work. This will encourage both cross-functional and cross-project learning and in time lead to more effective organizational learning.

Moe et al. (2014) studied a development project at a software development company distributed across four development locations in three countries. The research focuses on how a newly introduced role, the technical area responsible (TAR) could be useful to support teams with problem solving, ensure knowledge sharing and safeguard the quality in the product. Moe et al. (2014) further describe that a knowledge network is essential in a large evolving system development project where volatility is high on both the organizational level and the technical level. Organizational volatility relates to new employees, new teams and assigning new tasks. Technical volatility is described by assigning new components and adding new dependencies in the system, and also by performing major refactoring. Both these challenges require access to people's knowledge, where information sharing is vital.

The research is based on the organizational settings where cross-functional teams should take the full responsibility for a feature. However they realized that this is not feasible without adding external support to the teams, where the support consists of experts with certain knowledge within the product. These experts are formally called TAR. The TAR is usually a senior developer that works halftime or full-time in the role. From their research they could conclude that the necessary knowledge support in the teams are highly dependent on the team's knowledge network, which in turn is dependent on how long the team members have been working in the company. Another finding was that the availability of local TARs was usually sufficient at the sites but there was a lack of availability of TARs between the different sites. In addition they found that the TARs are often overloaded with work, which implies that the amount of necessary knowledge sharing is very high in these kinds of organizations. (Moe et al., 2014)

Moe et al. (2014) state that large-scale software development is associated with an inability for everybody to know everything. When the current state of knowledge within a team is not sufficient for completing a task, they need external support. Knowledge networks are essential for finding the right people with the necessary competence, where formal roles, such as TARs, are very important. This is especially true when teams are assigned to tasks outside their current area of expertise. The research finally concludes that the role of the TAR could play an important role both in other departments at the company and in other large-scale software projects outside the own
organization. (Moe et al., 2014)

2.1.3 Sustaining and developing expertise in project-based organizations

Enberg and Bredin (2013) performed a case study at a software development company regarding developing disciplinary expertise in project-based organizations using co-located interdisciplinary project teams. The study was conducted when this organizational setup was recently introduced to the department. The reason behind the setup was that a cross-functional team should be able to drive the feature development from start to release, where a feature is new functionality added to the existing product and systems. The disciplinary competence in the cross-functional teams cover competences from the formal organizational setting where system design, design and testing were located at different divisions. However, the interviewees in that study state that the individuals in a team should contribute to the teams overall performance no matter of disciplinary expertise they formerly belonged to.

Even though the main focus of the analysis investigates disciplinary expertise, the product knowledge is covered as well, and in addition considered the most problematic competence to develop. The two types of knowledge are described as interrelated because when you are working with problem solving, experts within a discipline use both their disciplinary knowledge and their product knowledge about different subsystems. What could be concluded from this research is that many teams suffer from the impediment with a lack of product knowledge needed to develop a feature, especially within some subsystems of the product. The research shows that the cross-functionality of a team is true regarding disciplinary knowledge but it is not true with respect to the whole product. Although, the product knowledge is considered extremely important when it comes to the teams’ performances, but at the same time it would be impossible for a team to develop deep knowledge in every subsystem because of the size of the product. Anyway, the objective with the cross-functional teams is that any team should be able to develop any feature regardless of which subsystems they impact. This is an implication that the product knowledge in the teams must be broaden from the current state of knowledge. (Enberg and Bredin, 2013)

Enberg and Bredin (2013) highlight that structural and activity-based solutions aimed to sustain and develop disciplinary expertise should be vertically integrated into the daily work, i.e. by the functional dimension of a project-based organization. Structural solutions are cross-functional and co-located teams which offer a good basis for knowledge sharing between disciplines. Activity-based solutions are both formal and informal
upon-request activities aimed to share knowledge about a specific competence area, disciplinary or product specific. What remains unanswered is how to develop and sustaining deep product knowledge in the best way. Although, the implication is to be near the product and do the actual work.

2.2 Related work

This subsection describes theory about large-scale organizations, how building knowledge is dependent on the organization as a whole, and how to deal with complexity.

2.2.1 Agile in large-scale

Research has shown that many large companies are using agile methodologies that are inconsistent with the original ideas. This inconsistency can lead to problems in collaboration, knowledge management and the application domain. (Bamampubi et al., 2013) The transformation into agile in large-scale firms is one way of getting decentralized control with cross-functional competences in every team, which can take the full responsibility of developing and delivering new features (Bjarnasson et al., 2011). Bjarnasson et al. (2011) reports that agile in large-scale has caused new challenges when it comes to finding a good balance between agility and stability and ensuring necessary competence in the cross-functional teams. The balance between agility and stability is further explained. The agility is the core of an agile development process in order to be able to respond to changing customer needs, although these kinds of developing firms want a high degree of commitment by their cross-functional team members (Bjarnasson et al., 2011). This is a contradiction to Clark and Wheelwright (1992) which states that a team composition of specialized competences with high commitment during the whole process is essential when developing complex products.

The original idea of agile methodologies when developing software is to be flexible and fast, characterized by small and co-located teams with a high degree of customer collaboration (Abrahamsson, 2002). The co-location as well as the customer collaboration causes problems when it comes to using agile in large-scale organizations. The teams are often located at different departments in several countries. The customer collaboration is hard to carry on because large-scale developing firms tend to be market-driven. Market-driven software development entails special challenges when it comes to requirements engineering because of a communication gap
between the end-users and the developers (Karlsson et al., 2007). The communication gap is a result of the indirect contact between the developers and the end-users where information such as changed requirements travels through several proxies before they reach the developers (Karlsson et al., 2007; Cataldo et al., 2006). Bjarnasson et al. (2011) agree that large-scale organizations suffer from a communication gap. The internal and informal communication between the members of a cross-functional team cannot fully replace the more formal communication between different responsibility areas in a traditional line organization (Bjarnasson et al., 2011). This statement is supported by Hillebrand and Biemans (2004) which explain that organizational learning is highly dependent on good knowledge sharing within the organization, and emphasize that if the internal integration works sufficiently the competence level of the whole organization can increase.

Heikkila et al. (2013) reported a difficulty with creating generalist teams, which can implement software features in all components of a product. This is due to the technical complexity in large-scale systems where many components are interdependent. Such components require many years of experience to be fully understood. These organizations can therefore suffer from a long takeoff before new developers have gained a sufficient amount of experience to implement anything useful. Another impediment for these organizations is to identify who has the required knowledge to implement the features. (Heikkila et al., 2013)

2.2.2 Building knowledge in cross-functional teams

Hobday (2000) argues that a project-based organization can have a negative impact on human resource development in terms that it affect the long-term effectiveness and learning because of the lack of incentives for training. This is further explained by Clark and Wheelwright (1992) which state that long-term people centered issues shouldn’t reside with the project manager because projects are by definition a temporary organization. Research on organizational structure has emphasized that the responsibility of competence development is every employee’s responsibility when working in a project-based organization. At the same time it is the line manager’s responsibility to give incentives for such competence development and to have a focus on human resource issues. The project settings can also be of different characteristics, fragmented or focused. If the work settings are classified as fragmented, the project workers are assigned to projects on a part-time basis and commonly to several projects at a time. If the project participation is focused, the team members are co-located with their team on a full-time basis. (Bredin and Söderlund, 2011) An organization can also have intra-functional project work, which means that the project members mostly collaborate with
people of the same competence. The opposite is inter-functional project
work where the teams consist of people from different knowledge areas,
such as design, testing and systemization. Intra-functional project work
tends to develop specialist competences while the inter-functional project
work tends to drive the project workers to go broader in their competence.
(Bredin and Söderlund, 2011)

Xu and Ramesh (2009) argue that complex problem solving in software
development requires two types of knowledge to be effective, generalized
knowledge and contextual knowledge. According to the different types of
necessary knowledge needed to produce software in a complex context;
contextual knowledge needs a higher level of knowledge support to be
developed than the more general technical knowledge. Either way, it is
suggested to focus more resources on developing the contextual
competence, since it is motivated by the fact that contextual knowledge is
more effective. The reason behind that takes a standpoint in cognitive
science that emphasize that contextual knowledge helps to reduce the
complexity of the given task. (Xu and Ramesh, 2009)

Ghobadi (2015) lists several challenges in knowledge sharing for
cross-functional teams, including overcoming coordination challenges
across distributed sites, managing the diverse social identities, motivating
stakeholders to share embedded knowledge with the developing teams and
creating homogeneous teams with a shared understanding. Knowledge
sharing drivers are defined as: factors that drive the exchange of
task-related information, ideas, know-hows, and feedback regarding products
and processes (Ghobadi, 2015). To better understand the different drivers
behind knowledge sharing, Ghobadi (2015) performed a literature review
and identified drivers categorized into four different categories:
people-related, structure-related, task-related and technology-related.

People-related drivers can be one of three subcategories. The first is
diversity-related, for example the geographical location of the team
members. They can also be capability-related, for example the length of
being part of a team and current knowledge state of the team members. A
third subcategory is team perception drivers which relates to teams
interdependency, trust, sense of identity, project commitment and clarity
of the reward system. (Ghobadi, 2015)

Structure-related drivers can be of one of two subcategories: team
organization or organizational practice. Team organization drivers relate
to remote leadership, temporal restructuring of team members, assignment
of representative roles, and clients’ embedment. Organizational practice
includes organizational norms and networks that may drive knowledge
sharing. (Ghobadi, 2015)

Task-related drivers are project risks, project knowledge, task complexity,
and shared tasks between developers and users. (Ghobadi, 2015)
Technology-related drivers relates to project methodology, standardization of technology used and collaborative technologies. Templates and tools that can help the developers are included as well. (Ghobadi, 2015)

Ghobadi (2015) emphasize that the different drivers can be somewhat unique for different organizations and that the classification framework would benefit from being updated from time to time. However, the framework can be used as an inventory of potential impediments for knowledge sharing in order to focus more resources to solve those issues.

2.2.3 Complexity

Hobday (1998) describes different dimensions of product complexity. Among them are number of components, number of design choices, degree of customization of both system and components, elaborateness of system architecture, and depth of knowledge and skill inputs required. Some products can also be categorized as extremely complex because of the occurrence of the dimensions where a subsystem of the product can be categorized as complex by its own (Hobday, 1998). The nature of these products, referred to as CoPS (Complex industrial Products and Systems), can lead to extreme task complexity and demand the whole organization to allow for particular forms of management. (Hobday, 2000)

Well established research of complexity in software engineering are McCabe (1976) and Halstead (1977). They have produced methods for deriving the complexity of software described in Kearney et al. (1986). What’s common in both McCabe (1976) and Halstead (1977) is that they neglect the impact of factors beyond the actual code. Programmers need to perform a lot of different tasks including design, coding, debugging, testing, modification and documentation, which cannot be assumed to be equally difficult to perform. What are of high relevance are the surrounding organizational factors and the variety of tasks performed by the developers. Kearney et al. (1986) also argue that different types of experience influence the difficulty of a complex task. This type of experience is general programming knowledge, but also deep understanding of the specific domain.

As Kearney et al. (1986) states and what Hobday (2000) implicates is that beyond the organizational factors, you shouldn’t only look on the actual amount of code without considering the variety of tasks that is involved in the development process. One example is that Henderson and Clark (1990) found that even very small changes in the architecture of a system can have major impact because of the task dependencies of the ongoing projects, and lead to substantial consequences in coordinating work for the organization. Earlier research has tried to find ways to effectively manage these types of dependencies in software engineering. Nidumolu and Subramani (2002) highlights that to effectively manage system development projects requires
two separate modes of control, standardization and decentralization. The standardization refers to standard components and modeling tools which is supposed to minimize the complexity and make the daily work easier for the developers. Decentralization can for example be handled by having project teams that can drive the development by their own.

It has also been shown in research that new product development with short time-to-market and high competition between companies drive the competitors to be fast and flexible (Takeuchi and Nonaka, 1986; Hobday, 2000). Hobday (2000) explains that when developing complex industrial products and systems you need technical competent teams that can handle the difficulties where even small changes can have major impacts on the system as a whole. He highlights that this environment requires the organizations to have particular forms of management that can handle this complexity, and emphasizes that a project-based organization is the most suitable organizational structure in order to handle complex non-routine tasks. Hobday (1998) highlights the importance of system integration and project management as core activities to be able to handle development of complex systems. The teams in this type of organization should also be formed by a high integration of specialized competences with involvement during the whole project in order to integrate the different developed parts in a good way (Clark and Wheelwright, 1992).
Chapter 3

Method - Case study design

This chapter gives a comprehensive description of the method used in the study, and also the rationale behind it. The chosen method is an improving case study with a critical approach. The data collection is conducted through a qualitative data collection consisting of semi-structured interviews. The collected data was further validated by a focus group. The structure of the research is according to the guidelines by Yin (2012) with addition of Runesson and Höst (2008), which argues that case study method is a suitable method in software engineering research, as this research is.

3.1 Case study research

A case study is a suitable method in software engineering because it studies a contemporary phenomenon in its natural context (Runesson and Höst, 2008). More specific, the case study method used is action research which has the purpose to "influence or change some aspect of whatever is the focus of the research" according to Robson (2002). This is specifically appropriate in software process improvements, where both Dittrich et al. (2008) and Iversen et al. (2004) suggest that the research method should be characterized as action research as described in Runesson and Höst (2008). Runesson and Höst (2008) explain four different research methodologies:

- Exploratory
  - Finding out what is happening in order to generate new insights and hypotheses for new research.

- Descriptive
3.2. RATIONALE BEHIND CHOSEN METHOD

– Describing a phenomenon or situation
  • Explanatory
    – Aims to find an explanation of a situation or problem
  • Improving
    – Trying to improve a certain aspect of the studied phenomenon

In this case, the improving methodology was chosen because of the final goal of the research; to find possible solutions of the addressed problem areas.

3.2 Rationale behind chosen method

The methodology used in research must be suitable to answer the research questions derived from the problem definition, i.e. the reason why the research is conducted. The problem definition presented in chapter 1 has the characteristics of seeking for impediments in order to improve a situation. Runesson and Höst (2008) define this type of study as a critical case that aims to identify social, cultural and political domination that hinders human ability. This critical approach is interesting in this case because of the underlying organizational structure with processes that may include impediments for the people involved, in this case of difficulties in developing deep product knowledge. Improving case studies can have the characteristic of being critical (Runesson and Höst, 2008). This improving and critical approach is also chosen to bring more value to the company under investigation, for which the study is conducted.

3.3 Research questions

The problem definition can be categorized into three different aspects that may influence the human ability; these are product knowledge, distribution of tasks and resource allocation. These three categories are broken down into research questions which work as a basis in the case study. The research questions are further broken down into interview questions which can be found in Appendix A and B. The research questions are presented in the next three subsections.

Product knowledge
  • What’s the current state of the product knowledge in the cross-functional teams?
CHAPTER 3. METHOD - CASE STUDY DESIGN

- How do people think that the product knowledge affects their performance?
  - Which are the impediments for development of deep product knowledge?
  - How can the impediments be avoided or better managed to allow for better product knowledge?

**Distribution of features**
- How is the distribution of features to develop by the cross-functional teams handled today?
- How does the distribution of features affect the product knowledge and feature-flow?
- How can the distribution of features be better managed to maintain and grow product knowledge?

**Resource allocation**
- How’s the resource allocation handled today, both with respect to the cross-functional teams and with respect to the supporting roles around the teams?
- How does the resource allocation affect the product knowledge and feature-flow?
- How can the resource allocation be better handled to allow for increased product knowledge and long-term feature-flow?

### 3.4 Data collection

The data collection in a case study can be quantitative or qualitative. The quantitative data is normally collected from surveys and can be analyzed with statistical tools. Qualitative data is collected from observations or interviews and the result is rather unstructured which means that it can’t be a basis for statistical methods in order to prove validity. (Runesson and Höst, 2008) In order to improve the validity in a qualitative study, triangulation can be used (Yin, 2012). This means that you compare the collected data, in this case the answers from the interviews, with multiple sources (Yin, 2012). The different sources for this purpose are the theoretical framework, secondary data and metrics from the company and already conducted studies from other parts of the company. The triangulation is supposed to verify
(or challenge) the findings from the primary data source, in this case the interviews, and that way improve the validity of the outcome.

### 3.4.1 Interviews

Jacobsen (1993) described the interview as a conversation between two parties with the goal of mediating the knowledge, opinions, experiences and perceptions from the interviewee to the interviewer. An interview can be structured, unstructured or semi-structured. In a structured interview series, every interviewee gets the exact same questions which are decided in advance. An unstructured interview is the opposite and allows the interviewee to speak freely without any previously produced questions. A semi-structured interview is somewhere in between. The interviewer has a protocol with questions to be answered, but it allows a more free discussion of topics that pop up during the interview. The questions can be asked in any order depending on the interviewee’s answers. This type of interview can be considered flexible in that sense. (Denscombe, 2009) This flexible approach seemed most suitable in this case study because of the characteristics of seeking for action points to the impediments found, i.e. trying to collect possible solutions for the obstacles the interviewees describe. Runesson and Höst (2008) describes data collection techniques, originally divided into three levels by Lethbridge et al. (2005), where interviews are categorized as first degree. They describe this degree as direct methods where the researcher is in direct contact with the subject and that way collects the data in real time.

### 3.5 Data analysis procedure

The research collected data should be handled differently depending on the type of data collected. If the data is quantitative the most applicable method is statistical analysis such as correlation, and hypothesis testing. The data analysis objective in qualitative research is to derive conclusions from the data from a chain of evidence, which means that the researcher must present enough information from each step in the process and describe every decision taken. It is also recommended to carry on with the analysis in parallel with the data collection in a systematic way. Another recommendation is to let multiple researchers analyze the data and merge it into one analysis, in order to reduce bias from individual researchers. (Runesson and Höst, 2008) Braun and Clarke (2006) suggest a method for data analysis called thematic analysis approach, which is widely accepted in scientific and social science research. The method consists of six steps, which can be used as a basis in the case study protocol:
1. Familiarization with the data
2. Generating initial codes
3. Searching for themes
4. Reviewing themes
5. Defining and naming themes
6. Producing the report

3.5.1 Interview recording and summary writing
To allow for the use of the recommendations from Braun and Clarke (2006), it is suggested to record the conversation, even if notes are taken. This is because of the problem with writing down all details and the difficulty to knowing which information is the most important. After the interview, the data should be transcribed into text, preferably by the researcher because new insights can be made during the process. It can also be beneficial to let the interviewee review the text so he or she can validate the interpretations made by the researcher. (Runesson and Höst, 2008)

3.5.2 Coding the summaries
Runesson and Höst (2008) suggest several useful analysis techniques for qualitative research. One technique is to code the data with representations of different themes, areas or concepts. One code can be assigned to several pieces of text and a piece of text can have many codes. The codes can further build hierarchies and be combined with memos. This procedure will help the researcher to analyze the data and identify similarities and contradictions. One useful technique for analysis of the coded data is to use tabulation where the coded data is arranged in tables to give the researcher a better overview. There is also useful software available that support qualitative data analyses, which provide visualization tools and models that represent the coded data. (Runesson and Höst, 2008) This step in the data analysis contributes to step 2-5 in the thematic analysis approach by Braun and Clarke (2006).

3.5.3 Weighting the outcome
One threat to the validity because of the conversational nature of an interview is reflexivity, which means that the conversation leads to a mutual and subtle influence between the interviewer and interviewee Yin (2012). One way to handle this validity threat is to let a third party review
3.6. VALIDITY PROCEDURE

The validity procedure in a case study is aimed to verify the credibility and the trustworthiness of the result. Even though the validity is not finally evaluated until the analysis phase in a case study, it should be considered during the whole process. (Runesson and Höst, 2008) Yin (2012) describes four commonly used tests to establish the quality of any empirical social research: construct validity, internal validity, external validity and reliability.

3.6.1 Construct validity

The construct validity test is aimed to identify correct operational measures for the concepts being studied. The construct validity relates to two different phases of a case study: data collection and composition. During data collection the researcher must identify correct operational measures for the studied concept. This means that the interview questions are interpreted the same way by the researcher and the interviewees and that they relate to the research questions in a good way. Construct validity also relates to the composition phase where the researcher risks to report false interpretations or conclusions. Different tactics can be used to increase the construct validity. The first is to use multiple sources of evidence, so called triangulation. Another tactic is to use a chain of evidence when deriving conclusions. The last tactic is to let key informants review the draft case study report until agreement is settled. (Yin, 2012)

3.6.2 Internal validity

Internal validity reflects whether the results reflect reality or if the causal relationships are spurious. In a case study where the researcher tries to find such relations during the analysis phase, one threat is that an event rather is explained by a factor outside the study. (Yin, 2012)
3.6.3 External validity

Yin (2012) describes external validity as defining a domain to which the results from the research are generalizable. The researcher should analyze to which extent the findings are relevant to other people outside the investigated case (Runesson and Höst, 2008). The form of the initial research questions can directly influence this aspect. In other words it is a matter of research design to increase the external validity. (Yin, 2012)

3.6.4 Reliability

Reliability concerns whether the data collection and analysis procedure can be repeated by someone else, with the same outcome. Two possibilities to prevent this validity risk is to use a case study protocol and develop a case study database so that all operational steps in the case study can be repeated. Yin (2012)

3.7 Conducting research ethically

Case studies in software engineering often include dealing with confidential information about the contemporary phenomenon (Runesson and Höst, 2008). Yin (2012) highlights the importance of giving specific ethical concern to all case studies involving human subjects. Included within this topic is to have a responsibility to scholarship, according plagiarism and falsifying information, as well as avoiding deception and protect the interviewees. The information about the interviewees should be handled with respect to both privacy and confidentiality. A plan for how such information is handled should be performed as well as providing information about the ethical considerations to the subjects involved in the study. (Yin, 2012)
Chapter 4

Case study

This chapter presents the conducted case study. This includes background of the company, data collection procedure and focus group meetings.

4.1 The organization

The department where the case study is conducted started to use Agile and Lean methodologies with cross-functional teams about three years prior to this study. The product that is being developed is a highly complex product consisting of several subsystems. The department is divided into two units where one of the units is developing a subsystem of the product. This subsystem and the employees working with it is the main focus of the study. The subsystem in which the interviewees are working is one out of seven logical layers in the product. The logical layers are a way to simplify the architecture of the product, where the developers don’t have to worry about the details in the other subsystems. Instead there are interfaces between the different layers that the developers need to know about. The subsystem the interviewees work in has such interfaces to five other subsystems in the product. There are external dependencies between the different subsystems, which affect the programmers both in a technical dependency manner but also in an organizational manner.

The department is feature-driven which means that every cross-functional team is responsible for developing features. The features are further packaged into releases to the market twice a year. Every feature can involve several modules and software units in the subsystem and they are also dependent on other subsystems outside their own knowledge area.

To be able to draw general conclusions from this case, concern is given to
both the organizational factors and the technical factors that may affect the outcome of developing and sustaining deep product knowledge.

4.2 Initial focus group meeting

An initial focus group meeting was held in order to settle an agreement of the research questions and the appropriate scope of research. The focus group consisted of eight representatives from different roles at the department in order to give a comprehensive overview of the situation from various perspectives. Ten interview subjects were provided from the company.

4.3 Pre-study

The pre-study phase consisted of gaining a theoretical background within the field of research and to conduct observations. The objective with the observation was to give the researcher an appropriate amount of knowledge about the way of working at the department. Except from the focus group meeting and reading a lot of internal documentation, the researcher was also invited to a project retrospective performed by the project office in order to highlight impediments that occurred during a release. Every team involved in the release performed a retrospect and the outcome was evaluated and further investigated by the Program Office. The researcher was also invited to daily stand-up meetings dictated by Scrum, to give insights of the way of working in the cross-functional teams.

During the pre-study phase, a case study plan was written which worked as a planning tool for the whole case study. The problem definition from chapter 1.3 was broken down into research questions and further into interview questions (see Appendix A and B). Two separate documents with interview questions were created, depending on the interviewee’s role in the company.

4.4 Data collection and analysis

Nine out of ten of the interview subjects agreed on a meeting. Four of the subjects were members of cross-functional teams, and five subjects had supporting roles outside the teams but with direct influence on the teams. A summary of the subjects can be seen in Table 4.1. Prior to each meeting, an introduction document was sent to the interviewee in order to establish an understanding of why the case study was conducted (see
4.4. DATA COLLECTION AND ANALYSIS

appendix C). The meetings were held during a period of five weeks, to enable time for documentation of the data collected from the interviews in parallel. Every interview was held in between one hour and a half and three hours. Notes were taken and the conversations were recorded. The interview session started with an introduction part to make sure that the researcher and the interviewee had the same interpretation of why the study was conducted (see Appendix D). Towards the end of each interview session, the interviewer gave a summary of his interpretations of the answers to avoid misunderstandings. The interviewee also got the opportunity to reformulate and regret his or her answers.

<table>
<thead>
<tr>
<th>XFT / Supporting role</th>
<th>Role description</th>
<th>Time at the company (years)</th>
<th>Time in current position (years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XFT</td>
<td>Scrum master and testing</td>
<td>7</td>
<td>1,5</td>
</tr>
<tr>
<td>XFT</td>
<td>Scrum master</td>
<td>16</td>
<td>10 as team lead, a few years in current position</td>
</tr>
<tr>
<td>XFT</td>
<td>Developer</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>XFT</td>
<td>Scrum master and developer</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Supporting role</td>
<td>Line manager</td>
<td>24</td>
<td>16</td>
</tr>
<tr>
<td>Supporting role</td>
<td>Program manager</td>
<td>7</td>
<td>0,5</td>
</tr>
<tr>
<td>Supporting role</td>
<td>Specialist (Senior developer)</td>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>Supporting role</td>
<td>Operational product owner</td>
<td>7</td>
<td>1,5</td>
</tr>
<tr>
<td>Supporting role</td>
<td>Product guardian</td>
<td>6,5</td>
<td>2,5</td>
</tr>
</tbody>
</table>

Table 4.1: Interview subjects

The following six steps were performed in the data analysis according to the recommendations from Braun and Clarke (2006):

1. Familiarization with the data

After each interview, the researcher wrote a summary of the recorded interview. Each of these summaries were then sent back to the interview subject for review, which resulted in a few minor changes.

2. Generating initial codes

The reviewed documents were analyzed in an analysis tool for qualitative research, MAXQDA\(^1\). In a first step, the text was coded

\(^1\)http://www.maxqda.com
CHAPTER 4. CASE STUDY

into relevant categories from the perspective of the research questions.

3. Searching for themes

The codes were grouped into identified themes according to their similarities. Both identified impediments and possible solutions for the impediments were grouped into the same theme.

4. Reviewing themes

The themes were reviewed in order to identify mistakes and also to regroup codes that would work better in another theme. Some themes were identified to be too extensive why they were broken down into smaller themes. This built hierarchies in the codes, which made it possible to identify the origin of each code and which other codes they relate to.

5. Defining and naming themes

The theme definitions were settled which resulted in nine different themes. With respect to subthemes twenty different impediments were found, with different possible solutions described for each of them. The final hierarchy can be described as four levels: categories (product knowledge, distribution of tasks, resource allocation, and technology), themes, impediments, solutions.

6. Producing the report

The result of this step is this paper, where the result is presented and discussed. In addition there are recommendations for handling the impediments.

4.5 Second focus group meeting

The impediments from the data collection were listed and described in a focus group meeting document (see 5.2.1. Impediments). The identified impediments were organized into a prioritized list according to the frequency of occurrence in the interview summaries. This document worked as a basis for the second focus group meeting where a risk analysis was performed in order to give the researcher input on how to prioritize the impediments and to verify the researcher’s interpretations. The risk analysis was performed according to a company standard (see Table 4.2) in order to minimize the risk of the methodological threat, and to enable focus on the actual analysis rather than the method used. One out of the twenty identified impediments was excluded because of the researcher’s misinterpretation of the phenomenon. It was also concluded that some of the impediments didn’t directly affect the product knowledge, but was
rather a result of a misinterpretation by the interviewees of how things were actually handled at higher levels of the organization. Because of that, some impediments were not further investigated because it was just a matter of information that needed to be addressed.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Risk (R)</th>
<th>Consequence (K)</th>
<th>Cost (C)</th>
<th>Priority (P) = (R + K) / C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Impediment X</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Table 4.2: Risk analysis

4.6 Third focus group meeting

The third focus group meeting had a focus on evaluate the researcher’s interpretations of applicable solutions to handle the impediments, which were extracted from the interview data. Because of inter-dependencies between the solutions, they were grouped in order to allow for comprehensive solutions of the way of working. The highest prioritized impediments from the earlier focus group meeting where grouped into 8 categories, all related to product knowledge. Possible solutions for the impediments were described, both based on the collected data and on the researcher’s own interpretations.

The different categories were evaluated and discussed during the meeting. Proposed solutions from the interviews where discussed in order to give a broader picture of what they should mean for the organization and which other parts that may be affected. A few solutions were excluded because they were a contradiction to the way of working demanded from corporate level, and by that outside the scope of this case study. Another reason why some solutions in different categories could be excluded was that they were not precise enough, and the actual origin of the problem it was aimed to resolve could be challenged.

4.7 Classification into frameworks

In order to give a holistic picture of the situation and to provide a basis for focusing resources to resolve the found problems, all impediments were grouped into four categories according to the drivers for knowledge sharing by Ghobadi (2015). These categories are people-related, structure-related, task-related and technology-related. This can be seen as a step towards
CHAPTER 4. CASE STUDY

providing useful insights for the company in order to focus resources to resolve issues. Hence it is an appropriate step in an improving research approach and also because of the rationale behind the chosen method.
Chapter 5

Results and analysis

This chapter presents the result from the case study together with an analysis of the result in reflection of the studied literature.

5.1 The organization

Important aspects to consider at the department are the number of cross-functional teams, developing several different products where every product has different subsystems under development. The part of the company which is included in this study is working with one subsystem in one of their products. This product development can be argued to be classified as large-scale by its own in terms of number of people involved and number of developing teams, as well as number of parallel projects, according to Moe and Dingsøyr (2014). When a development process includes so many different teams and roles, the process flow must be adapted to fit for agile development in large-scale (Moe and Dingsøyr, 2014). Nidumolu and Subramani (2002) suggest two types of control to effectively manage these kinds of developing organizations, standardization and decentralization. The company in this research has adapted to both. Standardization is their use of modeling tools in the development and decentralization is their use of cross-functional teams taking the full responsibility of developing features.

5.2 Result of first degree data collection

This section analyses the result from the interviews, with respect to the result of the focus group meetings. The first subsection lists the found
impediments together with an explanation of each. The second subsection views the result from the risk analysis performed by the focus group. The third subsection describes the current state in the organization and highlights the impediments that hinder knowledge evolvement. The fourth subsection analyzes possible solutions for the addressed impediments.

5.2.1 Impediments

This subsection describes all impediments found in the interviews. The different impediments are further analyzed based on the focus group feedback and according to theory.

Lack of support from Product Guardians
Many people are experiencing difficulties to reach the Product Guardians to get the necessary support regarding the product, due to their high workload. The Product Guardians has the responsibility within a technical area of the product, and is aimed to help the cross-functional teams with an overall perspective and to implement their features in a decent way. The Product Guardians are forced to prioritize their time, why teams allocated to features with a lower priority feel ignored. Meanwhile, the Product Guardians don’t get sufficient time to evolve their own expertise and work proactively with architecture and design environment.

Lack of support from system organization
The teams often suffer from a lack of experience within systemization, which is why they need external support. The system engineers are not involved in the design phase which is described as a problem in knowledge sharing. The system organization hands over a document describing the feature from a black-box perspective, which the teams try to implement.

Lack of support from Test Manager
There are specific mile stones that must be passed in order to get an approwvement for continuing the development process. Some mile stones include the Test Manager, who often is overloaded. The teams experience these mile points as a bottleneck in their feature development, but would still like them because it provides good feedback.

Hard to find support from other teams
Both team members and people from surrounding roles highlight the idea of enable the team to support each other in a better way. The problem is that it’s not obvious which team that has the necessary expertise. The way it works today is mainly by informal networks or by being coordinated by
the managers.

**Lack of reviews from other teams**
The team members want to enable reviews from other teams, but it is not possible when they are allocated to their feature development 100%. One particular task that should be reviewed by other teams is the configuration documentation for customers in order to secure that the feature is configurable according to the document.

**Allocation to features in too many areas of the product**
A direct impediment for gaining deep product knowledge is to get features in widely separate areas of the product. It hinders the team members to get necessary knowledge depth in any area. The optimum situation would be to get feature where you have some previous experience and that way develop your competence to get both deeper and broader.

**The teams are not involved in the distribution of features**
The teams are not involved in the process which affects the motivation and engagement to the feature. The problem according to the team members is that the different options of possible features are not visible.

**Prioritization of feature backlog is held too tightly**
The team members feel that the priority in the feature backlog is held too tightly. This result in features allocated to teams without sufficient knowledge in the area. One idea is to sacrifice the priority a bit, especially those with lower priority, to enable knowledge growth and strategic distribution of tasks.

**All features are allocated to a whole cross-functional team**
All features are allocated to a whole team even when 2-3 persons would be enough, which creates unnecessary overhead costs. If a few team members would be allocated to a small task for a few weeks, it could provide new experience to the whole team by sharing their new insights when they come back.

**Lack of product care**
The feature-driven development contributes to hasty decisions and shortcuts. This creates a technical depth which is not grazed in the same rate as it grows. The team members would like to work with product care items and that way enable knowledge growth which they don’t have time to in the feature development. Refactoring is described as a good way to gain new insights and grow knowledge depth in the product.
Lack of system knowledge in the teams
The teams feel that they sometimes lack expertise to be able to take the full responsibility for the feature. One particular competence they lack is systemization and how the product works from a broader perspective. This creates an uncertainty about how the feature they develop affects other parts of the system.

Lack of high level testing knowledge in the teams
The teams feel that they sometimes lack expertise to be able to take the full responsibility for the feature. One particular competence they lack is testing at a higher level and how to establish a good test environment. This creates a need for external support which is described as a bottleneck.

No presence of technical leadership in the teams
Many team members feel that they lack technical leadership (formally or informal) in the teams, who can drive the team forward and act as a role model within technical expertise. This person could also be a natural way of getting support and to spread knowledge within the team.

Cumbersome tools for documentation
The software developers must be able to manage about 50 different tools. The most advanced tools which are not use that often is a source for frustration. Both the tools for customer configuration documentation and documentation in the UML modeling tool has been described as impediments. The internal company Wiki is perceived as messy and out of date. These tools would work as a good source for knowledge sharing if they were better.

Overall system perspective
There is a problem with the teams being responsible for the whole feature while they lack necessary overview of the whole picture. Three areas are described: who the feature is for, why it’s being developed, and how it affects other parts of the product.

Time pressure
Relates to the feature-driven development process where teams are allocated 100%. No clear incentives are given for extensive knowledge sharing, product care, competence building or taking responsibility in certain areas of the product.
5.2. RESULT OF FIRST DEGREE DATA COLLECTION

UML modeling tool hiding information
The modeling tool is an impediment in learning the product, because of the lack of information provided to the user. Even when you do know what to implement, you must be able to apply it in the modeling tool. In addition, the tool abstract away essential information which makes it much more difficult to understand and monitor system flows.

Unstable testing environment
The entire testing machinery is perceived as problematic. It is not possible with any level of significance to say that the problems that fall out and causes errors depend on the implementation of the product or problems in the test environment itself. Hence, it hinders the understanding of the product.

Difficult architecture
The architecture is perceived as unclear and difficult to learn throughout the subsystem, where one particular part is highlighted as by far the hardest part to learn. Desirable would be a simpler architecture with clearer components where each SWU (software unit) has a clearly defined responsibility, which is correctly reflected by its documentation.

5.2.2 Prioritized Impediments

The different impediments described by the interviewees and interpreted by the researcher were prioritized in a list from a risk analysis performed by the focus group. The result of the risk analysis can be found in Table 5.1. The risk analysis works as input for the researcher to get a holistic picture of how different roles in the organization estimate the extent to which different impediments affect the product knowledge in the organization.

What is important to understand about the result from the risk analysis is that the priority is not necessarily the same priority as they should be handled. As an example the highest prioritized risk: prioritization of feature backlog is held too tightly, was according to the interviews an obvious impediments for the team members in their process of gaining a deep product knowledge in a specific domain of the system. Although, according to the focus group this was just a misinterpretation of how the feature backlog is managed. In other words it is just a matter of information about the feature list that hasn’t reached the team members in a successful way. Because of this type of result, the method used in the risk analysis can be criticized, where the cost factor is too significant for
the outcome. The risk analysis should rather work as support to understand the estimated effort necessary to better manage the impediments. What could be of even more importance is to neglect the cost-factor from the risk analysis and instead focus on which impediments that affect the knowledge sharing the most. There are also inter-dependencies between the found impediments, which are grouped and analyzed together in the following subsection.

5.2.3 Current state

All of the interviewees agreed that the domain specific knowledge is more important than general technical competence in order to produce features in a decent way. Furthermore they agreed that the domain specific competence is harder to develop and that it takes a long time, several years, in the company to build sufficient competence. This is a direct confirmation of Heikkila et al. (2013) which highlights the problems with creating generalist teams in large-scale software engineering because of the technical complexity. One of the interviewees with a supporting role in the organization describes the situation:

"Product knowledge is very important. It is the most crucial building block to be able to develop a feature, that you understand the product well. We have experienced that when we expand and grow and teach new people within our domain, there are several examples of problems that occur because of a lack of knowledge about the product. You affect one and others work negatively if you don't really understand the consequences of all different parts of the product, because of the interdependencies in the product. [...] When I speak to the Product Guardians about the current state in the organization they often highlight the problems with new teams starting to develop features, that they don't realize the consequences of what they are doing. It is an area where we need to improve and increase the competence level in how to develop features in a decent way from the perspective of the product and its different parts."

The interviewees had a consensus about how to gain deep knowledge about the product. All of the interviewees said that the best way to get better knowledge about the product is to work with development at the operational level and struggle with the problems to gain new insights. This goes well in line with Enberg and Bredin (2013), which implies that developing product knowledge require the employees to be near the products and do the actual work. Further, the best way to do this is to get tasks in a limited subsystem of the product for a sufficient amount of time. The implication of this result
### 5.2. Result of First Degree Data Collection

#### Table 5.1: Prioritized Impediment

<table>
<thead>
<tr>
<th>Risk</th>
<th>Consequence</th>
<th>Cost</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Area</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prioritization of feature backlog is held too tightly</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>All features are allocated to a whole cross-functional team</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teams are not involved in the distribution of features</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>No presence of technical leadership in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hard to find support from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time pressure</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features in too many areas of the product</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cumbersome tools for documentation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML modeling tool hiding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of reviews from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of product care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML modeling tool hiding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of reviews from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of product care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML modeling tool hiding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of reviews from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of product care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML modeling tool hiding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of reviews from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of product care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML modeling tool hiding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of reviews from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of product care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML modeling tool hiding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of reviews from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of product care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Difficult architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UML modeling tool hiding information</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Allocation to features from system organization</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of reviews from other teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of support from Product Guardians</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of product care</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of high level testing knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lack of system knowledge in the teams</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Overall system perspective</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
is that if you don’t get tasks in a subsystem where you have any recent knowledge you will need more support from the different specialist roles, according to Xu and Ramesh (2009). Otherwise there is always a risk that you just do a point effort to be able to deliver your feature before deadline, which creates a technical depth mentioned as a problem in several interviews. This problem is related to the complexity in the product, which a developer describes from the lowest level of development, the actual executed code.

"We use this modelling tool to generate code. The model is graphical with different states and transmission between the states. We can build different solutions in the transmission with C++ code. However, the generated code is impossible to understand. It would be impossible to go backwards from the generated code to the model. So you need to understand the product at a higher level, and then you just put your code where it should be."

Different tactics can be used in order to increase the competence level, where the different tactics works dissimilar depending on the type of knowledge. A software developer with a few years of experience within the product shares his thoughts:

"Within the product it is possible to learn from each other, by sitting together with someone more experienced, both by looking over the shoulder and to do the actual development and testing. Different courses can help to some extent, but mostly as an introduction. It is the practical work you learn the most from. To get a suitable task as soon as possible is the most effective way to learn the product. When it comes to tools, frameworks and different languages it is much easier to learn. There are online courses and also external courses you can attend. They help a lot in that area. But when it comes to the product you must struggle with the problems and learn from the more experienced people’s knowledge."

This quote summarizes the problem with evolving deep product knowledge and increasing the competence level in the organization. The product knowledge is domain specific and can only be developed in the own organization. The best way to gain new insights, is to work with development in the product and struggle with the problems. One way to increase the learning curve is to learn from more experienced people, for example senior developers and specialist roles. That way you can get the necessary knowledge support as stated by Xu and Ramesh (2009). Since large-scale software development is associated with an inability for everybody to know everything, knowledge networks are essential to find the right external support (Moe et al., 2014). A developer, who also has the role as Scrum Master in a team, describes her thoughts about
knowledge support:

"The most effective support is when you know exactly who to ask. Unfortunately you don’t always know who to talk to, especially when you are new. Then it is much easier to turn to the more formal supporting roles, such as Product Guardians, but they are often overloaded with work. It is also a matter of prioritization. If you work with a low prioritized feature it is harder to get the necessary support."

Product Guardians are a similar role to what Moe et al. (2014) described as the TAR, an expert with a supporting role that plays an important role to share experience. Anyway, what’s asked for by the interview is a way to increase the knowledge sharing, not necessarily by formal roles. The thoughts about knowledge sharing goes well in line with Hillebrand and Biemans (2004), which state that ineffective knowledge sharing can affect the competence level of the whole organization, where they also highlight the importance of knowledge sharing between different departments.

The product knowledge is essential for feature throughput in the organization, and the conventional truth is that the competence level is too low. The competence is highly dependent on the tasks and where in the system you work, which finds support from Zika-Viktorsson et al. (2006) with the addition that multi-project environments decrease the time for reflection, learning and recuperation, and by that the possibilities to increase the organizational competence level. The next subsection will focus on impediments for knowledge sharing and the reasons behind the current state of knowledge in the organization.

5.2.4 Knowledge-driver impediments

From the perspective that the best way of gaining deep product knowledge is to work with the product, the most commonly mentioned impediments refers to difficulties in enabling enough time for reflection and refactoring. Also, contrary to general technical competences, the product knowledge is not possible to build by attending courses and workshops, so the deep domain specific knowledge is a matter of experience in the product. The different impediments from the interviews can be categorized into the drivers for knowledge sharing by Ghobadi (2015). In this case, the impediments are obstacles that hinder the knowledge sharing in some way. A summary of the identified impediments from the perspective of the drivers for knowledge sharing can be found in Table 5.2.

People-related impediments include: teams are not involved in the distribution of features, time pressure, lack of system knowledge in the
<table>
<thead>
<tr>
<th>Knowledge-drivers</th>
<th>Impediments</th>
</tr>
</thead>
</table>
| People-related    | Teams are not involved in the distribution of features  
|                   | Time pressure  
|                   | Lack of system knowledge in the teams  
|                   | Lack of high level testing knowledge in the teams  |
| Structure-related | No presence of technical leadership in the teams  
|                   | Lack of support from Product Guardians  
|                   | Lack of support from Test Manager  
|                   | Lack of reviews from other teams  
|                   | Lack of support from other teams  
|                   | Lack of support from system organization  |
| Task-related      | Prioritization of feature backlog is held too tightly  
|                   | All features are allocated to a whole cross-functional team  
|                   | Allocation to features in too many areas of the product  
|                   | Lack of product care  |
| Technology-related| Cumbersome tools for documentation  
|                   | Unstable testing environment  
|                   | Overall system perspective  
|                   | Difficult architecture  
|                   | UML modeling tool hiding information  |

Table 5.2: Knowledge driver impediments
teams, and lack of high level testing knowledge in the teams. Time pressure directly influences the people’s opportunity to learn, because it reduces the time for reflection, learning and recuperation, according to Zika-Viktorsson et al. (2006). This is people-related because it is the employees that experience it, although the origin of the problem may not be people-related. Two of the other mentioned people-related impediments, lack of system knowledge in the teams and lack of high level testing knowledge in the teams, are impediments that are directly experienced by team members. The lack of knowledge tells that the competence level needs to be increased. Anyway, as experiences from the teams, the lack of knowledge is often related to the remaining people-related impediment: teams are not involved in the distribution of features. So while the experienced impediment is time-pressure, because the current competence level require a high level of external support in many cases, the origin of the situation is rather dependent on structure and task-related impediments, which goes well in line with Zika-Viktorsson et al. (2006), which tells that multi-project environments increase the risk of high workload.

The two subcategories of structure-related drivers, team organization and organizational practice, are both represented by some impediments, and connected to the people-related impediments. Team organization includes no presence of technical leadership in the teams, lack of support from Product Guardians, and lack of support from Test Manager. Organizational practice includes lack of reviews from other teams, lack of support from other teams, and lack of support from system organization. What’s common for all these impediments is that they relate to external knowledge support to the teams. The reason behind that is said to be the extreme task-complexity, which is a result of the complex domain including both organizational factors and the product under development, which require a high degree of knowledge support. The required support from specialist roles, such as the Product Guardian and the Test Manager is in turn dependent on the teams’ knowledge network. The supposed way to handle knowledge support is to first trying to solve the situation inside the team. Secondarily you turn to other teams to get the needed support. The problem with support from other teams are twofold. First of all you must know which team that has the right competence in different product areas. Second, there are no clear incentives to re-allocate a lot of your time to support other teams. This is where the specialist roles come in. It is much easier to turn to the Product Guardians, which are supposed to help the teams according to their role description. This situation is similar to what Moe et al. (2014) describes; that the necessary knowledge support is highly dependent on the team’s knowledge network. The fact that the specialist roles are overloaded further implies that the amount of necessary knowledge sharing is very high in the organization (Moe et al., 2014).
There are two different types of knowledge support needed, general technical knowledge such as testing and domain specific knowledge about the product, similar to Xu and Ramesh (2009). Although the lack of support from the Test Manager is mentioned as an impediment, the origin of test environment issues often relates to an overall picture of the product, where the test manager tries to provide the teams with that perspective. It is therefore motivated to take the same standpoint as Xu and Ramesh (2009) and highlight that the contextual knowledge needs a higher level of knowledge support to be developed. Their recommendations is also to focus more resources on developing contextual knowledge, since it is more effective and helps to reduce the complexity of a given tasks. So except from trying to increase the possibility of external support, this leads to the task-related impediments which are related to the consensus about that the product knowledge is evolving during the operational work with the product. The task related impediments include prioritization of feature backlog is held too tightly, all features are allocated to a whole cross-functional team, allocation to features in too many areas of the product, and lack of product care.

From the perspective that there is a feature backlog with tasks to be distributed to the teams, they must distribute the features to the teams in a way that secure delivery to the customers. This is not necessarily a contradiction to a distribution of features to the teams from a strategically competence growth perspective. Because of the complexity in the product and the size of the system under development, it would be impossible for everybody to know everything. The feature-flow and the product knowledge are further described as interrelated. One of the interviewees who work as a line manager describes the relationship between features and knowledge:

"There is definitely a direct link. If you can work with features that are connected to a specific product area, you will learn that area after a while. You build the competence around it so it relates to the principle I mentioned earlier, that you learn by working within the product. [...] I definitely think that it is a link there, quite clearly between what feature you develop and what product knowledge you build."

So there is a linkage between the features you develop and the knowledge you build, but how should the features be distributed to allow for both knowledge growth and feature-flow? A manager from the Program Office describes his point of view:

"For me, it is the absolutely most important factor to consider, that you have product knowledge within the feature you are supposed to develop. I think that you, in practice, should only work with features in a product area you are
familiar with. Of course you need knowledge on the other side of your interfaces to some extent and possibly make a few minor changes there as well, but the majority of the work should be done in the area where you have the best knowledge."

The impediment prioritization of feature backlog is held too tightly was up for discussion at a focus group meeting, and concluded to be a misinterpretation by the interviewees. The misinterpretation is a result of the people-related impediment teams are not involved in the distribution of features, which during this research already has been improved and is therefore not further investigated.

The remaining task-related impediments concern the distribution of features and additional tasks to the teams. The distribution of features to the cross-functional teams are handled by the Program Office, which in turn are provided a prioritized lists of features that should be included in the next release from a higher level of the organization. From the perspective of the developers, it is beneficial to be familiar with the product area where you get a feature because of two reasons. First, to deliver the feature in time but also to enable to gain deep product knowledge that requires tasks within a limited area of the product for a long time. Second, it is a matter of motivation. When you get a feature to develop you want to feel that you can contribute. A developer at the company describes the situation:

"I wouldn’t have any problem with working on features depending on different modules in the same subsystem I’m currently working in. What would be a problem though is to be put in another subsystem. I mean, I only know a subset of the modules in my own subsystem after one year and a half."

A manager from a supporting role agrees and describes what he thinks is the most important to consider when distributing tasks:

"The most important is that it is a task that the team can manage, that you feel that it is realistic within the given timeframe. But at the same time it shouldn’t be too much of a routine task. It should always include a challenging element, so that the competence can be increased automatically."

One further problem with features in product areas that the teams are not familiar with, except from feature-flow and knowledge growth, is lack of product care. Product care items are put on a backlog and the teams have the responsibility to resolve one item in parallel with the feature development. The product care items are highly related to the earlier mentioned technical depth, and are often a result of bad implementations. The lack of product care items to resolve are obviously not the origin of the problem, but the number of items is increasing. Anyway, as described
by the developers, the product care items are a way of refactoring and allow for further reflection and therefore appreciated from the perspective to learn the product, which goes in line with Zika-Viktorsson et al. (2006). A senior developer described the problem with product care items:

"We have built this technical depth from day one, and it’s really hard to get rid of. [...] Of course you can take shortcuts sometimes, but you have to make sure that you fix it later. I think the technical depth we build is bad from the perspective of feature-flow as well, in the long run. If you are supposed to work according to agile, it is even more important, I would say, to take care of the product."

Except from the organizational factors such as resource allocation and the fact that the development process is feature-driven, the complexity the developers sometimes suffer from is best described from the perspective of technology-related impediments. These are cumbersome tools for documentation, unstable testing environment, overall system perspective, difficult architecture, and the UML modeling tool hiding information.

One of the specialist roles mentioned several times in the interviews is the Product Guardian. The fact that this role is overloaded, which is similar to the situation for the TAR described by Moe et al. (2014), implies that the need for external support to the teams is high. These implications are also related to problems such as time pressure, lack of support and complex technology. One of the interviewees who work as a developer in a cross-functional team summarizes the problem:

"It is all a matter of time. If you get a feature in a new area of the product there will be a long take-off distance. The supporting roles who are supposed to help in these situations are overloaded with work, and when you get a chance to meet them you don’t have the right questions as a result of your lack of knowledge. [...] There is always a risk that you just paste the code needed for the feature and then leave it behind. The design tool we use is not really helping; it actually makes the whole product even more difficult to understand."

The modelling tool is a way of making the daily work easier for the developers, which enables a graphical visualization of the implementation. This standardization is a way of effectively manage system development, as stated by Nidumolu and Subramani (2002). Anyway it is not always appreciated by the developers and may harm their possibility to actually understand the system. An employee from the Program Office describes the tool:

"Regarding the tool, it is important to know when and how to use it, otherwise it will easily get messy. Sometimes it fits
very good to use these state machines but sometimes it would be better to use classes instead. When it is suitable to use state machines it is a perfect tool. But for more lightweight tasks it is inappropriately difficult."

Another developer with a specialist role has a similar view of the tool:

"However, it is quite difficult to address the knowledge growth, especially when we have this UML modeling tool which is a stupefying tool because it hides a lot of things that you really should know as a programmer. The fact is that it is abstracting away what you want to see and understand as a programmer, to make the right decisions. That is what I mean with stupefying. You don’t understand how the program is really built, and what actually executes and what makes the program run. I don’t say that we should abolish state machines, it is extremely central in what we do, but we have to improve."

So while the standardization is good from a perspective to deliver features to the market and effectively manage a large-scale organization, according to Nidumolu and Subramani (2002), it may harm the knowledge growth and possibly even the feature-flow in the long run.

Nidumolu and Subramani (2002) also mention another typical way to standardize the development process, to use standard components, which related to the impediment with difficult architecture. The system is built upon such components but these have been more difficult to define and to understand what they actually are responsible for. A senior developer describes the components:

"The architecture is very messy today, with components that actually not are components, because all of them need each other anyway. So it’s a spider web of components really stuck together where no one stands alone. That’s the reason why it’s so problematic when you enter your code somewhere, that it always affects someone else. We should create something that is clearer encapsulated so that it is more obvious where sequences and use cases should be implemented and that you can be sure that it won’t affect other’s code. [...] If you have a good design with clear and separated functions, it would also be much easier to learn. Today you don’t know which actual responsibilities different parts have. A software unit should exactly like a class have a well-defined responsibility, but the problem is that we have just added new functionality. So the lack of product care and that you don’t work proactively with the architecture is a vicious circle which makes the learning harder."
The technical impediments, the UML modeling tool and the difficult architecture, make it hard to understand the product at a high level, and could be subcategories to the impediment overall system perspective. Cumbersome tools for documentation make the situation even worse, and the focus is often on tools rather than development, documentation and programming. One further technical impediment is the testing environment, which is described as unstable. A manager from the Program Office describes the testing environment:

"I don't think that we have allocated enough resources on the surrounding problems, such as design and testing environment. My picture is that those are the most important factors, that they work correctly. The testing environment must be much better than the product, so that when you find an error, you can be sure that it is the product and implementation that is wrong and not the testing environment."

From the perspective of Sekitoleko et al. (2014) we can see that there are many dependencies between different challenges in the organization. The features are planned for a delivery which takes place twice a year. This is challenging for large-scale organizations even though the way of working with agile development principles aims to solve problems related to this area such as changing customer requirements (Bjarnasson et al., 2011)(Karlsson et al., 2007). The market-driven feature prioritization is further complicating the situation since the features, which the company gets paid for, are often prioritized over product care items. This task prioritization challenge, together with the knowledge sharing challenge such as a high need of external knowledge support to the teams affect the product in a bad way and creates a technical depth, related to the code quality challenge described by Sekitoleko et al. (2014). The knowledge sharing challenge is further affected by the integration challenge where both different teams developing features with technical dependencies affect each other, such as the absence of the system organization’s integration into the feature development. As we can see in this case, and what is supported by Hillebrand and Biemans (2004) is that the lack of integration and knowledge sharing affect the whole organization. This is further proved by Sekitoleko et al. (2014) which states that the knowledge sharing is vital and a good starting point to break the vicious circle. Anyway, it could be improved by starting to mitigate any of the challenges (Sekitoleko et al., 2014).

5.2.5 Solutions

The perspective from the research questions imply that the product knowledge is affected by both resource allocation and task distribution. It
5.2. RESULT OF FIRST DEGREE DATA COLLECTION

has been shown that these relate to the structure-related and task-related
drivers for knowledge sharing described by Ghobadi (2015). What has
been further shown is that technology-related problem areas cause
impediments for the knowledge growth as well, while people-related issues
are more abstract experienced problems where the origins of the problems
are often somewhere else. Sekitoleko et al. (2014) have shown that the
planning challenge, task prioritization challenge, knowledge sharing
challenge, code quality challenge and integration challenge are
interdependent and can create a vicious circle. The results in this research
point in the same direction. The vicious circle can be broken by starting to
mitigate one of the challenges where the knowledge sharing challenge is a
recommended starting point (Sekitoleko et al., 2014). The knowledge
sharing drivers can therefore be a good starting point in focusing resources
in an appropriate way (Ghobadi, 2015).

The people-related impediments teams are not involved in the distribution
of features, time pressure, lack of system knowledge in the teams, and lack of
high level testing knowledge in the teams are hard to focus resources on, since
they relate to people’s feeling about the situation or lack of knowledge. The
lack of involvement makes people less motivated in the feature development.
The lack of knowledge makes the knowledge sharing suffering and is mostly
a matter of insufficient time for learning, refactoring and reflection, which
is supported by Zika-Viktorsson et al. (2006). So in order to increase the
knowledge sharing and develop deep product knowledge we have to find the
origin of these impediments, and resolve what hinders the human ability.

The structure-related drivers for knowledge sharing relates to the
organizational settings and the possibilities to get knowledge support when
needed. It also relates to the research questions about resource allocation
and the surrounding supporting roles. The cross-functional team members
are co-located on a full-time basis, and the work settings would therefore
be categorized as focused by Bredin and Söderlund (2011). The
inter-functional project work is a result of the decentralization where every
team should be able to take the whole responsibility of a feature. The
decentralization is a good way to effectively manage dependencies in
software engineering, according to Nidumolu and Subramani (2002), why
the cross-functionality itself is not questioned. Anyway, the
cross-functional work is somewhat a contradiction to the desired knowledge
and competence growth, since inter-functional project work makes the
different competences within a team to learn from each other and rather
go broader in their knowledge Bredin and Söderlund (2011). This is
especially true when it comes to general technical competence. The
organizational structure puts great demands on the surrounding technical
supporting roles, where we have seen that there is a high workload. The
results from the interviews highlight suggestions that you should increase
the number of supporting roles. A senior developer explains:

46
"I do certainly not mind working agile, but if you say that you should work this way and still focus on the product, then the supporting roles such as the Product Guardians are completely undersized in comparison to what we require from them in order to work completely feature-driven. We have about as many supporting roles now that we would have if we still had product-oriented teams."

Since there is a lack of knowledge in the teams which require more knowledge support from expert roles, it is possible to look beyond the overloaded roles and see what is actually needed. The demanded support is knowledge from a high level perspective of the product, like the perspective from the system organization. An Operational Product Owner describes a way of increasing the integration between these organizations:

"So it’s an ongoing missions [the system organization] that it as important as design and testing, it’s the third leg somehow. It would therefore be good to have them in the teams as well. Firstly, it would be a good thing to increase the knowledge exchange, but above all, one should have a rotation between the design organization and the system organization to a higher degree. It becomes an indirect way to spread knowledge, which could gain both parties."

This type of overlap gets support from the literature where Hillebrand and Biemans (2004) emphasize that if the internal integrations works in a decent way, the competence level of the whole organization can be significantly increased. The cross-functional teams often demand the perspective from the system organization from the Product Guardians, why it is an appropriate assumption that the workload for that supporting role can be decreased as well.

What also could decrease the workload for the expert roles, and by the same time increase the knowledge sharing is to make it easier for the cross-functional teams to support each other. Today the support is mainly requested from informal networks, which is especially hard to use for newly hired employees, i.e. the employees that require the most knowledge support. Newly introduced is a team certification which is an inventory of the cross-functional teams' competence that could be useful from several aspects. A Product Guardian explains how he thinks the team certification can help:

"There are very few teams that have any deeper knowledge within any area, and it is probably unclear which competence they really have, especially among the new teams. I think that we also have unclear expectations on the teams and which competence they should have. I am cautiously optimistic to that the team certification can help here. I mean it is good because it
provides an inventory of the current state of competences in the
teams, but I would also like to add that with a certain
certification you also have the product responsibility in that
area. We have a few teams today which are on that level and
actually take that responsibility. I would like to focus the teams
clearer to a responsibility of certain areas."

The team certification could further work as a basis to visualize who has the
right competence, and by that formalize the knowledge networks, described
by Moe et al. (2014) as essential for providing support and allow for teams’
full responsibility of the feature development. A senior specialist describes
how this could work:

"I think that it would be a good idea to make the support
between the teams easier. There should be a list of who to turn
to when you have problems. [...] I don’t think there are such
lists today. Technical management could have such a list and
just describe what they work with. Maybe the Product Guardians
could be described there as well. And also the team certification
should be there, easy to access so that you can go directly to the
team with the right competence. The list must however be easy
to access, maybe on the wiki or something."

So while the organizational structure with market-driven feature
development seems to be an appreciated way of working, the coordination
and knowledge access between teams and departments must be better
handled to allow for knowledge sharing. As Moe et al. (2014) states, the
necessary knowledge support is higher for the employees with shorter time
at the company and the knowledge network is therefore even more
important.

Regarding the task-related driver for knowledge sharing described by
Ghobadi (2015) it would also be possible to decrease the necessary support
by distribute the features to the teams in a better way. One of the
impediments for this knowledge sharing driver was a lack of product care
items. Of course it is desirable to decrease the list of product care items
and by that getting rid of the technical depth. But the origin of the
impediments is rather that the guilt is built from the beginning which is
described as a result of lack of knowledge about the product and lack of
incentives to care about the product. To increase the deep knowledge
about the product and by the same time get incentives to protect the
product, the developers need to feel responsible for both the features and
the product.

Like Chandler (1990) states, that the strategic capabilities must influence
the functional capabilities to build the right knowledge, there must be
incentives to grow knowledge alongside the feature-development. Since the
organizational setting is a contradiction to the evolvement of deep product
knowledge according to Bredin and Söderlund (2011), the incentives should rather come from the distributed tasks. This is a way to integrate the necessary knowledge development into the daily work, which gets support from Davies and Brady (2000) and can encourage both cross-functional and cross-project learning and in time make the organizational learning more effective. At the same time, the organization is market-driven and gets revenue from delivered features, why the distributed tasks can’t be replaced by product care items. There is always a trade-off between the resource allocation, product knowledge and feature-flow. The cross-functional teams are focused and the teams remain together as a team for several features. The features are allocated to the teams with the background of being market-driven, prioritized from a feature backlog and then distributed to the teams depending on the teams’ current competence areas, and by that to opt for feature-throughput. This distribution cannot always be a perfect match to the teams, why the need for external support from specialist roles is inconsistent, and highly dependent on both the feature and the team composition. A senior developer describes a way to combine feature-development with building knowledge:

"I think that one should have a natural rotation among the employees, to spread knowledge. But at the same time I definitely think that one should have a home ground, so to say, where you feel comfortable with the product. And even if you work in other product areas for a while to learn other parts, you still know that you will come back to your home ground. I think that would increase the incentives to take responsibility for the product, because you will work there a lot and struggle with the problems often. If you build an own frustration over something, you are more likely to just fix it instead of putting it on a backlog. […] Perhaps a good idea would be to let the cross-functional teams have the responsibility for some software units and at the same time work with features related to those areas. I think that would integrate the product care into the daily work in a good way."

The natural rotation can address both the knowledge support between teams and the integration overlap between the system organization and the developing teams. The incentives for product responsibility could, as mentioned, be handled in different ways. One way is to have a formal responsibility according to the team certification, where a certain level of certification requires the employee to take responsibility and protect the product in that area. The other way is to distribute the tasks to the teams in a strategic competence growth manner which enables the teams to build a competence area in a limited area of the product, which automatically should give incentives to take care of the product in their own home ground. Either way, the important factor here is to integrate the
knowledge growth and product care into the daily work for the developers, and make it a natural way of working. Further there is a contradiction that the tasks, which are shown to affect the product knowledge to a high degree, are distributed by the Program Office which don’t have any formal competence development responsibility. One way to integrate the knowledge growth into the tasks distribution is to involve the human resource responsible into the feature distribution. In other words, the line management with the formal human resource responsibility should influence the task distribution, and thereby the knowledge growth, to a higher extent. Since the product knowledge is essential from the long-term competitiveness perspective, these strategic capabilities must influence both the functional capabilities and the project capabilities to be sustained and evolved, according to Davies and Brady (2000).

5.3 Suggestions for improvements

The suggestions for improvements are based on overcoming the challenges described by Sekitoleko et al. (2014), and structured by the knowledge-drivers described by Ghobadi (2015). The current state in the organization has witnessed similarities to the vicious circle described in Sekitoleko et al. (2014). Since the suggestion from their research is to break the vicious circle through mitigating the knowledge sharing challenge, the research by Ghobadi (2015) can be used as a base for focusing resources in certain areas that can drive the knowledge sharing forward. Research by Davies and Brady (2000) also describe that organizational learning is highly dependent on knowledge sharing and that the competence level in the organization can be increased by focusing on spreading knowledge. Related research such as Hillebrand and Biemans (2004) supports this statement and emphasize that if the internal integration works in a decent way, the competence level of the whole organization can be increased. In other words, enough support is found in literature for focusing on an increased competence level, which is described as a critical success factor, by enabling better knowledge sharing.

5.3.1 People-related drivers

The people-related drivers are very important since they tell about people’s feelings about the current state in the organization. However, in this case, they are not the origin of the problem but rather tells that the focus should be on overcoming the lack of knowledge by increasing the knowledge sharing and allowing for time to work in a limited subarea of the system for a sufficient amount of time. This relates to the subcategory capability-related drivers, and describe the current states of knowledge in the teams (Ghobadi, 2015). In other words it is the rationale behind this
case study, to find ways to increase the competence level and thereby get better feature-throughput. Another of the subcategories described by Ghobadi (2015) is the team perception drivers, including interdependency and clarity of the reward system. The case study witnesses that the interdependency is high because of both organizational factors and technical factors. The current state of knowledge in the organization also relates to the reward system, since it is feature development that is promoted, where knowledge building is perceived as secondary. The result of the current state of knowledge in the organization implicate that it would probably be beneficial to investigate the people-related factors on a regular basis to find the origin of the impediments.

5.3.2 Structure-related drivers

Organizational structure and market-driven agile feature development are not questioned because there is enough proof in literature to be considered a suitable choice (Lagerberg et al., 2013)(Petersen and Wohlin, 2009)(Hobday, 1998). It is even possible to gain benefits from agile principles also when they are just partially implemented (Lagerberg et al., 2013). This underlines the utility of agile methods even more given that they are implemented with care, since the large-scale environment raises new issues as described by Petersen and Wohlin (2009).

Even though the structure is right, the escalation paths for support between the teams could be improved by visualizing the competence of each team and to show which parts of the product they are currently working with. This would make it easier both to ask the right team for support and to understand which teams that are developing features with dependencies to each other. Such information would also support the corporate strategy of using cross-functional teams that can take the full responsibility of the features.

Regarding the internal integration described by Hillebrand and Biemans (2004), a higher overlap between the system organization and the developing teams would increase the knowledge exchange and provide the teams with a better overall system perspective and the same time provide feedback to the system organization. Further it could decrease the workload for expert roles, which instead can work more proactively and focusing on sustaining their expertise.

So as a summary of structure-related drivers, the agile methodology is the right choice, but it is possible to increase the possibilities of organizational knowledge sharing by visualizing the knowledge networks.
5.3.3 Task-related drivers

The features are highly dependent on the market and prioritized by that. The features shouldn’t be replaced by tasks like product care items, which would enable time for refactoring and get rid of the technical depth. The reason behind that is the fact that the features drive the revenue for the company. They should rather be distributed in a way that increases the incentives to care about the product and to integrate the knowledge building in the daily work. This way to integrate knowledge development into the project capabilities is described by Davies and Brady (2000) and will encourage both cross-functional and cross-project learning. The features could be matched to the teams’ team certification in order to optimize the throughput and on the same time give the teams the possibility to develop deeper understandings about certain areas of the product. This creates a home ground where the teams feel comfortable when developing and creates incentives to care about the product and in time decrease the number of product care items. This is supported by Enberg and Bredin (2013), which highlight the importance of working near the product on the operational level for a sufficient amount of time in order to increase the domain specific knowledge. Further it finds support in cognitive science, since it is easier to grow new knowledge upon earlier experience (Xu and Ramesh, 2009). As requested from the interviewees, it would also increase the motivation because it would increase the feeling of being able to contribute in the development process. To build a home ground where the team feels comfortable is especially appropriate when developing technical complex products in large-scale organization. This statement is based on Heikkila et al. (2013), which emphasize the difficulty of creating generalist teams that can develop features in all components of a product, because of the technical characteristics with many interdependencies.

5.3.4 Technology-related drivers

The technology-related drivers are to a big extent a matter of cost. However, a clearer architecture with well-defined components would increase the learning curve. It would also reduce the complexity of the product which is an effective way to decrease the time needed for problem solving, which characterize the daily work as a software developer. The decrease of complexity is also supported by cognitive science described by Xu and Ramesh (2009), which argues that it is most effective to focus resources on the contextual knowledge since it is the most effective type of knowledge in problem solving. Another way to reduce the complexity and increase the possibilities to actually understand the product would be to discard the UML modeling tool. Today it is very much a focus on the tool,
rather than the development, which once again is ineffective for learning and productivity (Xu and Ramesh, 2009). However, it would be a huge investment and deeper investigations need to be done. The subsystem in the product described as the most complex is up on the agenda for a possible replacement in the future. There is a prototype under development, where the modeling tool is replaced and the architecture should be clearer. This is a good example and could provide more clarity of whether the modeling tool should be replaced in the whole system. The replacement of the tool would also discard the use of the UML modeling tool for documentation. Anyway, since the tool is described as inappropriately cumbersome which leads to bad documentation, it hinders developers to get a fast overview of earlier implementations. Something that is requested in order to decrease the take-off distance in new areas is to have a well-documented overview easily accessible. The documentation could have a more lightweight alternative, for example on the wiki, which just describes the different software units on a high level, with their responsibilities and dependencies. This would also decrease the problems with uncertainty about the architecture and how different components affect each other. Another technology-driver impediment is the configuration documentation for the customers, which is a time-consuming task and gives very little in return. However, the activity of thinking about the possibilities to configure the feature, according to the implementation details is an appreciated activity from a learning point of view. The developers know what to write but the tool makes it a difficult task. An alternative solution is to let the developers send the documentation in plain text to someone that know exactly how the tools work, since the problem is caused by irregular use of the tool by the developers.

The test environment causes problems in product knowledge for the developers, since it is not obvious why an error occurs. Even in this area it is to a large extent a matter of cost. The most direct solution is to focus more resources on the test environment to make it more stable. Since this problem also relates to the lack of testing knowledge in the teams, an increased involvement by the Test Manager would also be beneficial.

As an overall perspective of the technology-drivers, they are aimed to make the daily work more effective for the developers by providing standard solutions in terms of tools and components, which goes very well in line with Nidumolu and Subramani (2002). Although, the fact that it hides important information about the product and thereby makes it more complex and hard to understand, harms the understanding of the product and results in a long-term threat to the deep product knowledge evolution.
Chapter 6

Discussion

This chapter discusses the results of the research from the perspective of significance of the results. It also outlines an evaluation of the validity.

6.1 Discussion

The research investigated challenges in sustaining and evolving deep product knowledge in a large-scale organization. The findings show that domain specific knowledge that only can be developed at the specific workplace is harder to deal with than more general technical competence. At the same time, the domain specific knowledge is considered as a critical success factor in the development process, why resources should be focused on evolving it. The reason behind the importance is that the knowledge affects many parts of the organization. This is concluded from a mapping into the challenges described by Sekitoke et al. (2014). These are the planning challenge, the task prioritization challenge, the knowledge sharing challenge, the code quality challenge, and the integration challenge, which are all represented in the result. All these challenges are interdependent and can create a vicious circle. The suggestion from Sekitoke et al. (2014) is to start to mitigate the knowledge sharing challenge, in order to break the vicious circle and increase the competence level, which will be beneficial for the whole organization. The findings further show that the knowledge is affected by four factors: people-related, task-related, structure-related and technology-related. These knowledge sharing drivers are a framework developed by Ghobadi (2015) with its origin in a systematic literature review, aimed to allow organizations with cross-functional teams to overcome the coordination challenges across distributed sites. This is the same organizational setting as in this
research, why the generalized method can be considered suitable.

The challenging element for allowing knowledge growth is to integrate the knowledge evolution into the daily work without affecting the feature-flow negatively. Since a bottom-up approach is used in the research, where people from the operational level of the company are interviewed, factors from the corporate level such as long-term strategy and economic constraints are not considered. However, the results of the data collection are impediments that are actually experienced as obstacles for the growth of product knowledge and should be considered anyway.

In addition to finding impediments and grouping them into the knowledge-driver framework, the impediments were further investigated to find interdependencies and the actual origin of the experienced problems. The solutions are based on the findings from the data collection, a workshop with the focus group and the researcher’s interpretations. Even in this case, corporate level factors are to some extent neglected.

The significance of the result for the company should be considered high, since all impediments are experienced at the operational level. To increase the competence level and evolve deep product knowledge within the organization, the recommendations from the previous chapter should be considered. The main focus to increase the competence level is to give incentives to integrate the knowledge growth into the feature development. The recommendations are built upon that, and by following the suggestions for improvement, most certainly the competence level would be increased.

### 6.2 Evaluation of validity

This section evaluates the method used from the four perspectives given by Yin (2012): construct validity, internal validity, external validity and reliability.

#### 6.2.1 Construct validity

As Yin (2012) states, the interview questions must be interpreted the same way by the researcher and the interviewees, and relate to the research questions in a good way. This was handled by starting each interview session with an introduction part. Prior to each interview session, the interviewee got an introduction document (see appendix C) with the overall objective of the study. The interpretation of this document was also discussed during the introduction of each interview session, according to the interview plan (see Appendix D). This enabled a shared understanding.
6.2. EVALUATION OF VALIDITY

of the topic between the researcher and the interviewee. The construct validity is also considered through the process of breaking down the aim of the study into research questions relating to three subcategories. Those subcategories, product knowledge, distribution of tasks, and resource allocation were further broken down into interview questions.

As Yin (2012) suggests, triangulation can be used to increase the construct validity. This was considered through an analysis and comparison of the collected data with the state of the art. The draft case study report is also reviewed by both the company and key informants at the university.

6.2.2 Internal validity

The threat to the internal validity is primarily because of false interpretations during the analysis phase Yin (2012). In order to decrease the risk for internal validity, the causal relationships found were compared to state of the art and also compared to earlier case studies at the same company. The findings were also reviewed by a focus group during two occasions, where possible misinterpretations by the researcher were discussed. Findings that were not well-defined, and where the origin of the phenomenon was unclear were eliminated from the study. This decreased the number of findings but the remaining findings should be considered as more valid.

6.2.3 External validity

The external validity is a matter of research design and considers whether the findings are generalizable or not (Yin, 2012)(Runesson and Höst, 2008). Since Yin (2012) describes that the initial research questions can influence this aspect, the design of the research questions was considered from the perspective of being generalizable. Even though this aspect was considered, the research questions are based on the work-settings and organizational factors at the company. This includes a large-scale organization with cross-functional teams developing highly complex market-driven software in an agile way of working. However, those factors are true for many developing firms, why the result can be considered generalizable in a limited domain. Further, suggestions for improvements were built upon the found impediments. These should be considered as unique for the company under investigation and are not directly generalizable. However, if a similar study is conducted, the outcome of possible solutions could be used to broaden the perspective of how certain impediments for knowledge growth can be resolved.
6.2.4 Reliability

The reliability concerns whether the study could be repeated by someone else with the same result (Yin, 2012). In order to increase this validity factor, a case study protocol was used according to the recommendations by Yin (2012). All interviews were recorded, summarized and collected in a case study database. The following analysis phase were conducted through a step-by-step recommendation by Braun and Clarke (2006) and described in this report. Even though the method used is documented and reported, the findings are primarily based on the researcher's interpretation of the data collected. This reliability threat can’t be neglected and it is possible that someone else with another background and experience than the researcher should interpret the situation in another way.

6.3 Ethical evaluation

Concern is given to the ethics of conducting a case study including human subjects, according to the suggestions by Yin (2012). All interviewees got prepared for the meetings by receiving an introduction letter including information of the research (see appendix C). During the beginning of each interview session, the informants got the opportunity to approve or disapprove a recording of the interview (see appendix D). They were further informed that their answers should be handled with care of their anonymity, and they also got the possibility to review their answers.

From the perspective of the overall results of the case study, i.e. this report, two key informants from the company reviewed the draft version until agreement of the information in the report was settled. This resulted in that the company for which the case study is conducted supports the publication of this report. Anyway, the researcher himself takes the full responsibility of all interpretation made during the research in the published paper.
Chapter 7

Conclusions

This chapter highlights the conclusions drawn from the research from the perspective of the research questions. Finally some propositions for future research are described.

7.1 Conclusions

The product knowledge evolvement in a large-scale organization is a troublesome concern because there are many factors that affect the situation. There is always a trade-off between short-term feature-flow and knowledge development. The feature-flow can’t be overlooked in order to build competence, since the revenue comes from delivered features to the market. Instead, the product knowledge development should be integrated into the daily work, by creating incentives to evolve the product knowledge alongside the feature development.

The research has shown that product knowledge is essential for feature throughput when developing complex products in large-scale. As the research questions implies, the product knowledge and necessary support to evolve the knowledge is dependent on both the distributed tasks and by resource allocation, including both team composition and the surrounding roles. The research has shown that the structure-related drivers for building knowledge should except from resource allocation also include a better knowledge exchange between the system organization and the developing teams. The structural limitations could also benefit from visualizing the teams capabilities in a better way, to allow for easier access of expertise between the teams. The task-related drivers should also, except from distribution of features, include incentives for taking responsibility of the product. The responsibility to care about the product
could be integrated into the feature development by building a home ground for the teams, in which they feel comfortable when working so the feeling of contribution and motivation is increased.

The research has also shown that technological factors are important to consider for enabling deep product knowledge. These technology-related drivers include the architecture of the product, development tools, and tools for documentation. Documentation is in contrast to the Agile Manifesto an important event in a large-scale organization, where the verbal interaction is limited by geographical constraints. Well documented implementations is also a way to decrease the take-off distance for developers that continue on others’ work, and to spread knowledge. The reasons behind architectural choices and development tools are often to make the daily work easier for the developers by reducing the complexity through standardization. In this case, the unclearly defined components with many dependencies, together with the development tool used which abstract away important information makes the complexity rather increase and thereby hinder the deep product knowledge. The motive should rather be to strive for ways to reduce the complexity to enable a better learning curve and optimize the feature-throughput.

Since the task distribution is shown to be the factor that influences the product knowledge to the greatest extent, particular consideration of how the tasks are distributed to the cross-functional teams should be taken into account. If the product knowledge is considered important for the long-term performance of the company, this factor should be a more important parameter in the task distribution.

### 7.2 Future work

The implication of this research is that large-scale software development organizations should investigate the current state of knowledge on a regular basis to find impediments for knowledge-drivers, in order to mitigate them and overcome the vicious circle that can affect many parts of the organization. What remains unanswered is how to work more proactively to reduce the risks of ending up in the vicious circle in the first place.

To take this explicit research further, a cost analysis of the recommended solutions should be performed. This is especially true when it comes to technology-drivers for knowledge sharing, which often are associated with high costs.

Further research is needed in how to integrate incentives for evolving product knowledge in a market-driven environment, where the feature-flow is essential for revenue. Such incentives require corporate capabilities and
project capabilities to be integrated in a way that allows functional capabilities such as long-term human resource management to be a natural part in the daily work.
Bibliography


Appendix A

Questionnaire - Team members

Background

• What’s your education?
• Describe your role at the company
• What are your main responsibilities?
• For how long have you been working here?
  – For how long in your current position?
• Describe your team composition

Product knowledge

• Describe your perspective of product knowledge in the feature-development
• How would you like to do to improve the overall competence level?
• Regarding knowledge: What’s your interpretation of the most important factors to be able to perform the work in the teams?
• Is there any area which is more problematic than the other?
• What’s your interpretation of your team’s best competences?
• How could the different knowledge areas best be improved?
• Describe how your own knowledge within the product is evolving?
• How do you work to increase your knowledge?
• Relate your current state of product knowledge with one year ago.
• Describe how you can maintain your knowledge.
• Would you say that your knowledge primary is broadened or deepened?
• Describe and optimal environment to evolve deep product knowledge
  – How is the current situation related to that?
  – How could the most important factors be improved?
• Do you have any additional thoughts about product knowledge?

Distribution of tasks
• Describe your view of the distribution of tasks in relation to product knowledge.
• Describe what you think is the most important factors to consider when the features are distributed to the teams.
  – How do you think that works today?
• Describe the main problem areas in evolving knowledge caused by the distribution of features.
  – How could that be improved?
  – What could you contribute to from your role?
• Do you have understanding of why the distribution works like it does?
• Describe your tasks in relation to what you believe you should spend time on.
  – Which tasks would you like to spend more time on?
  – Which tasks would you like to spend less time on?
• Describe the optimal environment for distribution of features to enable knowledge growth.
  – How is the current situation related to that?
  – How could the most important factors be improved?
• Do you have any additional thoughts about the distribution of tasks?

Resource allocation
• Describe your view of the resource allocation in the teams and the supporting roles, to enable knowledge growth.
• What are the most important factors to consider in the team composition?
• Do you think that your team is lacking any particular competence?
  – Are there any particular activities during the development process where sufficient knowledge is missing?
    * How could this be improved?
  – If you could choose one person with any particular competence to your team, what would that be?
• Describe what could be improved to enable better knowledge growth in the product.
• Describe how the supporting roles contribute to the knowledge evolution in your team.
  – Do you think that you get enough support?
• Describe your view of the knowledge exchange between the teams.
• Describe the optimal situation in terms of resource allocation to enable knowledge growth.
  – How is the current situation related to that?
  – How could the most important factors be improved?
• Do you have any additional thoughts about the resource allocation?

Closure
  • All three areas are now covered. Is there anything else you would like to add?

Summary and discussion Possibility to change, reformulate or withdraw quotes.
Appendix B

Questionnaire - Supporting roles

Background

• What’s your education?
• Describe your role at the company
• What are your main responsibilities?
• For how long have you been working here?
  – For how long in your current position?
• How close to the teams are you working and what’s your influence on them?

Product Knowledge

• Describe your perspective of product knowledge in the feature-development
• How would you like to do to improve the overall competence level?
• Regarding knowledge: What’s your interpretation of the most important factors to be able to perform the work in the teams?
• Is there any area which is more problematic than the other?
• What’s your interpretation of the teams’ best competences?
• How could the different knowledge areas best be improved?
• Describe how your own knowledge within the product is evolving?
• How do you work to increase your knowledge?
• Relate your current state of product knowledge with one year ago.
• Describe how you can maintain your knowledge.
• Would you say that your knowledge primary is broadened or deepened?
• Describe and optimal environment to evolve deep product knowledge
  – How is the current situation related to that?
  – How could the most important factors be improved?
• Do you have any additional thoughts about product knowledge?

Distribution of tasks
• Describe your view of the distribution of tasks in relation to product knowledge.
• Describe what you think is the most important factors to consider when the features are distributed to the teams.
  – How do you think that works today?
• Describe the main problem areas in evolving knowledge caused by the distribution of features.
  – How could that be improved?
  – What could you contribute to from your role?
• Do you have understanding of why the distribution works like it does?
• Describe your tasks in relation to what you believe you should spend time on.
  – Which tasks would you like to spend more time on?
  – Which tasks would you like to spend less time on?
• Describe the optimal environment for distribution of features to enable knowledge growth.
  – How is the current situation related to that?
  – How could the most important factors be improved?
• Do you have any additional thoughts about the distribution of tasks?
Resource allocation

- Describe your view of the resource allocation in the teams and the supporting roles, to enable knowledge growth.

- What are the most important factors to consider in the team composition?

- Do you think that any particular competence is lacking in the teams?
  - Are there any particular activities during the development process where sufficient knowledge is missing?
    * How could this be improved?

- Describe what could be improved to enable better knowledge growth in the product.

- Describe how the supporting roles contribute to the knowledge evolvement in your team.
  - What opportunities are there to support the teams when needed?

- Describe your view of the knowledge exchange between the teams.

- Describe the optimal situation in terms of resource allocation to enable knowledge growth.
  - How is the current situation related to that?
  - How could the most important factors be improved?

- Do you have any additional thoughts about the resource allocation?

Closure

- All three areas are now covered. Is there anything else you would like to add?

Summary and discussion Possibility to change, reformulate or withdraw quotes.
Appendix C

Introduction letter

The interview is intended to be the basis for an assessment of how the teams can evolve deep product knowledge. Based on the current situation, problem areas that hinder development are identified and suggestions for improvement should be developed. The interview will deal with three main areas where you get the chance to contribute based on your own perspective. The three areas are:

1. Product Expertise
2. Distribution of Features
3. Resource Allocation

**Product Knowledge**  This topic mainly considers how learning about the product under development works. The goal is to identify the impediments for optimal learning in the product. We will treat both the perspective of the individual and the team as a whole.

**Distribution of Features**  This area deals with how different features are distributed to the teams and how you think it affect the product knowledge. This topic also covers the knowledge growth in relation to incentives given for competence development. We will focus on identifying if the features goes in line with what is expected in terms of knowledge development.

**Resource allocation**  This topic covers team composition, including the supporting roles and how it affects the knowledge growth. Questions to consider are whether the resources are sufficient in order to develop the features you expected to do. Furthermore, you sometimes need to go outside your comfort zone. Are the necessary support obtained in these
situations?

After the interview, I will give you my interpretations of the conversation where you have the chance to provide feedback and correct formulations. After I have compiled the interview I will send it to you for revision in which you once again have the chance to review my interpretation of your answers.

Thank you for your participation
Henrik Andersson
Appendix D

Interview introduction

Intro
• Before we start, have you read the introduction document?
  – Do you have any questions regarding that?

Disposition
1. Introduction
   (a) About myself
   (b) Why the study is conducted
2. Arrangements for the interview
   (a) Background questions
   (b) Main part with questions including three perspectives
      i. Product knowledge
      ii. Distribution of tasks
      iii. Resource allocation
   (c) After we have gone through all main parts, you will have the possibility to add additional thoughts.
   (d) I will give you my own interpretation of the interview where you have the chance to change, reformulate or withdraw your answers.
3. Closure
(a) After the interview, I will write a summary of the answers which I will send to you. You will then have the possibility to confirm that you are comfortable with all the answers.

Introduction questions

• Do you feel comfortable if I record the interview?
  – You will be anonymous and the record will be deleted after the end of the case study

• Do you have any questions before we start the interview?
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framstida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extra-ordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: http://www.ep.liu.se/

©Henrik Andersson