CHALMERS
UNIVERSITY OF TECHNOLOGY

Exploring Cost Drivers in Software Development

A Case Study at a Large Automotive Multinational

Master's Thesis in the Master's Programme
Management and Economics of Innovation

MATTIAS KARLSSON
LINUS SCHÖNBECK

Department of Technology Management and Economics
Division of Entrepreneurship and Strategy
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018
Report No. E 2018:016

MASTER'S THESIS E 2018:016

Exploring Cost Drivers in Software Development

A Case Study at a Large Automotive Multinational

MATTIAS KARLSSON
LINUS SCHÖNBECK

Tutor, Chalmers: Charlotta Kronblad
Tutor, Company: Jonas Adolfsson

Department of Technology Management and Economics
Division of Entrepreneurship and Strategy
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Exploring Cost Drivers in Software Development
A Case Study at a Large Automotive Multinational

MATTIAS KARLSSON

LINUS SCHÖNBECK

Master's Thesis E 2018:016

Department of Technology Management and Economics
Division of Entrepreneurship and Strategy
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Telephone: + 46 (0)31-772 1000

Chalmers Reproservice
Gothenburg, Sweden 2018

Acknowledgements

Without the help of several people, this thesis would not have been possible. We would like to thank Jonas Adolfsson at the department for his help and thoughts during the study. Charlotta Kronblad at Chalmers University of Technology has our deepest gratitude for helping guide us through this project, always providing thoughts and feedback whenever they were needed. Last but not least, we extend our thanks to everyone at the department who we interviewed or who helped recommend potential interviewees, your expertise and insights were invaluable to this thesis.

Gothenburg, May 31st 2018

_____ _____

Mattias Karlsson Linus Schönbeck

Abstract

Knowledge of what drives costs in software development is crucial for organisations to successfully operate such divisions, yet the high speed of change in recent years makes it an area where available research is limited. Specifically, the way costs of agile development projects differ from traditional project processes has received little attention from researchers. Knowledge in the area could help managers better understand the intricacies of software development and how it differs from other project work.

This study addresses the issues of cost driver identification and exploration through a case study of a software development department at a large Swedish multinational in the automotive industry. By conducting interviews with people throughout the department, cost drivers were identified and divided into 4 categories; Organisation, Staff, Code complexity and Code Quality. The 4 categories are then divided into sub-categories. These have been analysed with the help of a theoretical framework on software development and large influencing factors in it.

The conclusion from the case study is that the major drivers of costs in the department are personnel turnover and complexity in the code. Recruiting new employees gives short-term learning costs, both for the new employees from the time experienced workers spend as teachers. New recruits will further make more mistakes for a time which creates more bugs and errors that take time to fix. Complexity in the code is another core driver as the legacy over the years of patching the code together and not focusing on readability and maintainability makes the code very tricky and expensive to work with.

This paper's contribution to the scientific area is twofold. First, it extends the reader's knowledge about software development in the context of organisational growth and time pressure. Secondly, it provides an approach for identifying cost drivers within this context.

Table of Contents

1 Background 1

1.1 Purpose 3

1.2 Disposition 3

2 Theoretical Framework 4

2.1 Identification and Analysis of Cost Drivers 4

2.2 Intellectual Capital in Software Development 4

2.2.1 Capabilities and Knowledge Required from Software Developers 4

2.2.2 Experience and Turnover in Software Development Teams 5

2.3 Introducing Agile Development into a Large Organisation 6

2.3.1 Processes, Priorities and the Importance of Management 6

2.3.2 Customising the Approach to Smooth the Transition 7

2.3.3 Creating a Work Environment for Agile 7

2.4 Software Quality and Complexity 8

2.4.1 Complexity Growth due to External Pressures 8

2.4.2 Architectural Challenges in Software Development 9

2.4.3 Maintaining and Updating Software Systems 10

2.4.4 Internal Quality and Technical Debt 11

3 Method 13

3.1 Chosen Methodology and Justification 13

3.2 Reliability and Validity of the Study 13

3.3 Literature Study 13

3.4 Data Collection 14

3.4.1 Case Description and Context 14

3.4.2 Secondary Data 15

3.4.3 Interviews 15

3.5 Data Analysis 16

4 Findings - Identified Cost Drivers at the Department 17

5 Analysis of the Findings 26

5.1 Factors Impacting Productivity at the Department 26

5.1.1 Coordination between Functions 26

5.1.2 Changes in the Organisational Structure at the Department 27

5.2 The Risk of Increased Personnel Turnover at the Department 27

5.2.1 Reasons that Personnel Turnover Might Increase 28

5.2.2 The Costs Associated with a Higher Turnover 29

5.3 Software Quality and Complexity 29

5.3.1 Demand and stress induced complexity 29

5.3.2 Maintenance inefficiencies and costs 30

6 Conclusions 32

6.1 What are the main drivers of costs in software development at the department? 32

6.2 How do these affect and reinforce each other? 34

6.3 Actions for lowering the impacts from the cost drivers 35

7 References 36

1 Background

The importance of software has been increasing for all organisations, and software development activities are becoming more important also for companies who traditionally have had limited exposure to IT (Porter & Heppelmann, 2015). The scope of these activities is therefore expanding, and they are becoming larger parts of their corporations' budgets. This trend can be identified at the Swedish multinational studied in this report, which twenty years ago developed a software product, including hardware components, involving just ten engineers. Over the years this product was split up and expanded, and as the number and complexity of products has increased, so too have the amount of people involved. As a step in this, responsibilities were parcelled out across the organisation and the department studied was started, tasked with developing one part of the software. This department has since grown in size, particularly in recent years, and awareness of the reasons for this growth is low. Recently, the department has introduced agile methods to their software development and organisational changes aimed at encouraging its adoption throughout.

When planning projects, the general mind set at the organisation is to consider the constraints of project management. Maylor (2010) explains that not all three parameters of cost, time and quality can be prioritised, but that one will inevitably suffer in an effort to maximise the other two, especially if the scope of the project changes. This relation can be illustrated by the iron triangle, which shows performance according to these three parameters. In this case, the department's major project currently is the development of a new generation of their largest product platform, which has a fixed release date. This release date has been decided by external factors and the department as such have no ability to influence it. The quality dimension for the new platform is also fixed, as it has to be stable and tested correctly, with a predefined quality level. These two factors being fixed in the triangle means that the third dimension; costs will not be prioritised. As such, the organisation has tried to solve issues in the project by dedicating more resources to it. An illustration of this can be seen in figure 1.1.

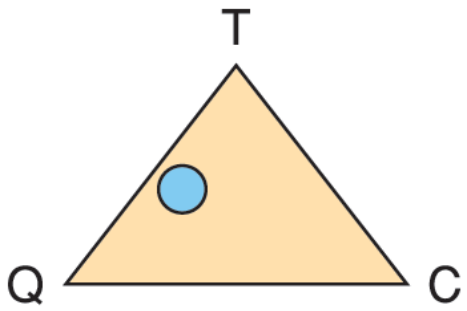

Figure 1.1. Iron triangle example illustrating the demanded quality and time for the department which has led to increased costs. (From Maylor, 2010).

The new platform's scope was also set early on; it needs to have the same functionality as the previous generation, with some additional functionality added as well. The fixed nature of time and quantity thus exist as a backdrop for this study, without it the findings would likely be completely different. It was noted early on by an interviewee that *"if we had more time none of this would really be a problem"*, in line with this reasoning. The rest of this report therefore assumes these constraints to be there, and the theoretical framework is built around them.

The initial estimations for the platform project have proven to be incorrect in terms of the man hours required to achieve the required scope and quality. New estimations have been made several times, and each time the total amount of effort needed has been increased. This increase comes from two factors; underestimating the work needed and increases in the scope of the project. These estimations can be seen in figure 1.2.

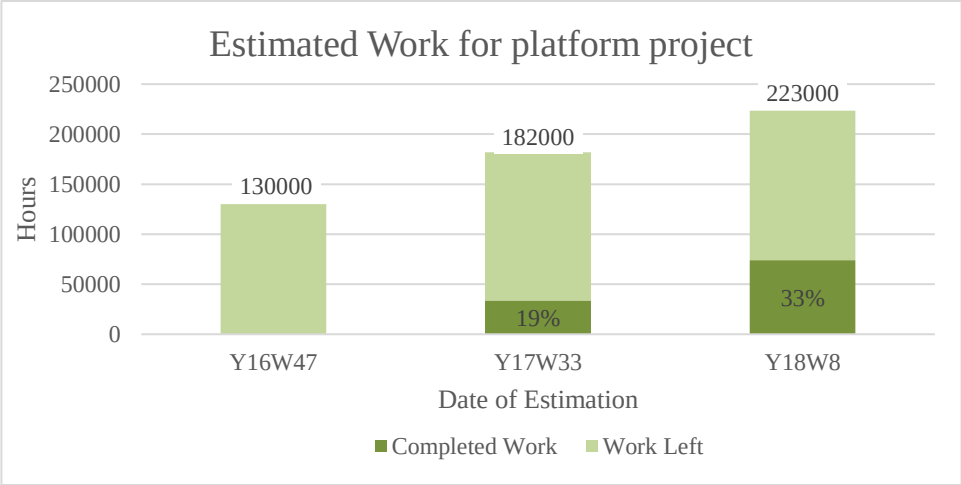

Figure 1.2. Estimations of total man hours needed for the new platform project.

Due to this increase in estimated effort needed, and in accordance with the priorities made in terms of the iron triangle, adjustments to the increased output demands have been made in the form of increasing staff, by hiring both new employees and consultants from external firms. Some expansion was already accounted for in the initial estimates, but these have been exceeded greatly. The department has grown in size and in January 2018 the number of workers exceeded 250 people. The growth of workers for the last years is illustrated in figure 1.3.

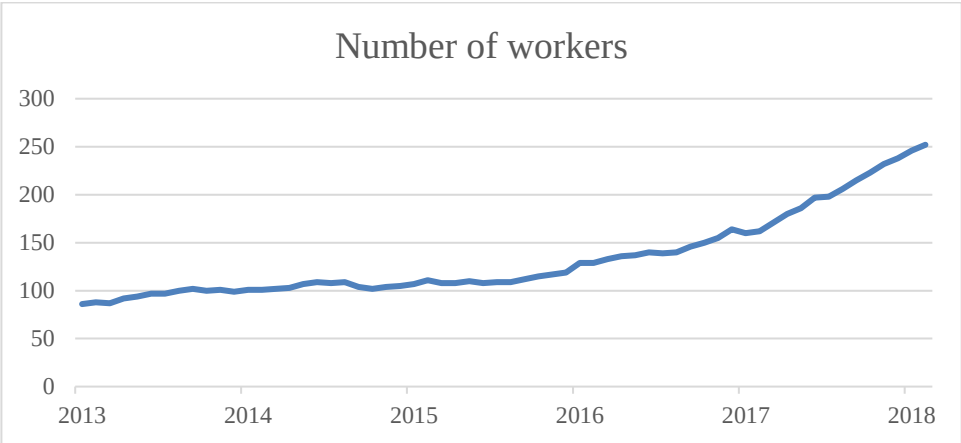

Figure 1.3. Number of full-time equivalents at the department each month.

Despite this increase in personnel, however, output has not increased to the levels expected. Management at the department is unsure as to the reasons behind this and are questioning whether the average productivity of workers has gone down. There has been some theorising that costs have had an exponential growth without output increasing to the same extent. Based on this, there is a need for an investigation into the costs of the department, and how these have increased.

Cost control is crucial for any department. As said by James Harrington: “Measurement is the first step that leads to control and eventually to improvement. If you can't measure something, you can't understand it. If you can't understand it, you can't control it. If you can't control it, you can't improve it" (Levy, 1999). Achieving cost awareness can thus be seen as a crucial first step for all companies.

1.1 Purpose

Explore cost drivers for a software development department with an increased demand of output.

Despite the importance of cost awareness as described earlier, there is a lack of research available related to total costs for software development, particularly during times of growing demand. There is thus a need for research aiming to identify and understand cost drivers in this area. This study has therefore been conducted in an attempt to start bridging this particular gap in research. This will give management better insight of the reasons behind the increasing costs and identify areas with potential for improvement. To reach this purpose; two research questions have been formulated:

Q1: *What are the main drivers of costs in software development at the department?*

Q2: *How do these affect and reinforce each other?*

1.2 Disposition

Chapter 1 provides a background to the study and the problem it is meant to explore, and defines its purpose and research questions.

Chapter 2 contains the theoretical framework which is later used to analyse the findings.

Chapter 3 describes the methodologies used throughout the study and how they have helped answer the research questions. It further explains the methods used to gather and analyse the data and evaluates the validity and reliability of the study.

Chapter 4 presents the findings from the study. This is done in the form of a table with quotes and interpretations.

Chapter 5 analyses the findings with a basis in the theoretical framework. It also discusses potential ways to improve on the current situation.

Chapter 6 concludes the thesis by answering the two research questions.

2 Theoretical Framework

The following part will present the theoretical framework used for analysis of the findings. It starts by explaining cost drivers and continues with sections on important factors in software development that affect the cost of development; either directly or via an effect on productivity, have been included, as lowering productivity in turn increases costs. First, a section on the most important personnel factors is presented, followed by a section concerning implementation of agile development and the issues associated with it. Third, the quality and complexity dimensions of software are covered, explaining how these two factors add significant costs to development. The constraint of time explained in section 1 exists as a backdrop for the entire framework; all of the effects would have different impacts were this constraint to be lifted.

2.1 Identification and Analysis of Cost Drivers

A cost driver has been defined as an activity that results in increased costs (Foster & Gupta, 1990), but for the purpose of this study a wider definition was used, also including factors that cause activities to use up more time. The concept was originally developed as part of the Activity-Based Costing (ABC) method, but the usefulness of cost driver-identification has been recognised outside of its original purpose. Nanni, Dixon & Vollman (1992) suggested that many organisations did not implement a full ABC system as most of the potential benefits were already found in the cost driver analysis. As such the identification and the analysis of cost drivers are useful for the purposes of this study, even though it will not cover later stages of the ABC-method.

2.2 Intellectual Capital in Software Development

Within software development, an organisation's main asset is its intellectual capital; the knowledge of its employees (Rus & Lindvall, 2002). As such, ensuring that the value of intellectual capital remains the same, or ideally increases, is a major objective and challenge for these organisations, and has been seen as a source of success (Nahapiet & Ghoshal, 2000). The challenge becomes increasingly important as organisations grow in size, both in terms of their products and number of employees. In small organisations, it is possible for team members to use personal knowledge and experience, or go via informal contacts to get knowledge needed to make decisions (Rus & Lindvall, 2002). But as more and more people are involved in development of more complex functionality, this process becomes both harder and less efficient.

2.2.1 Capabilities and Knowledge Required from Software Developers

The results of software development projects strongly depend on the competencies of the individuals involved. Faraj & Sproull (2000) stated that expertise (which was used as a synonym for competency throughout their paper) is the most critical resource for software development teams. Its importance has further been stated by others, including Wagner &

Ruhe (2008), in whose study on productivity factors for software development "Programmer Capability" was the most commonly identified factor. Without this factor, software teams suffer from both low output and quality.

Rus & Lindvall (2002) mentioned two categories of knowledge that are crucial for software development; technical and business domain knowledge. Technical knowledge refers to the capabilities one would take for granted in a software developer, such as programming and testing skills. Business domain knowledge refers to knowledge about the industry in which the development is taking place, including product knowledge and the customer's business goals. The existence and coordination of both these knowledge types was deemed highly important; it is not enough for a development team to have one or the other. Faraj & Sproull (2000) provided empirical results suggesting that coordination of specialised knowledge, both product and domain, is a significant factor in explaining team performance, specifically as they relate to efficiency in terms of project cost and time-to-completion.

2.2.2 Experience and Turnover in Software Development Teams

In a large empirical study comparing exceptional and non-exceptional software engineers, Turley & Bieman (1995) concluded that experience is a significant predictor of performance in this field. This is especially true when the experience has been acquired in the same organisation where the person currently works, implying that keeping employee turnover low is very important. The relation between experience and competency is not only true for individual competency, but also for the team as a whole. Huckman, Staats & Upton (2009) found that teams whose members have worked together before have a higher performance, both in terms of quantity (lines of code) and quality. Others have found that team performance, especially regarding product quality, is affected by teamwork quality, which in turn is affected by team member turnover (Lindsjørn, Sjøberg, Dingsøyr, Bergersen & Dybå, 2016). Furthermore, team member turnover has been found to have a disruptive effect on teams in general, impacting productivity (Melo, Cruzes, Kon & Conradi, 2013). Negative effects such as a reduced capacity to deliver due to energy being put to teaching the new member(s) and losing the knowledge held by those members who leave the team was seen. There is also a positive effect however, in that new members bring new ideas and solutions which can be helpful. A trade-off thus exists, and Melo et al. (2013) emphasised the importance of being aware of this, and managing it.

Manawadu, Johar & Perera (2015) identified seven technical competencies relevant for software engineers. They then showed that inexperienced developers have negative gaps in all identified technical competencies related to software engineering, and concluded that adequate training and introductions at the workplace are necessary in order for recruits to become successful at their jobs. Melo et al. (2013) found that a major issue for productivity in agile teams is 'the new guy', who requires significant time and resources before he is able to positively contribute to the team. This is in line with Brooks' Law, which states that adding manpower to a late software project makes it later (Brooks, 1995). Later studies have at least partially confirmed its validity (Hsia, Hsu & Kung, 1999), and it is now seen as an established fact in software development circles. Ensuring that new employees is given proper training and is kept on board after they become productive can thus be seen as very important.

2.3 Introducing Agile Development into a Large Organisation

Boehm & Turner (2005) stated that most large organisations have difficulties implementing agile methods due to organisational barriers, which they broke down into three categories; development process conflicts, business process conflicts and people conflicts. Dikert, Paasivaara & Lassenius (2016) conducted a systematic review of literature on success and failure factors for agile implementation in large organisations and identified several factors that have caused implementations to fail as well as succeed. The most common reasons for failure can be found in the categories of change resistance, difficulties in implementation and integration of non-development functions. On the other hand, management support, choosing and customising the agile approach as well as mind set and alignment are the most common categories in terms of success factors.

2.3.1 Processes, Priorities and the Importance of Management

Development process conflicts concern merging agile and standard industrial processes (Boehm & Turner, 2005). In particular, the issue of delivery was brought up; agile focuses on quick delivery of functionality, whereas most traditional organisations use long term plans to optimise development. This issue was also mentioned by Dikert et al (2016), as part of the issues surrounding integration of non-development functions. Some activities, such as product launch related ones, are characterised by long lead times which require producing units to commit to deliver a specific set of functionalities early on. This is in direct conflict with the emphasis that agile methods have on flexibility (Fowler & Highsmith, 2001). Examples presented include one case where functionality in the final product differed from the pre-produced marketing material, which resulted in having to create new material with short notice (Dikert et al., 2016). Boehm & Turner (2005) elaborated that in order to combine these two approaches team leaders who are competent in both ways are needed despite them being a rare commodity, and recommended realigning existing processes to work better in an agile context, such as redefining milestone reviews, as well as building processes from the ground up instead of top-down. This last part requires work in assessing the current processes, and determining which parts of them to keep and what to change.

Business process conflicts concern the differences between traditional and agile approaches in the everyday business. Boehm & Turner (2005) explained that uncertainty levels are higher in an agile context, and that the business processes in large organisations usually are not structured to deal with this uncertainty. Additionally, the way HR functions are organised is often not compatible with agile, as agile developers require more skills and experience in order to perform, as well as being empowered to take decisions on their own. In order to solve this, organisations are recommended to identify where incompatibilities can be found and eliminate them, as well as addressing potential HR issues and ensure that support functions are run in a way that accepts the uncertainty of predictions in agile methods.

Under the term people conflict, Boehm & Turner (2005) covered issues stemming from management and logistics within an organisation. Managerial attitudes in large organisations usually stem from manufacturing and emphasise control and role division for employees. This is incompatible with the multi-dimensional characteristics of agile developers, who require high autonomy to perform. Additionally, resistance to change is common in the management of large companies, something which Dikert et al. (2016) also mentioned under their category

"change resistance". They clearly emphasised the importance of ensuring everyone is on board in order for the organisational change to work, both managers and workers. Management support and the education of managers are both important factors in many successful implementations, failing to educate managers lead to them feeling left out and increasing resistance in multiple studies included in Dikert et al. (2016). On the other hand, educated managers with a commitment for the change help assure teams that it is necessary and a good thing.

2.3.2 Customising the Approach to Smooth the Transition

Difficulties in implementation as referenced by Dikert et al. (2016) mainly stem from a failure of properly customising the methods used to the organisational context. Failing companies either tried to implement strictly according to some available guidelines or framework, or skipped crucial practices when trying to create a specialised solution. Choosing and customising the agile approach was an important factor in successful cases, organisations that spent time choosing an approach carefully before implementation were more successful, as was those who customised the chosen approach. Even at a team level, those who modified practices performed better than those who simply adopted an approach. The customisation should be done with regards to the current organisational environment in order to simplify implementation, and leave room for teams to adapt the solution to fit their needs (Dikert et al., 2016). Leaving too much freedom to the teams can make it difficult to compare work or relocate people, however, and as such it is important to find the right balance there.

In order to achieve the right mind set and alignment for success in an agile context, Dikert et al. (2016) explained that emphasis need to lie on the principles of agile as opposed to specific practices or mechanics. They claimed that when people understand the values it is easier to motivate them to put in the extra work needed to ensure a successful implementation. Aligning the surrounding organisation with the new ways of working is also important, in order to avoid unnecessary work to fit the agile processes to external, non-agile work.

2.3.3 Creating a Work Environment for Agile

It has been proven that agile teams perform better when co-located (Melo et al., 2013), and they also require specific workspaces, which often clash with "efficient" spacing according to managers (Boehm & Turner, 2005). The keys to solving these issues are found in education; stakeholders (especially top management) need to be taught about agile, its strengths and weaknesses and what support is needed from them to ensure successful implementation. This opinion was shared by Misra, Kumar & Kumar (2010), who further stated that adopting agile requires changing the entire organisational culture and deliberated that this will be more difficult in larger organisations.

Brennan, Chugh & Kline (2002) showed that when moving to an open office environment, employee satisfaction went down in four dimensions; physical environment, physical stress, co-worker relations and perceived job performance. They also showed that this dissatisfaction was not temporary, but stayed with employees long after the move. As reasons for this, they in particular mentioned an increase in the number of disturbances and distractions, which employees found counterproductive.

Their findings were further backed up by Kim & Dear (2013), who showed that private offices "clearly outperform" (p. 1) open offices in environmental quality. They further showed that the benefits often touted as an argument for open offices, in particular enhanced interaction between workers, did not outweigh the performance losses from increased noise and decreased privacy. Both sets of authors recommend organisations to provide many so called break out rooms, where conversational tasks can be performed without disturbing the open office (Brennan et al., 2002; Kim & Dear, 2013). Other layouts, such as shared offices for a team of people, performed better in many of the dimensions measured.

2.4 Software Quality and Complexity

Wolff & Johann (2015) explained that there are two types of quality in software, external and internal. External quality is that quality which is perceived by the customer, whereas internal quality is only seen by those that work with the code. It regards how much effort is needed to maintain and extend upon the software. "Complexity" was defined by Dvorak (2009) as how hard something is to understand or verify. He stated that the main consequence of complexity is risk by this definition. Further he highlighted the human role in the equation, as lack of understanding can be counteracted through education and training. Humphrey (2001) stated that technology can change quickly, however changing people takes longer. He argued that this is the reason that software development has had essentially the same problems for 40 years. Humphrey argued that these problems won't fix themselves due to the low changeability in people. Unless organisations change how they work with software, the current problems will likely get worse in the future as the trends suggest products will have more software and be more complex than the products of today.

2.4.1 Complexity Growth due to External Pressures

Godfrey & German (2014) said that it is impossible to anticipate all the complexity of the surrounding environment which will affect requirements of any software. They also mentioned that instantly when software launches it becomes part of, and changes, the real-world environment, which also creates new demand. This inevitably leads to evolution and change of the software, often in unexpected directions, as existing features get modified and new features are implemented. They further stated that a reason for this is the increased access for feedback from users (demanding new features), the application domain (legal and regulatory changes), the technical environment (if surrounding or dependent systems are changed in a way that impacts a features functionality) and by the system itself (when bugs are identified and fixed). The demand changes requested in these feedback loops are enabled by the evolution and spread of technical knowledge. As the user's demands and expectations of the software evolve, it will become progressively less satisfying to its users over time, unless it is improved and adapted to their new needs. Feedback must be considered and evaluated carefully as the demanded changes to evolve the system tend to increase its complexity and lower its quality, as perceived by the various stakeholders. Further, they stated the importance of adding additional effort in managing the growth of complexity to keep it under control, and elaborated on the necessity of having resources explicitly allocated for this for it to be achieved.

Lyu (1996) stated that the demand for hardware/software systems has increased more rapidly than the ability to design, implement, test, and maintain systems. The growth in utilization of software components is highly responsible for the increased complexity of many system designs. This reason was also noted by Reiner & Schaper (2010), as they said that managers are under intense pressure to bring new products to market quickly while also saving costs. This often results in development teams having to do quick fixes or "strokes of genius" rather than sustainable solutions. Dvorak (2009) further argued that instead of fixing an issue with a proper solution, taking the high one-time cost associated with it, when an "operational workaround" exists it gets patched in and continues to cost over time by increasing the complexity. This increased complexity in turn increases the risk of operational error, especially as the numbers of such workarounds accumulate.

2.4.2 Architectural Challenges in Software Development

Dvorak (2009) stated that good software architecture is the most important defence against incidental complexity. He stated however that good architectural skill is rare. Reiner & Schaper (2010) stated that establishing the right architecture is not a one-time effort, which is also stated by Godfrey & German (2014). It is an on-going process and it is important that it supports the product life cycle. Godfrey & German (2014) further highlighted the difficulties of maintaining a deployed software system and the popularized view that software systems are not maintained in a traditional mechanical sense of fixing worn out pieces, but rather that the essential function is to adapt and reshape the software to meet changing expectations. The paper further described the risk of the evolution of systems for the future, and the importance of being aware of it. The pressure for change is inevitable for any system, and if you can't predict what will happen at least you can prepare for the risks. To do so Godfrey & German (2014) emphasised the importance of developing systems that are amenable to change, which should be prioritised over striving to build a system that perfectly satisfies the requirement at deployment time. This importance is not reduced once the system is deployed, rather it becomes continuous work to manage and reduce complexity. Dvorak (2009) also mentioned that engineers and scientists often are unaware of the impact their local decisions have on the downstream complexity and cost. Overly stringent requirements and simplistic hardware interfaces can increase software complexity. Ill-conceived autonomy can complicate operations and a lack of consideration for testability can complicate verification efforts.

In the initiating phase for development of a large system, devising workable software architecture is the hardest and most important design task that must be undertaken. As the project proceeds internal boundaries within the system emerge, meanwhile the understanding of the problem space improves. Over time an inflexible design becomes a problem and will lead to a significant increase in effort due to needing to work around its flaws, this type of complexity is defined as "accidental complexity" of the system. Further a good design can greatly lessen the need for knowledge about other parts of the system required by developers (Godfrey & German, 2014). Card (2006) described the phenomenon of "Diseconomy of Scale" which has long been recognized in software development. This phenomenon means that larger software projects often have a lower productivity than smaller projects, all other factors being equal. He stated that projects of different sizes must take this effect into account as it won't be linear, meaning that the difference in productivity becomes larger as the range of software size increases.

2.4.3 Maintaining and Updating Software Systems

Evolution and maintaining costs for software systems are high; according to an informal industry poll referred by Erlikh (2000) 85 to 90 % of software budgets go to legacy system operation and maintenance. Taba, Khomh, Zou, Hassan & Nagappan (2013) also mentioned how costly bugs and maintenance are, they stated that identifying and fixing errors in software systems could be estimated to account for 80 % of the total system costs. The fact that maintenance is a large part of total life cycle costs for a software system is generally accepted; according to Chikofsky & Cross (1990) the numbers could be as high as 90 %. Even though reports show that at least 50 % of software efforts come from maintenance, empirical findings show that only 2 % of software engineering is focused on maintenance (Kemerer & Slaughter, 1999). Hunt, Turner & McRithchie (2008) illustrated the failure rate and maintenance needed in practice and in theory, see figure 2.1. The figure illustrates how errors occur and continuously are fixed. Erlikh (2000) mentioned another problem with maintenance and that is the difficulties of finding qualified personnel to do it, which becomes even harder over time.

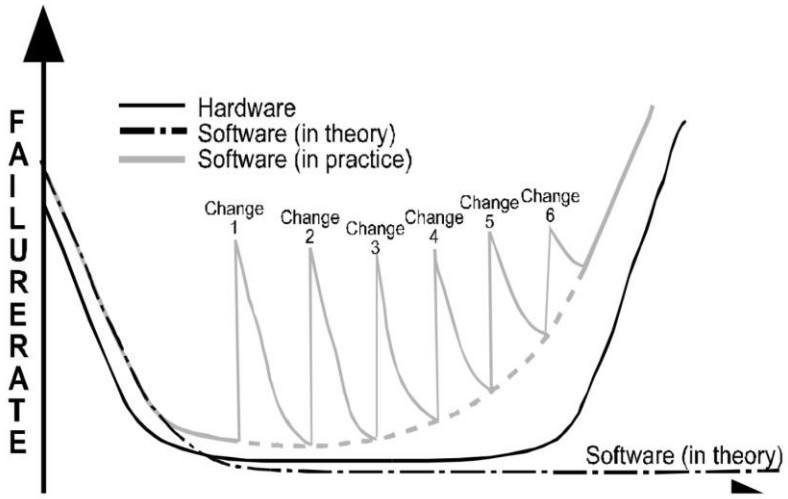

Figure 2.1. The figure illustrates a product life cycle's failure rate for hardware and software and includes a theoretical and practical view of the software's failure rate (Hunt, Turner & McRitchie, 2008).

Grubb & Takang (2003) described the importance of up-to-date system documentation as it is one of the major problems that programmers maintaining systems face. The lack of documentation decreases the maintenance productivity even if the programmers is maintaining code written by himself/herself which is rarely occurring as in the majority of cases programmers are maintaining programs written by others. Increasing code quality and good documentation will reduce the dependency and demand for qualified personnel. The estimated costs for maintenance are high enough to justify strong efforts from managers to monitor and control complexity. Tryggeseth (1997) stated that documentation is an important tool when maintainers must do changes in a system they are unfamiliar with and noted that without proper documentation the time needed to understand how to fulfil a modification request was 21.5 % longer. Tryggeseth further noted that there was a significant correlation of skill level and performance with documentation. However, without documentation no significant correlation between skill level and performance was measured. Engelbertink & Vogt (2014) mentioned that understanding the code takes around 50 % of the total costs of software maintenance. That gives a 12 % saving of total cost of maintenance with proper

documentation. They further pointed out that good and bad programmers do equally bad work without documentation, therefore spending money on the best people is a waste without proper documentation. Rostkowyz, Rajlich & Marcus (2004) found that the break-even point from investing in a complete reset of the documentation in a software system occurred after approximately 18 months. Further noted effects in that study were less effort needed as a percentage of total maintenance and lower costs for future documentation.

Grubb & Takang (2003) stated that "error-free" software is non-existent and some errors are difficult to detect even with the most powerful testing techniques and tools available. They further described and illustrated, as in figure 2.2, the increased costs of error fixing the later it occurs in the life cycle of the software. Hunt, Turner & McRitchie (2008) wrote that when it comes to maintenance, "a penny spent is a pound saved". They further explained that by increasing efforts in the development early phases will significantly reduce the maintenance effort needed and will reduce the overall life cycle cost. Further they stated that the more effort put into the development the less is required in maintenance.

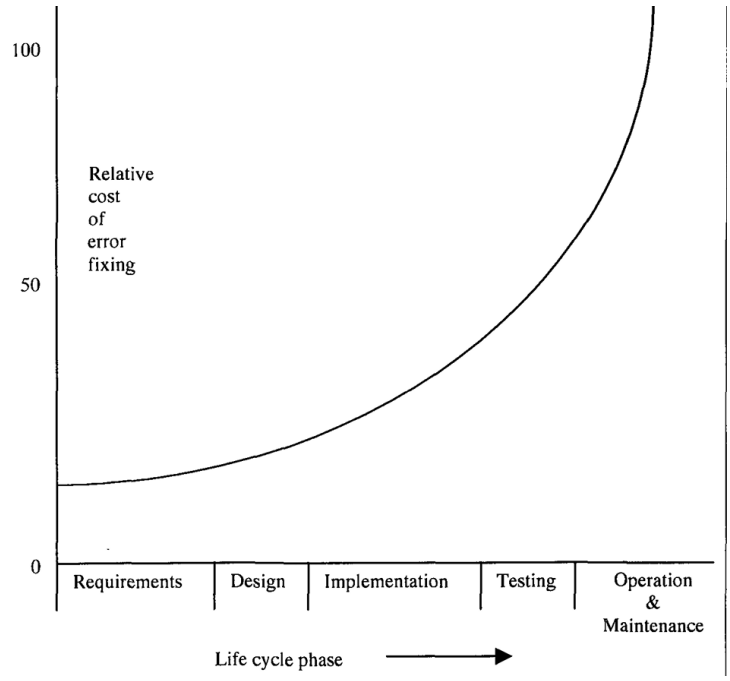

Figure 2.2. Cost of fixing errors increases in later phases of the life cycle (Grubb and Takang, 2003).

2.4.4 Internal Quality and Technical Debt

Wolff & Johann (2015) explained that a risk in software development is that too much focus is put on external quality at the cost of internal quality. In these cases, you build up technical debt (Schmid, 2013). Guo, Spínola & Seaman (2016) explained that technical debt means pushing costs of keeping code quality into the future, often to realise short-term benefits for the product. Examples of technical debt could be inadequate testing or documentation as well as architectural decisions that make the software faster to develop initially but harder to change. Expanding or changing codebases with large technical debt will be more difficult and

incur higher costs compared to doing the same in organised and well-designed codebases (Wolff & Johann, 2015).

Wolff & Johann (2015) described how taking on technical debt is often a strategic decision, in a way similar to taking on loans to finance investments. Working with the high debt code is then analogous to paying interest, as doing so will mean reducing the amount of work that can be carried out. This can also be looked at through the iron triangle (Maylor, 2010) mentioned in our introduction; by sacrificing quality the initial development time can be reduced.

Guo et al. (2016) explained that the process for identifying and evaluating technical debt was highly costly in their studied cases. Performing this process also required experience on the part of the developers doing it. The authors explained that the process is one of estimating work, something that is known to be difficult in software development, also for experienced developers but doubly so for those inexperienced. Despite this, it was shown that engaging in technical debt management gave enough benefits, in that it increased awareness of issues that otherwise were overlooked since solving them would not provide immediate customer value, to be considered a good cause of action for the case company.

3 Method

The following part will explain how we underwent this study, describing the methods used and why they were chosen. This will start with an overview of the research methods chosen, and then continue to describe the research process. This will in turn describe the literature study, data collection and data analysis.

3.1 Chosen Methodology and Justification

In order to address the purpose of this paper, a case study has been carried out, with a single unit of analysis, the department. Through the use of a case study, a researcher can examine data within a specific context closely (Zainal, 2007). The issues facing the department were unique within the company, and for obvious reasons it was not possible to study similar departments at competitor firms. Therefore, a single-case design was chosen, studying the phenomenon in its natural setting at the department.

Although numerical data has been used in the study, it has been in the form of budgets and financial reporting, and therefore statistical analysis of these is not relevant. All other data gathered is qualitative in nature, and thus not suitable for statistical analysis in this manner either (Pope, Ziebland & Mays, 2000). Analysis was instead carried out using an inductive approach as suggested by Pope et al. (2000), who recommended analysing data gradually during the course of a study, to be able to work iteratively and form a grounded theory base.

3.2 Reliability and Validity of the Study

The nature of a case study, especially one with a single-case design, is that it provides low generalisability, and thus reliability (Zainal, 2007). The results of this study were mainly meant to be applicable to the department, the unit of study, which alleviated this issue to some extent. It was our ambition, however, to provide an approach to identifying cost drivers that can be used in other organisations faced with similar challenges of time pressure and expansion. By following the research steps outlined in this chapter, we believe that others would be able to identify and explore cost drivers at organisations developing software, also outside of the case company. Further research will be needed in order to confirm that this is the case.

The problem with case studies, in regard to rigour, was carefully considered throughout the study. In accordance with Darke, Shanks & Broadbent (1998), we spent extra effort documenting all steps taken to explain the methodology and communicating the findings in a clear manner. Furthermore, work was conducted to ensure a firm theoretical base for all claims made, as can be seen in the earlier theoretical framework.

3.3 Literature Study

As has already been deliberated upon, the literature study started out as exploratory, using broad search terms such as: "*cost drivers*", "*software development*" and *productivity*. This

because of our limited knowledge in this area prior to starting the study. The search engines used were Google Scholar and that of the Chalmers Library. In addition, the publication databases from several Swedish universities were used early on in an attempt to locate thesis work already carried out on similar topics. Further into the study, more specific search terms were used in order to find theory about subjects chosen for analysis. Examples include: "*Organisational change*" AND "*Productivity*" and "*Agile development*" AND "*Complexity factors*". After the first round of interviews had been carried out, searches generally aimed at finding theory in the areas identified there and therefore became more and narrower in scope.

In addition to this academic search, other sources such as consultancy reports were used. This because the area of scaled software development is relatively new, and the availability of academic sources was somewhat lacking. These reports were found through the use of Google search, using search words such as "*IT Complexity*" and similar terms.

When we perceived a paper (or parts of one) as useful, its reference list was analysed for further relevant sources, if found these were then categorised, creating something resembling a database for the project. Articles that cited the paper were also identified and treated likewise. This process was sometimes repeated through multiple steps in order to acquire information about subjects relating to the original works. The intention of this process was to avoid using literature whose relevance could be low, due to either limited citations or incorrect use of their own sources.

3.4 Data Collection

Two forms of data were used throughout the study; secondary data gathered from the company database, including organisational charts and budget data and primary data in the form of interviews. The former was used to help us form a better understanding of the situation at the department and form the problem background, whereas the latter constituted the bulk of the data used for results and analysis. The following section will first describe the context of the case study and then move on to explain how the researchers went about collecting this data.

3.4.1 Case Description and Context

The study took place at a medium-sized department of a Swedish multinational corporation located in the Gothenburg region. Software development has traditionally not been the core focus of the organisation, but its importance has increased in recent years and at the time of the study there were several departments working with software. The study was carried out at one of these departments.

The department does not hire software engineers straight out of university, but only those with some prior experience. As we refer to inexperienced developers in this report, we mean those with little experience at the organisation, and therefore lower domain knowledge. As mentioned in the theoretical framework, experience working inside an organisational context is worth more than experience gathered elsewhere.

Another aspect worthy of consideration, which might affect the generalisability of the study, is that of the market for software engineers in Sweden. At the time of the study, there was

currently a lack of developers available on the market (TT, 2017). A recent study showed that companies expected this would lead to a higher amount of effort needed before new developers could achieve high levels of productivity then before (Andersson & Wernberg, 2018).

3.4.2 Secondary Data

In order for the study to be able to provide value to the department, it was important for the researchers to understand its current situation at the outset. Because of this, secondary data in the form of internal organisational documents were gathered. These included organisational charts, financial data and project estimates and results. In addition, an exploratory literature study was started in order to ensure the availability of academic sources for the considered topics. Any topics deemed potentially relevant were saved in order to be analysed further later on.

During this phase, the scope and aim of the study was continually discussed between the researchers and the supervisor, leading to multiple changes. Based on these discussions, as well as communication with other key people with experience and the exploratory literature study, a list of preliminary cost driver categories was developed. This list was used as a basis for forming questions for the first round of interviews.

3.4.3 Interviews

As the primary data source, a series of interviews was conducted with employees of the department. Interviewees covered all aspects of the department, from the head of the department to software developers. Initial identification of interviewees was made in collaboration with the project supervisor at the department, and further interviewees were found with the help of those interviewed during the course of the study.

Semi-structured interviews were used in order to ensure identified themes were covered, while leaving room for interviewees to elaborate on topics they found especially important as well as identify new ones previously missed by the researchers. All interviewees were invited via the company's appointment-booking system, and as the department is co-located, it was also simple to physically contact those that did not respond to meeting invites. According to Opdenakker (2006), face-to-face interviews have the advantage of providing the interviewer(s) with social cues such as intonation, which can hold extra information. This is especially true when interviewees' opinions on matters are of relevance, which has been the case in this study.

In total, 14 interviews were held, all of which were recorded as this provides more accuracy than writing out notes (Opdenakker, 2006). In order to minimise potential misunderstandings, all interviews were held in Swedish, the native language of both the researchers and all interviewees. This in accordance with Welch & Piekkari (2006), who described how interviews in foreign languages can provide lower quality data. The two reasons identified for this were native language producing more authentic answers and the use of corporate language (which is English at the company) leading to interviewees "providing "company speak" rather than "rich" responses" (Welch & Piekkari, 2006, p.12).

The interviews were held at the department's office in two rounds, during the period from early February to mid-April. As the objective of the interviews was to identify and explore cost drivers, during the first round of interviews broad and open-ended questions were asked initially, followed by more specific questions. For the second round of interviews more specific questions were developed, in order to explore gaps and provide more opinions on matters that had been discovered during the first round. These were developed by brainstorming after analysis of the first round of interviews and secondary data. In order to avoid certain interviewees having a larger impact on the findings, none of the same people were interviewed in the two rounds.

All recorded interviews were manually transcribed. Quotes were codified using a label system in order to simplify analysis. This work was conducted in the original language used in the interviews, in order to avoid translation differences between quotes. Quotes with similar labels were grouped together into categories and then further broken down into cost drivers, which were used as points of analysis. These are the first steps of the constant comparison method of Glaser and Strauss (Seaman, 1999). The results of this categorisation are presented in the Findings section. This process was repeated three times, after the first round of interviews, again after the literature study following the first round of interviews and after the second round of interviews.

Because of the fact that interviewees held different roles within the department, it was deemed likely that they would bring up mainly issues which concerned their respective roles. Therefore, during the course of codifying quotations records were kept over which interviewees mentioned issues in what categories. The results of this can be seen in table XX. As can be seen, a majority of the identified issues were brought up by a large share of interviewees, whereby all were deemed relevant to analyse.

3.5 Data Analysis

During the analysis the constant comparison method, a theory generation method, was used as described by Seaman (1999). This means generating theories, in the form of statements or propositions, about phenomena based on the data. It is an iterative process where statements are constructed based on early data, and then modified and expanded upon when other, related, data points are found. In this case this could be from different interviews or other parts of the same interview. This results in statements or propositions with a good description of a phenomenon (Seaman, 1999). In the presentation of our findings in chapter 4, our interpretations of the quotations presented are the final version of these statements.

After labelling and grouping quotes, as described in chapter 3.3.3, the next step in the constant comparison method was to look for underlying explanations of a phenomenon and write down a preliminary hypothesis to explain it (Seaman, 1999). This was done for all the different categories identified. After more interviews had been held and quotes from these also had been labelled, the statements made earlier were looked over again, and changed or confirmed depending on the new results. As such, the collection, codification and analysis of interview data formed an iterative process over the course of the study. The final result of this process is the analysis chapter of this report.

4 Findings - Identified Cost Drivers at the Department

In this section, findings from the data collection will be presented. Issues incurring costs have been categorised in two steps, as explained in section 3.3.3. The following section will present the four categories. Each cost driver will be presented separately by three columns in table 4.1. The first column covers the categorisation of cost drivers; the second column covers example quotes from the interviewees representable for the core content, the third column is the interpretations of the writers of this paper and explanations of the cost drivers and how it effects the department.

Table 4.1: Presentation of identified cost drivers.

Category	**Example Quotes**	**Interpretation**
Organisation		
Work environment	"I don't think it's good. Of course, you get disturbed by all the noise and coming-and-going." "I think working in a landscape is okay, but that people's sound discipline is too bad. It is all about a mind-set, that people should not have loud conversations when working in a landscape."	Working in an open landscape has received multiple complaints, claiming it is harmful for individual focus and productivity. A lot of noise from surrounding people conversing as well as visual noise from movement interrupts and forces people to refocus. Some blame this on the landscape itself, whereas others blame individuals with a lack of discipline. Several people have limited their view field and started using noise-cancelling headphones to try and improve productivity, whereas others have started working from home some of the time.
Boundaries between functions	"We often feel forced to use certain concepts by [other department], which we argue don't work for us, and then we have to fight to get changes made." "It would be possible to cooperate in a completely different way. Perhaps it's a lot due to us as well, when we get a project and feel that achieving a change towards [other department] would be hard, we build it in a different way to get around the problem, instead of pursuing the change which would give the best solution. Building around it takes less work than the discussion." "When buying hardware, they do not account for what it shall be able to do five years later, rather they say "it needs this performance, this memory", and then five years later we sit and wonder how we can ever get new functions implemented with those resources. … The more software departments we become, the more we get an acceptance to have hardware that is prepared to allow growth."	Several interviewees expressed that the department have issues due to poor communication and fit with other departments at the company, including purchasing and departments developing other software and hardware products which need to align with those from the department. Extra work is often carried out in order to create a better alignment, building around the deficiencies, as this requires less work than convincing other departments that the chosen solution is the right one and waiting for long lead times for change in overarching structures. Communication between departments is described as lacking, and an assumption made was that departments rate their solutions as the best even when they are not in order to win power internally in the organisation. The supplied hardware is often just enough to run the software, which makes it difficult to improve upon it later on. The communication and synchronisation between different parts of the department was also mentioned in multiple interviews. Developers experience that a disconnect between the development teams and other parts, including design, management and later stages of testing. Issues include receiving designs that are "impossible" to implement in the environment, not realising all dependencies between teams so that

Boundaries between functions (continuation)	”It works decently, we have little communication outside, the team is very isolated. It’s good that we can be isolated and work undisturbed, but there are risks that we miss things, for instance that we do not understand the entire functionality, or discover dependencies or use cases late which forces us to go back and change the code, which lead to delays”	integration becomes difficult and problems based on the old organisational structure, the mind set of working iteratively with all aspects have not yet taken root in some parts. Teams are sometimes taken aback by changes they believe have been decided upon, but later find out that others have not been informed about, which lead to issues in integration.
Criticism and stress	“There is a large time plan for us to deliver a product, and we’re behind. Management and people with responsibilities become more and more stressed. It’s not meant to rub off on us, but of course it does. And obviously it’s no fun when our estimates are questioned. When someone from higher up comes and asks, “how can this take so long?” we wonder what they mean, are we doing a bad job? … It’s probably pretty unique in world history to underestimate needed effort in a project.” “We’re not a software company, which is evident. We talk about being one but when it comes down to it, its nuts and bolts and engines, that’s good stuff …, when you estimate projects you hear “how can it cost so much? It costs as much as changing an axle! It’s only software, how hard can it be?”	Interviewees explained that the department is currently under a lot of stress, coming both from inside and outside. Critique about projects taking too long and incorrect estimates have been impacting the performance of the department. Several developers mentioned that management inside the department had voiced complaints about the teams setting too low estimations for their coming work, even though they often failed to meet the previously set estimations. This was mentioned as a source of stress by several interviewees, some of whom blamed management for “not knowing what it is like” as they do not have a programming background. The new platform project was mentioned as a source for lower than average accuracy in estimation, as it is difficult to anticipate where issues will arise when multiple components are being developed at the same time. Non-developers, including interviewees in the management team, mentioned that they have received similar complaints from outside the department. Estimates have traditionally been inaccurate and the department’s deliveries are often not predictable according to the way the organisation runs projects. Complaints about the performance of the department have also been recorded, which some interviewees believe stem from a lack of understanding for software development in the wider organisation, which is traditionally very mechanical. Especially regarding the increase in manpower in recent years there have been many complaints, stating that the functionality being delivered is very similar to the old one, which was delivered faster with less manpower.
Organisational changes	“It’s gotten bigger, but at the same time smaller. We used to work all in one with 30-40 people, now we are teams of 7 instead.” “The daily work is easier now as we have our feature within the team, but when there are problems and you need to get a wider perspective there are many others, unknown people, who make it difficult when trying to integrate different parts and some things don’t work because someone changed something you depended on for instance. You notice this before we run system demos; it’s chaos with everyone running around trying to solve things in the last minute.” “We have people who are all-in on these changes, but they don’t agree on where they should lead to.” “I’m still sceptical about some things; it takes long from discovering you need to add something new until you can start to do it since we’ve planned a fixed scope for a rather long time forward, so it’s not fully agile. But after some adjustment time it’s	The new organisational structure and size have both been mentioned by interviewees as potential issues for productivity. The expansion is perceived as having driven a need for more control, resulting in more meetings and other activities aimed at ensuring work is kept cohesive and running in one direction. Despite this, interviewees mentioned issues with integrating contributions from different teams, as they have become more independent and therefore sometimes have solved their problems without seeing the bigger picture. This can also lead to some issues being overlooked, as the division of labour may sometimes miss things in the middle. Many issues mentioned were attributed by the interviewees as being due to the organisational transition, both the rapid growth and the structural changes. A lack of knowledge by people in roles that are new to the organisation such as the ones breaking down the work mentioned earlier was a recurring factor mentioned, as was not everyone having the same idea of what the end goal of the organisational changes is. Misses in communication between different functions, which ended up causing rework of functionality, was also attributed to these changes.

Organisational changes (continuation)	gotten better …, there's still too much time put on planning, all of it doesn't feel meaningful to us."	Opinions of the structural changes have been mixed, interviewees that mentioned issues related to them often followed these up by stating that they liked the changes in theory, and that things have started to get better.
Investigation and planning	"These two worlds [agile development and the traditional stage-gate process] still doesn't go together …, especially when we are under very large pressure as we are today." "We can spend six months before starting to code to do these pre-studies, and it ends up in one document. The question is if it is worth it?" "You could probably do something in-between [the technical report and the change request] there to speed it up" "Burn downs are useless. We have no good way to measure, no KPIs. We spend a disproportionate amount of time on them compared to what we get out of doing them."	The process changes in the department and the way they differ from the standard project process at the organisation are factors that potentially have caused issues, according to interviewees. Specifically, issues concerning the planning phase were brought up. In order for a project to be given a go ahead, a technical report needs to be created, which debates different solution alternatives and recommends a course of action. This is then translated into a change request. According to several persons this document requires a lot of work in the early phases, and its relevance once a project is under way was questioned. Opinions on the work itself are mixed, but the way the report is structured received mainly complaints. One complaint was that the report quickly becomes obsolete, and is not used in projects after initiation. Another common complaint was that the value gained from the report does not equal the time spent making it. This complaint was also made regarding estimation work within teams; that estimates end up giving very little in terms of making the work easier but still there is a lot of time put into making them.
Shortages of resources	"Our biggest problem is where to sit …, there are never enough desks. That process [at the company] is so slow that once we get enough desks we have already grown past those as well." "Hardware deliveries, and knowledge about them; it's been improved but for a long time we never knew when we would get the things we needed to do certain tasks …, you get less efficient when you need to dig around the backlog for other things to do or build workarounds."	Multiple issues with delivery of hardware components needed at the department were mentioned by interviewees. This ranged from hardware needed to run tests all the way to desks for new employees when department growth was at its most rapid pace, and internal order processes were to slow to keep up. Results from this have included spending time creating workarounds in order to be able to proceed without necessary hardware, or even in some cases preventing people from accomplishing work all together. The main type of these issues has been delays, complaints centre around not having things delivered by the time they are needed. Additionally, uncertainties about when hardware would be delivered, not just that it was delayed, were a problem.
Staff		
Product competence	"In general, we get a bunch of problems because they don't understand the functionality we are supposed to build, it's very specific. They simply don't get the functionality; they shall create what we call "Friday functionality", which is basically recreating [the old platform], but with new tools and different code, so it's a bit difficult to go back and look how it's done. And they don't understand why it's done the way it is." "It's been decided that this old stuff is bad and it needs to be redone. But when you say that it feels like if it's not the same people doing it again you miss things and it may become as bad	A lack of product knowledge was mentioned as an important factor causing work to take time; both in terms of understanding the context where the software will run and understanding the old software that will be replaced. Some said that parts of the old software could be easily transferred to the new platform, as the solutions used were good, but that requires the developers to understand both how the functionality is intended to work and how it actually works in the old software. When they do not understand these things, they instead create their own solutions from scratch, which may result in repeating old mistakes. Understanding the old product becomes increasingly difficult in parts that have been changed multiple times, as is discussed in other findings. This same reason was also said to affect others, not only developers, including project managers and architects.

Product competence (continuation)	again, but with different bugs and issues. But we can hope that [the new platform] gets a little bit better at least. But that's not guaranteed …, it's a mixed bag where parts of the old platform are things that have been added to and patched and it's evident that it's not doing the same thing anymore as it was meant to from the beginning. So there is a risk that you create the same functionality as the old one, but it doesn't end up being quite that, because you miss something that has been added." "But then you can ask an expert about how a function actually works and they can answer that it's never been used. And by not implementing that you can save thousands of hours. It's not common, but that knowledge just isn't there, one cannot learn that from reading. First you have to even think about checking whether it's been used, then you have to know who to talk to, it's so large here now that you have to have contact routes and stuff like that."	One interviewee explained that he had tried to talk to someone with high knowledge about a certain function, asking how it had been used in the old product, and the response had been that it never had been used, despite being in the code. Finding this out had saved considerable time because the team could then skip creating this function in the new product. He described it as a stroke of luck that he even thought about checking, and said that many probably would have spent the time to try and understand and implement the function, as the knowledge of how functions are used in the old product is not commonplace. In the new organisational structure, the teams are themselves responsible for the final breakdown of demands and functionality, after being handed high-level functional demands for their tasks. This means that there are decision-making at the team level, which in itself sets higher demands for knowledge from members. Several interviewees mentioned that this has caused difficulties when the high-level demands have been misinterpreted and the end product has not been what was needed. One opinion expressed was that the structuring of the new platform have reduced the need for expertise in developers, as functions are more independent than was the case in the old product. The more interdependent the code is, the higher the demand for understanding of the entire product becomes, according to interviewees. This should allow higher productivity for developers, some believe.
Employee turnover	"I think that's an issue the line managers have, that people are coming up and saying: "I am forced to work with this now, but I don't like it and I don't want to work with it. Solve my problem." I think the managers hear that, and they have to deal with it in some way. I think the risk has gone up of people quitting because they don't get to code anymore. Especially with so many consultants, they can say "this is not what I want to do" and quit. Then you have invested a lot in someone who just leaves." "There has been a lot of movement and breaking up of the teams. That is catastrophic for efficiency and you ruin the team spirit. Every time we change the team culture has to be built from scratch and that takes time and creates friction. That's an issue, even though it's hard to quantify because you cannot measure it. But there is a lot of friction early on [with a new team]." "Of course there is some turnover, hard to say but some turnover is natural. We have a lot of consultants here as well, and they are very different; some want new challenges all the time, so it doesn't have to be stress that makes them leave. Some are self-employed and some leave because [the company] doesn't pay enough. Suppose I go to the company next door, I can raise my salary quite a lot."	Employee turnover has been described as an issue on two levels, the team level and for the department as a whole. On the departmental level, interviewees mentioned that due to the high rate of inexperienced developers, those who have experience spend a lot of their time teaching new employees instead of coding. Several interviewees saw this as a risk for the future, that experienced developers may decide to leave the department as their job has changed, and they would prefer to do more coding than they are allowed to when they need to take part in teaching new developers. Another factor in a potentially increased rate of turnover is described in that other companies offer higher wages than those that developers currently get at the department. Several interviewees mentioned having received offers from other firms with higher salaries than their current ones. As for the team level, all developers interviewed mentioned that the teams they worked on had changed their members many times during recent times, and that it had had a disturbing effect on their productivity. It was explained that every time the team composition changes, teams have to start over in terms of team culture, estimation and other factors which over time increase productivity. The developers themselves say that they believe their teams have started to become more productive and better at estimating lately as there have been less turnover. The trade-off between introducing new members into existing team, lowering their productivity but hopefully helping the new people become productive faster, or starting brand new teams but leaving the productive ones alone was discussed by interviewees in management roles.

Lack of experienced workers	"There is a lack of software knowledge among system leaders; it is hard to find them". "Experience in general is the difference. It's not just that we've expanded, but we program in a completely different way" … "which means that there is a lot of work in learning how to do things in the new systems"	Lack of experience was described as an issue in two ways; as it related to finding new employees when hiring and as it related to new technical knowledge that has become necessary due to changes in the environment, such as programs and code languages used in the new platform project. Several interviewees explained that it is difficult to find new hires, which have resulted in varying levels of skill on those who are hired. This applies both to developers and other roles. Even the most skilled of the experienced developers are said to have a reduced level of productivity while becoming used to working in new ways and with new systems, which some think has caused progress on the new project to slow.
Consultants	"People think that being consultants are a nice deal. But there's another thing for me and that's that I have my consultancy and there I can be involved with other types of projects. I get to meet other people that I can talk to and there is this other world, you have a gateway open to what's happening in other places all the time. But if you are employed here at [the company], that doesn't exist. That's how I feel at least. Then there are also those who are self-employed, who just want to be their own boss. And [the company] only hire them via some of the big companies, who sit in between and take money, that's all they do." "I don't think there's a noticeable difference between consultants and employees, it's all very similar. …, I'm not employed here but it's almost the same except that I report my time in another system and have some activities with my consultancy on the side. So those activities I don't have with [the company]. And of course, those can help meld people together, but we have team activities where it doesn't matter at all if you are a [company] employee or not."	There is a large number of consultants currently working at the department. Attached to this is a higher hourly rate whereby the need for consultants can be seen as a cost driver. Interviewees have varied explanations for why the number of consultants is so high. One explanation told is that there used to be a maximum number for the headcount of the department, so when more people was needed the only possible way was to hire consultants. Despite this no longer being there, many consultants remain employed that way. Developers who themselves are consultants have in many cases said that they are not interested in becoming employees directly at the department, for various reasons. Having the possibility to quickly go to a different company should tasks change and work become less fun was one, as was having the opportunity to be involved in multiple projects and share knowledge with others from the consultancy firm. Several of those who expressed these feelings also said they do not really use them; many consultants at the department have been there exclusively for multiple years. There are also those who are self-employed consultants due to wanting to be their own boss, these are thus hired via an agency which adds a margin, increasing costs for the organisation. No interviewees said that they experience co-workers differently based on them being consultants or not, and some of the consultant interviewed said that they feel like they are employees of the department in all but name. Some speculated that there perhaps could be beneficial to ensure that consultants are not quite at the level of full time employees in this regard, to create reasons for consultants to become employees instead.
Learning time for new employees	"When people are new it takes quite a long time for them to get up to speed. And of course, it takes time from those who have experience, because the system is so large and complex that new people, regardless of how good they are, need help getting started. So initially capacity doesn't increase when we get more people, there is a start-up stretch before you get productive and there's a	The interviewees stated there has been a fast pace in recruitment of new employees at the department to cope with the increased demand from the new platform's development. The interviewees however implied that you don't get a higher delivery right away with new employees as they have a period of learning before they become efficient. The reason for this is that they need to understand the big and complex structure of the code which takes time, even if the recruits are highly skilled

Learning time for new employees (continuation)	benefit from being more people." "At the same time, bringing in someone new and having them sit by themselves is not a better option, they have to get that time. But you have to realise it is costly to grow and that you don't get the productivity boost right away, it's to do with the growth that we need new people, and that lowers productivity temporarily." "When the bosses say: "here are four new people, you'll have to use each other and get up to speed as fast you can." That's going to be slower. Say you have a team of seven that works and you put in an eight, he's going to work pretty fast. Then [the team] can still deliver and produce things, but if you have a team of four and put in four new people they are not going to deliver anything for four sprints or more." "The thought of [the new organisation] is that the teams get quite a lot of responsibility, but that doesn't work if they don't have the required knowledge to take responsibility."	programmers. The interviewees stated that the new recruits have been on an average/normal level when they start at the organisation. In the beginning the new employees ask the senior programmers lots of questions to understand the code and be able to work. These constant interruptions decrease the productivity of the experienced programmers. The senior programmers interviewed claimed their output almost goes to zero because of the ramp-up in the organisation. This was described as frustrating, as the programmers are hired to code and enjoy doing so, not just teaching new employees. The interviewees also highlighted a possible risk with this in that experienced workers could grow tired of their new duties and quit because they want to spend more time coding. The interviewees are aware that it is very hard to scale-up in a good way as new recruits need to learn somehow and some said the chosen strategy may be beneficial in the long term. The high pace of recruiting combined with the managerial strategy of splitting well-functioning teams, mixing them with new recruits, was mentioned by interviewees as reducing productivity in the short-term from one team that produces to almost nothing. From an experience perspective it was said that it would be more beneficial to add just one or two new persons to the well-functioning team as that doesn't affect the performance of the experienced developers too much. One of the interviewed managers experienced difficulties in the shift to the new organisational structure due to the number of new recruits, as the new structure are moving more responsibility and autonomy to the development teams. This is problematic as the teams, because of their high shares of new recruits; have lower product knowledge than earlier, which makes it hard for them to take on this responsibility.
Code Complexity		
Documentation	"In [the old product] it is rather useless, in essence it is non-existent. It's very bad. [The code] has had a long life and is developed by a lot of people, so it's different in places, but in general [the code] is poorly commented and documented. That makes it difficult to understand the code in places where it is highly complex." "I don't think we even look at the old code much now; we try to understand how functions work, which is not easy, but then we build it in a new way." "We're better [at documenting] now, for instance we write design documents that are scrutinised and hopefully maintained as well. It's too early to say, for now they are up to date but what happens with time we will se. Of course, there is a risk that we write the documents now that we develop the first version but fail to update them when things are changed. That's a very common problem and I don't think we're protected from that."	The areas of documentation commented by the interviewees is primarily separated design/function documents and comments integrated in the code. The design/function documents are supposed to describe the feature of the code and be kept updated and "alive". The comments are used for enhancing the readability of the code. The quality of these two documents is very poor in general for the software produced in the past. The level varies however greatly in different areas of the software. The interviewees are implying that the reason for this is who has written the code and is highly affected by the individual developer and how much effort he/she has given to the documentation. The interviewees further explain that the negative impacts of the lack of proper documentation varies depending on how complex the code is, if it is well written and simple it doesn't affect too much. However, the interviewees imply that majority of the code and functionality is rather complex and hard for the programmers to understand. For the new platform developed they now have a new documentation philosophy. Updating the documentations when working with the code is integrated in the process

Documentation (continuation)		and is mandatory for reaching the definition of "done" of the project. The interviewees are positive for the change however highlights risk of falling through again over time as the documentation has always been part of the work for the developers but obviously has failed in the past and how easily it could happen again if the focus on documenting fades consciously or unconsciously.
Increased number of products and variants	"It used to be a lot faster; one person could fix 5-10 bugs a day and implement the solutions, now we maybe fix one a week. The complexity of the product makes it hard to understand how to implement a change so that it works for every variant, and together with the testing that costs a lot." "What affects this is that that particular line of code is included in seventeen places, and sometimes it's included in a different way depending on the variant, so it's not enough to understand the thing you are changing, but you have to understand the entire product. This is due to the monolith mind set we had in the old days, which we are trying to get rid of now, making it a bit smaller and more independent, a divided product. We haven't gotten to the stage where we integrate the modules yet, however, so we cannot know how it will end up. Hopefully you will be able to make changes in a module without having to test anything but the things you change."	10-15 years ago, the department was developing one product for one automotive type. Today they have around 15-20 different products and variants of the product which creates different dependencies and unique difficulties. The products and variants are built of the same core of features and could be specified as different configuration specifications of the features. The code is built as a monolith which is explained as a solid code core which all features is integrated into. The dependencies make it harder to maintain and increases the complexity. The interviewee is also stating that the testability decreases as it is complicated to cover all the affected areas a change could have and it is takes long time to fix bugs and errors. Previously when there were fewer products and variants a programmer was able to solve around 5-10 bugs per day, now however they are maybe able to solve one bug per week. The interviewees imply that the overall impression is that the offered products and variants has increased too much and are mentions the possibility that the customer value doesn't exceed the cost for the increased complexity. For the next generation of the platform the focus is to develop a more modular design. The modularity is expected to decrease the dependencies in the code which will increase the changeability.
Client created complexity	"They want to be different, the brands, they want it in their own way …, often the wish is "clearly different from [the primary brand]", even if it's the same platform they do not want that to be noticeable. It needs to look like [other brand], so that there's a feeling of uniqueness." "We have to do the extra work. One market wanted a round map and another wanted a square one, no reason at all for there to be two different ones. But we failed to get them to agree so we had two build both. That increases complexity even more and you have to test both. And it draws more memory and other resources."	The interviewees describe the strong demand from clients (internal and external) for change in the software. The strong demands for changes in certain products and variants could force the programmers to lower the quality of the code and could fragment the product offerings even more. One reason for these demands is explained as different branding through the brand portfolio and differentiating strategy. The functionality is often the same it should just look different.
Coding legacy	"But right now we take the hit and do the job, we are trying to make the code simple for the future. A piece of code that you've worked on for 20 years, with different people, takes on a life of its own." "In some places we have gotten weird dependencies in the software, so in some cases things are included even though it isn't used in that variant …, even the core is different, there is no common things for all variants."	The interviewees states that the code is rather complex and one interviewee describe the code as a living organism as of its growing in size and interweaving from maintenance over the years. The code has been patch together and "tweaked" to fix the bugs that have occurred over the years which have made the product work the way it does. However, over the years everything has been intertwined together and for some products they include too much code that they aren't using because they are unable to solve the dependencies. The interviewees' states also that existing code is rather complex just by itself even without the dependencies because of its high-level coding. They are stating that the competence of the programmers in the past has been very high

Coding legacy (continuation)	”You are supposed to sit down and judge whether the old code is good or bad and get some inspiration from it. But the old code is so complex that the investment of sitting down and trying to understand it, many don’t think that is worth the effort. It’s extremely difficult to get to the point where you understand [the old product].”	and they have been able to solve problems with really high-end solutions. They state that the solutions could be superior to other solutions by performance but extremely hard to maintain for another programmer and then the solution could be inferior anyway compared to a simpler solution that are easier to understand. They are also stating that they aren’t looking so much of the code developed in the past because it takes to much effort to understand it. This is highlighted a risk when they now are developing the new platform as there are risk that they miss dependencies and solutions fixed in the previous platform and they will face the same bugs as they did with the previous platform and have to solve them once again.
Stress induced complexity	” Naturally, when working under time pressure you only do what you absolutely have to, and as soon as something is done; you deliver. Then perhaps you leave it in a state where you’re not entirely pleased, your code may not be perfect or you haven’t tested it enough et cetera, you’re not quite happy with your design or documentation. But you’re in a hurry so you have to leave it there and move on. And then we start building technical debt in the code, which is not good, it’s short sighted and we keep mobbing forward but in the long term it gets worse and more expensive. So, depending on the time perspective it is very different, but it’s never good.” “I think that many teams out there feel a lot of pressure to deliver now. Every week that passes it tightens. And I think that up to now we have been very careful to build good things, but that we’re reaching a point where we start making compromises. I already know …, if the delivery looks like this, then when the team delivers I see that they haven’t done these four things. “No, we removed them” but they haven’t told me. So, I can walk around and believe that our platform has something that it doesn’t have. Because I’ve ordered the work and at the sprint demo they sat and said that everything is going fine, but in the end, it isn’t there after all. Things like that build a frustration among those of us who are outside the teams.”	The interviewees described the pressure they are under to deliver quickly what is expected of them. Beside the stress the programmers experience they also explained some negative effects the time pressure has had on them. They said the pressure has made them fill sprints with more work than they believe they could handle, hoping that everything will go smooth and without problems, which they stated rarely occurs. This has made it harder to plan as they have spill-over between sprints, meaning tasks stay into the next sprint, which could otherwise have been filled with other tasks. Until now they have been very focused on the quality of the work and doing everything correctly, recently however they have sensed that they are compromising quality to increase speed. The main concern of that is that technical debt in the code increases when there isn’t enough time to build the code in a proper way. For example: bad design, not enough testing and insufficient documentation and commenting. The interviewees also stated that they are pressured to deliver faster than they are able to while producing proper work. They deliver just what’s being ordered and there is a risk of missing out on solutions because no one is taking on responsibility for the whole product and tasks between different subtasks are at risk of being missed or ignored. One interviewed software architect also stated that he is starting to notice some programmers are skipping functions he has ordered and still report that they have tested everything and it worked correctly. But then later on he noticed the ordered function wasn’t there and the explanation he got when he asked why it wasn’t there was that they didn’t think it was important. The software architect further stated that he believes the reason for this was that they didn’t understand the full domain, that it would be important for some other features, whose functionality will be developed later on.
Code Quality		
Quality assurance	”The code is scrutinised a lot more, all code is checked which it wasn’t before. This makes it so you cannot get away with writing sloppy or unclear code. The team itself goes over all code.” “When you push a commit you also make a pull request and then you can see difference in what has been done straight away. For example, this developer wants to implement this code, then	Quality assurance, specifically the level of which should be held, was referenced as a potential issue in several interviews. Quality demands have increased drastically in recent years, in an attempt to never release untested code, which used to happen. Produced code is scrutinised in a completely different way in order to ensure that it works as intended and that it will be easier to change the code later. With the old product, testing worked such that whenever updates were finished, manual tests were written, spread out over different functions and product variants to reach as

Quality assurance (continuation)	everyone can see what she has done and add comments straight to the code for everyone to see and comment. So you can make comments on the comments and turn it into a group discussion and in that way spread knowledge. It takes some time, but it spreads knowledge and increases quality." "The project is still running, it is the project that's responsible for the high-level testing. So if bugs are detected and comes back, this delays the next project." "A lack of knowledge generates bugs, and bugs cost. The closer to the developers we find them the better. We also try to spread the test so that we test something everywhere, when we cannot test everything." "Regression is difficult. Every time we find a bug the regression increases. Because of a lack in regression, we had to recall 30,000 [products]."	high coverage of the code as possible. Tests were run outside of the development teams, which caused issues when a team had already started working on another project by the time bugs were identified, and then had to reprioritise their backlog mid-sprint. This was described as a large source of extra work, causing the following projects to be delayed. Several interviewees expressed that they believe it is better to discover bugs as close to the source as possible, that they incur fewer costs then. With the new testing philosophy (see its separate findings section), tests are written in such a way that they can be run automatically, and easily repeated when the code is changed. With the new focus on testing, a larger portion of testing work has been moved into the teams' development work in an attempt to identify and be able to fix a large portion of bugs quickly. Some interviewees expressed concern that too much work is taken up by testing, and that it perhaps would be better to "fail fast", implementing code that maybe had more bugs, but could be done much quicker. This was explained as a trade-off where the decision has been taken to have a high standard on quality, at the cost of flexibility and speed.
Test strategy	"We have a different testing philosophy now, with a wider base of unit tests." "So we have a lot of unit tests and component tests and so on, which we didn't have in the old system. And writing unit tests drives a certain type of development, you have to design your classes and components in such a way that they are possible to test. I think that's the biggest piece, which takes a lot of time, especially now when people aren't used to them and make errors and have to redo it until it's good. I believe that …, I know that we put as much time on tests as the actual code now."	Previously, the department ran limited unit tests (a test on the smallest testable part of the software), but instead had more V3- and E2E tests, whereas they now try to do the opposite and focus more on unit tests. Generally, the interviewees thought this is a good strategy and highlight the benefits of finding errors early in the process. The interviewees however explained the difficulty of finding the right balance between building a test environment and producing functionality as they now spend around half their time on the test environment and half on developing the platform's functionality. Integrating the unit tests into the development phase requires a different way of coding, which even the experienced programmers at the department are unfamiliar with and needs to learn. The expected outcome of spending more time during the early phases by writing unit tests is that more errors will be found earlier in the development process. The code will be more expensive to develop, but the goal is to save time and money in total. Interviewees expected this will increase both efficiency and quality in the long term.

5 Analysis of the Findings

The following section will present an analysis of the identified cost drivers at the department, and their relation to each other. The first section discusses factors that impact the productivity of employees, the second analyses a potential increase in turnover and what effects it would have on the future and the third covers issues related to complexity and quality in the software products themselves.

5.1 Factors Impacting Productivity at the Department

In addition to employee turnover, several factors have been identified to be, or potentially be, harmful to productivity. These have been divided according to the way they drive costs for the department and how they affect each other. As employee turnover has been deemed to be the most crucial factor and also risks having an even higher impact in the future it will be treated separately in section 5.2.

5.1.1 Coordination between Functions

Coordination and communication between functions, both inside the department and with other parts of the organisation, were seen as less than ideal by interviewees. Cooperation with other departments was described and interpreted as poor, for example it is not uncommon for the department to conduct extra work to avoid having to coordinate as much with other departments. This is not at all in line with literature, which emphasised the importance of having a common view of agile and ensuring that non-agile functions align to and do not hinder the performance of agile functions. The sheer size of the organisation makes this difficult, as does the way projects are planned centrally. The issues seen in the planning phase of projects are another example of how this alignment is lacking, in that particular case the processes required organisation-wide are seen as directly obstructing the agile process of the department.

Internally at the department; some issues in coordination, such as that of receiving impossible designs, stem from a lack of product competency, but others imply other problems. The literature stated that adapting the agile approach to the situation where it is implemented is important, results were better when an approach was customised, also at the team level. Finding the right level of customisation was deemed important however, to not have too large differences within the implementation. The fact that different teams and functions at the department are having difficulties synchronising work could imply that there are perhaps too great a level of customisation at the team level. It is also possible that too much focus has been put on the specific implementation of the new organisational structure, and not enough on the underlying principles of agile. The literature stated that the emphasis must lie on these principles to achieve the best results.

Difficulties in coordination may also stem from low predictability. In order to plan the larger project, it is important that the teams deliver what they set out to in each sprint, and issues were identified in this area. Interviewees said that estimation work has become more and more optimistic and sprints often have some leftover work from the previous sprint when they

start. That estimation is an issue at the department can be seen as a fact; the difficulties of estimating were mentioned by several interviewees. It was also mentioned that whilst it is still perceived as highly difficult, the quality of estimations increases as teams work together over time. We argue this is another reason to attempt to limit the team member turnover, which has been very high during the recent period. Establishing a plan for how and in what circumstances team members are changed is imperative and together with, again, keeping experienced members on-board can be seen as the most important factor for the department moving forward.

5.1.2 Changes in the Organisational Structure at the Department

Another factor which has had an effect on the productivity of developers at the department is the new organisational structure. The changes introduced have evidently had an impact on the productivity of the department.

Interviewees stated that they have spent more time on administrative tasks such as meetings after the changes were introduced and that the new structure puts higher demands on individual employees and teams. This combined with the fact that new roles have been created in which those employed either have little or no experience if they are hired internally, or lack the necessary product competence if hired externally, further increases the need for coordinating meetings and education of employees. This implementation goes against the recommendations from literature, which stated the importance effort should be put on customising the agile approach so that it fits the current organisation in a good way. As in the previous section, we deem the lack of an overarching agile vision for the entire organisation to be a critical flaw, which have certainly furthered some issues.

Despite these criticisms, it is important to note that many interviewees also saw positive effects from the organisational changes. Especially lately some of the issues have lessened, as people start to get accustomed to their new roles. Whether or not the changes will have an overall positive or negative effect on the platform project remains to be seen, but for other organisations aiming to conduct large changes to the way they work, we recommend doing so before embarking on large projects with a high delivery pressure, not during such projects.

5.2 The Risk of Increased Personnel Turnover at the Department

In the findings section it was explained that several persons at the department see an increased risk of people quitting in the future. This stems from not being happy with the current situation and is something that the department has to deal with quickly, as keeping hold of experienced employees can be considered a key factor for success in software development. Several factors that could potentially contribute to discontent have been identified and will be analysed in the following section. After that, a discussion will be held on the potential costs associated with an increased turnover.

5.2.1 Reasons that Personnel Turnover Might Increase

Noise and movement in the landscape environment received many complaints, which are in concert with literature; several sources claimed that an open office environment has negative effects on employee satisfaction. Among other issues, workers in open offices perceive their performance as lower due to disturbances. For the department; it may be worth evaluating a different work environment such as team offices or co-locating individuals whose work involves a lot of conversations. The department already have some break out rooms, which is one recommended course of action in literature, but providing more, in particular with a larger size (most rooms today do not fit entire teams) could potentially provide benefits.

Stress is a factor that drives employee turnover in all industries and was seen a lot at the department. Critique, stemming from both inside and outside of the department was recorded, which in many ways match the traditional critiques directed towards agile implementation. There we also explained the issues people conflicts and change resistance and elaborated that the best way to ensure support for implementation was to educate management in agile development. It is our belief that this has not been done to a large enough extent at the company, a belief that was also shared by several interviewees. Knowledge about how needed work increases exponentially with increased complexity and external factors need to be taught, and strategic alignment between departments must be reached both between all those working with agile methods and with those not working with agile. Other factors that indicate a less than optimal relation between departments are the lack of required resources, in particular necessary hardware, as well as a lack of forward-thinking in terms of hardware choices.

As for consultants, their higher mobility, in terms of being able to leave the department with short notice, could potentially have negative consequences in the future. Many experienced people at the department are consultants and losing them could severely worsen the issues related to a lack of knowledge among employees. In the cases of those who have this freedom as their main motive for being consultants it is unlikely that much can be done to better this situation, but for others it would be beneficial to try and employ as many as possible directly. A first step towards this should be taken by further exploring why people choose to work as consultants and potentially addressing as many of the reasons to this that can be identified as possible.

Another annoyance for employees has been the amount of time needed to educate new employees, and the associated loss of productivity. As explained in the theoretical framework, ‘the new guy’ is a common problem for productivity in agile teams; in this case we can also see that frustration has been caused by the rapid expansion, especially through the need to educate the new employees. There is a lack of academic sources on the subject, but in software development circles the loss of productivity when hiring new people has been widely accepted (see for example Yasin, 2015; Linders, 2016; Tirrell, 2018). Growing at a reasonable pace is seen as very important, and it is evident that the pace has been too high at the department. If the growth itself (and its effect on current employees) causes an even higher turnover, there is a risk that the entire situation will start to spiral. Attempting to replace those experienced developers who leave would cause an even higher strain on those who remain, and further increase the likelihood that they too choose to quit.

5.2.2 The Costs Associated with a Higher Turnover

Section 2.2 covers literature on productivity in agile teams, and takeaways include that keeping hold of experienced developers is important due to a higher value from experience earned in-house. Competency lost from experienced people leaving the organisation will take a long time to replace and reduce productivity even more than already is done through the process of adding new people, which was discussed in the previous section. This becomes especially important as organisations grow in size and networks of contacts are harder to acquire, meaning those already existing between experienced employees increase in value. Examples of these networks were mentioned by interviewees, for instance in the case of the function that had never been used in the old product, it seems safe to assume that it would have been next to impossible for an inexperienced employee to reach this conclusion, and instead a large amount of unnecessary work would likely have been carried out.

An increase in turnover would reduce the combined product knowledge in teams, the lack of which was already identified as a major issue driving costs for the department. The high autonomy levels of teams in the new organisational structure, including the fact that they are responsible for breaking down their own tasks, creates a higher need for product knowledge at a time when it is at its lowest. In hindsight, the decision to ramp up during an organisational transformation clearly had downsides. There may be upsides as well however; it is possible that making the transformation in itself was easier with new people, as there would not have been as much resistance to change. Old employees would have been more likely to hold onto an old culture and resist the implementation, whereas people entering from outside had no reason to go against the new structure. These factors are worthy of consideration for organisations trying to implement large-scale agile development in the future.

5.3 Software Quality and Complexity

The code complexity in the department's products was described by interviewees as high, making the code hard to maintain and understand. They also described the development and legacy of the code, referring to it as a living organism that has evolved over the years. This complexity is affecting development efficiency negatively as the code is harder to understand, maintain and develop. This section will further explore and reason around the causes for this increased complexity.

5.3.1 Demand and stress induced complexity

According to interviewees the department has over time been increasing scope and been very flexible towards their customers. They started off with just one product to one vehicle type, but they now have 15-20 different variants and products for different types of vehicles. The code base is described as a monolith, meaning it is not very modular, which has led to a lot of complex dependencies inside the code that are hard to solve between functions and products. Because they can't solve some of these dependencies, they are forced to include unused code in some of the variants as the product will not work without it. Some interviewees stated the product has probably been split up more than necessary, due to them having failed in keeping it as intact as possible. This has led to the large number of variants to their products. The

literature studied described that it is common for software to evolve over time through feedback loops from different stakeholders and external factors. Changes are inevitable but their negative impacts on code quality can be reduced if both the code and the organisation are prepared to deal with changes. The literature stated that over time this will increase the complexity and decrease the quality of the code if it is not actively managed with resources allocated specifically for this purpose. We conclude however, that if such allocation has been made at the department, it has been insufficient and should be considered for the future to ensure a higher level of internal quality.

Combining increased scope for the departments' products with the demand for producing features and functionality quickly have also affected the internal quality negatively. Interviewees described the pressure of delivering fast, the criticism they have received from management and the project itself as stressful and how they are building up technical debt in the code. They described some effects of this stress; planning for overcapacity in the teams, skipping development of features not believed to be essential, implementing bad design in the code, ignoring a need for refactoring, not testing enough and poor documentation. Producing at a pace so high that the developers cannot do the proper work lowers the maintainability of the code. An explanation to this could be seen from literature; too much focus has been put on delivering high external quality and not enough on keeping a good internal quality. The effects of this prioritisation could be seen when the interviewees described how much longer time it takes to fix a bug today compared to in the past. As teams are overflowed with work, the predictability of the department as a whole goes down, as they struggle to deliver what they have planned for. This causes negative effects for those later in the development chain as they cannot do what they intended to, as their plans depend on the deliveries from the previous team. So, the effect of overambitious planning in one team could be a chain reaction of lower productivity for other parts of the department.

We note the concern of the interviewees that the time pressure on the new platform will create technical debt. As the interviewees states that they sometimes do workarounds, solutions that are not optimal for the platform, because of decisions made in collaborating departments whose work concerns the platform, and developers at the department often find it easier and faster to do the workarounds rather than attempting to explain their reasoning to other departments in an effort to have them change their way. As these workarounds accumulate over time, there is a risk that they will create "accidental complexity" which will make it more difficult to further develop the platform.

5.3.2 Maintenance inefficiencies and costs

Literature studied described how maintenance is a large part of total costs for a software system. The poor quality of documentation combined with the high complexity, in both the product's composition and the solutions themselves, makes the code inefficient to maintain. The literature highlighted readability (documentation and solutions that are easy to understand) and internal code quality as important factors for maintainability, things that interviewees stated have failed over the years. The low readability and high complexity of the code and structure combined with the new employees' low domain knowledge lowers productivity. The department is aware of the problem and have implemented a new process in which the importance of documentation and design are emphasised; deliveries from both are included in the definition of done. This action is aligned with the recommendation from the

literature, as investing time to develop with a focus on maintainability has a great return in the long run.

The reviewed literature stated that most software development departments do not focus on maintainability, even though reports say that maintenance is a large part of the total costs for software development. We note however, that the case department are now aware and working actively with quality assurance and maintainability, knowing the cost for development is higher. The time pressure and its related stress are now forcing the developers to compromise on the quality of the code in order to reach their deadlines. This increases the risk of creating large technical debt once again, thus failing to ensure high maintainability and efficiency over the long term. By expanding on the unit tests the department is hopeful that they will start finding bugs and errors earlier on and therefore lower costs in the future. Interviewees stated however, that development times spent on the tests are equal to or even greater than the time spent on developing functionality. This will likely be a wise investment with great return in the long run; it is however important to realise that this is a big cost in the short term and another factor that lowers the productivity in terms of functionality, which increases the risk of being unable of deliver the new platform on time.

The goals of a tight deadline and keeping a high quality in the development phase are at least partially contradictory of each other and according to the iron triangle could be solved by increased costs (hiring more staff). However, if hiring more personnel decreases the short-term output as literature said, then the logic and reasoning are faulty and as such we believe the iron triangle model is too simple and not appropriate for agile software development. Simply adding more people in the belief that this will increase output will likely have the opposite effect, as the inexperienced developers are likely to deliver worse quality code and lower the output for the department as a whole for a time. Even though output and quality will increase long term, it is likely that these costs of expanding will exceed what can be gained during the project’s life time, and as such not contribute positively to it.

6 Conclusions

This study has identified cost drivers at a software development department in a large, multinational firm in the automotive industry. Further it has explored these drivers, both their appearance and the factors behind them as well as in what way they have resulted in an increased need of manpower at the department. This final part of our report aims to fulfil the study's purpose and answer the posed research questions. Finally, section 6.3 will provide some of our own thoughts on the matter, on how to avoid falling into similar situations in the future.

The fixed time frame of the large platform project and the delivery pressure it entails serves as a backdrop for all the other factors found in during the study. According to the iron triangle; the only option available for the department when both the quality and time dimensions were fixed was to increase costs, hiring more people. However, our findings indicate that the iron triangle is inaccurate for software development, as the hiring of new people have not increased output, but rather decreased it. Brook's law states that adding more people to a late project makes it later, which appears to be true in this case.

6.1 What are the main drivers of costs in software development at the department?

Four categories of cost drivers were identified during interviews and further divided into eighteen sub-categories. These are presented, with explanations, in table 6.1.

Table 6.1. The identified categories of cost drivers, with a brief explanation of the contents of each sub-category.

Category	Explanation
Organisation	The Organisation category covers inefficiencies derived from organisational structure and changes. The category has been divided into six sub-categories.
Work environment	Disturbing sounds and visual distractions because of the open landscape environment causes workers to lose their focus.
Boundaries between functions	Lack of communication with, and understanding from, surrounding departments creates unnecessary deficiencies. It also creates difficulties in synchronising the work within the department.
Criticism and stress	Implicates that the stress from delivering functionality at a high pace has a negative impact on the predictability and performance of the teams. The critique of inefficient work derives from both internal and external sources on all hierarchical levels.
Organisational changes	The rapid growth in size of the organisations and structural changes leads to misunderstandings which creates problems later on, in terms of misses in communication causing rework of functionality for example.
Investigation and planning	The gains from early investigations of solutions options in functionality are lost through translations into change requests. Doubt from the interviewees of return on investment in these investigations are identified and an improvement of the process is desired
Shortages of resources	Delayed delivery of hardware and resources within the organisation are stopping and/or slowing down work.

Staff	The Staff category covers inefficiencies generated from the personnel in the department. The category has been divided into five sub-categories.
Product competence	The lack of product competence primarily from new employees creates problems with understanding the ordered functionality. This makes them unable to understand old functionality and more likely to make the same mistakes that previously have been fixed or other errors, as they miss something which was intended to transfer from the old to the new platform. Another problem is that the new organisational structure is putting more responsibility on the team level which becomes problematic as the average knowledge is lower than before.
Employee turnover	The turnover of employees is creating an increased workload for the experienced personnel. As they need to help the new recruits by answering questions and other introductions of the systems. The dissatisfaction from a teacher role could increase the resigns from the experienced which creates a negative self-reinforcing spiral. On team level the turnover decreases the output if teams are split up and new members are joining.
Lack of experienced workers	The lack of experienced workers in both the market and within the organisation in terms of both programming and other roles within the organisation is noticed as the competence levels vary greatly. Partly is explained about new technical tools for the development which creates difficulties and learning time even for the more experienced.
Consultants	The consultants are costlier per hour compared hour which drives direct costs. The explanations for the use of consultants are legacy in culture from old regulations within the company and the convenience of easy recruiting. The consultants are often considered it to be beneficial of being a consultant over a full-time employee at the company because of more flexibility.
Learning time for new employees	The cost of learning time for new employees, even how skilled they are they need time to learn the system which lowers the productivity for the unit as a whole. The lower productivity comes from disturbance of the experienced and high performing workers in terms of questions which make them lose their flow and the time to create a new working culture for the new team and get to know each other.
Code Complexity	The Code Complexity category covers complexity of the code and how easy it is to work with and understand. The category has been divided into five sub-categories.
Documentation	The lack of documentation increases the time of understanding the previous code which decreases the productivity for maintenance. Poor documentation in the old platform makes it difficult to understand when creating the new platform which increases the risk of failing develop the new and decreases the productivity. Increased focus on documentation is noticed in the organisation which increases costs short term.
Increased number of products and variants	Over the year's one product has been divided into many more which increases the complexity and unintended dependencies which decreases the productivity.
Client created complexity	When customer and business responsible within the firm demands changes for differentiation it creates increased complexity which increases the cost over time.
Coding legacy	Describes the legacy of the years of maintaining the old platform which covers dependencies, "tweaks", high-end solutions, and misunderstood solutions for the new platform.

Stress induced complexity	The limited time within the development force the developers build the system in a pace which makes them unable to do it correctly which increases the technical debt. The predictability also goes down as the teams plan their workload over their capacity which they then fail to deliver. The missed delivery on time creates difficulties in synchronisation and cost in rescheduling and planning for surrounding functions in the department.
Code Quality	The Code Quality category includes internal quality of the code and how the quality is assured. The category has been divided into two sub-categories.
Quality assurance	Costs for testing and assuring the quality of the code has increased in the old product of the increased complexity. The costs are for both testing and maintaining the functionality for assuring an acceptable code quality.
Test strategy	Describes the costs from a new test strategy with a changed focus in how and where the quality assurance occurs. The strategy changes the software development process and takes according to the interviewees at least twice the time for developing the functions. The expectation is however that this cost in short term will be beneficial long term.

This identification is one scientific contribution of this thesis, as several of the factors are likely to exist also in other organisations with similar contexts in terms of time- and delivery pressure. Further empirical studies will be needed to assess the relative effects of the drivers, as well as how much they differ between different organisations and situations. To do this, the method used in this study should be helpful to researchers.

6.2 How do these affect and reinforce each other?

Several of the identified cost driver categories are affecting each other in crucial ways. Average product competence is lowered by the employee turnover, as new employees have less knowledge in the beginning and experienced ones need to spend their time teaching instead of coding, whereas the organisational changes increase its importance. Furthermore, the lack of product competence also worsens the negative effects of the work environment, as more conversations are needed when fewer individuals have enough knowledge to develop independently.

Issues of code complexity are also worsened by the lowered levels of product competence. Developer product knowledge has been found to be an important productivity factor for software development, having less knowledge means you produce less and lower quality code. This risk creating a negative spiral, as more complex code requires higher competence to understand, but the lower average competence available produces lower quality code, more complex and with more bugs and errors. If the code complexity is not carefully managed this could lead to a situation similar to the one experienced at the department in regard to the old platform; where only the most senior developers can make sense of and continue working with the functionality. This would also increase the costs associated with maintenance, whereas the new focus on maintaining high quality, both internal and external, from the start may cost more early on, but likely saves on maintenance costs down the line.

Traditional rationale states that employing more people increases output. This has led the department to ramp up in order to be able to finish their large platform project on time. For software development however, this is not the case. Brook's law, which is generally accepted as true in this area, states that adding more people to a late software project will make it later.

Before any new developer is up to speed and producing, he will have slowed down development considerably for a period. Growing at a stable pace was seen as a crucial factor for success in agile projects in the literature used for this study. Lacking an understanding for this, criticism has been targeted towards the department from other parts of the organisation over the productivity levels. This has caused additional stress and lead developers to start taking shortcuts in development; potentially creating more complexity which will need to be managed down the line. The risk is that this will counteract the good efforts made with the testing strategy and the efforts spent there and that maintenance costs will increase with time regardless. Additionally, there is a risk that the prioritisations made by individual developers, skipping or simplifying functionality, are not the best for the larger project. This risk is further increased by the low product competence; the less knowledge one has about the larger picture, the less likely such decisions are to be the best ones.

Besides the obvious effect, increased costs, of the cost drivers identified and the way they amplify each other, an overhanging risk is that their effect on the morale of employees will cause more experienced employees to quit, which could set of a negative spiral of lowered productivity, increased stress on those who remain and further resignations.

6.3 Actions for lowering the impacts from the cost drivers

The primary advice we can give to others considering major software projects within the context of a large organisation is to avoid falling prey to time pressure. The effects of not having enough time were acutely seen throughout this study and it is also the prevailing opinion in literature that time cannot be compromised in many areas. Growing in size takes time, avoiding rampant complexity demands time be put in early stage planning and ensuring a good implementation of agile methods requires settling time as well. Lacking this time, there will be issues. But despite this, some things can be done to at least reduce the impact of time pressure on the final results to a degree.

Firstly, the old maxim of doing one thing at a time can be applied. At the department a major organisational change was undertaken during a time of major delivery pressure and employee growth. Adding further to this, at the same time the demands on code quality and documentation were increased. This combination has undoubtedly caused the issues to affect and enhance each other, in the manners described in the previous section. Other organisations embarking upon large project with time pressure would do well in limiting the amount of organisational changes during this time, as well as ensuring that necessary employees have been sufficiently integrated before the project is started, or taking their learning time into account when estimating the effort needed.

To ensure a sustainable code quality and lower the total life cycle costs, it is important for an organisation to agree upon a strategy that emphasises maintainability and an acceptance of short-term costs which give long-term benefits. High effort should be put into prioritising which functions and modifications demanded from the business side should be implemented, to ensure high customer value for the development effort. Especially important is that this is accepted and acknowledged by managerial staff, as pressure from them will, accidentally and easily, spill over to developers and force them to lower quality in an effort to cope with the demands.

7 References

Andersson, M., & Wernberg, J. (2018). *Den Osynliga Infrastrukturen*. Stockholm: Swedsoft. From http://swedsoft.se/wp-content/uploads/sites/7/2018/03/Den-osynliga-infrastrukturen.pdf

Boehm, B., & Turner, R. (2005). Management challenges to implementing agile processes in traditional development organizations. *IEEE software*, *22*(5), 30-39.

Brennan, A., Chugh, J. S., & Kline, T. (2002). Traditional versus open office design: A longitudinal field study. *Environment and behavior*, *34*(3), 279-299.

Brooks Jr, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Pearson Education India.

Card, D. N. (2006, February). The challenge of productivity measurement. In Pacific Northwest Software Quality Conference (pp. 1-10).

Chikofsky, E. J., & Cross, J. H. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE software*, *7*(1), 13-17.

Cohen, D., Lindvall, M., & Costa, P. (2004). An introduction to agile methods. *Advances in computers*, *62*(03), 1-66.

Darke, P., Shanks, G., & Broadbent, M. (1998). Successfully completing case study research: combining rigour, relevance and pragmatism. *Information systems journal*, *8*(4), 273-289.

Dvorak, D. (2009, April). NASA study on flight software complexity. In *AIAA Infotech@ Aerospace Conference and AIAA Unmanned... Unlimited Conference* (p. 1882).

Engelbertink, F.P. and Vogt, H.H. (2014) How to Save on Software Maintenance Costs. Omnext White Paper. Accessed 3 May 2018.

Erlikh, L. (2000). Leveraging legacy system dollars for e-business. *IT professional*, *2*(3), 17-23.

Faraj, S., & Sproull, L. (2000). Coordinating expertise in software development teams. *Management science*, *46*(12), 1554-1568.

Foster, G., & Gupta, M. (1990). Manufacturing overhead cost driver analysis. *Journal of Accounting and Economics*, 12(1-3), 309-337.

Fowler, M., Highsmith, J., (2001, August 1). The Agile Manifesto. Dr Dobb's The World of Software Development. From http://www.drdobbs.com/open-source/the-agile-manifesto/184414755?queryText=the%2Bagile%2Bmanifesto

Godfrey, M. W., & German, D. M. (2014). On the evolution of Lehman's laws. *Journal of Software: Evolution and Process*, *26*(7), 613-619. 10.1002/smr.1636

Guo, Y., Spínola, R. O., & Seaman, C. (2016). Exploring the costs of technical debt management–a case study. *Empirical Software Engineering*, *21*(1), 159-182.

Grubb, P. and Takang, A.A. (2003) Software Maintenance: Concepts and Practice. 2nd Edition, World Scientific Publishing

Company, Singapore.

Hsia, P., Hsu, C. T., & Kung, D. C. (1999). Brooks' law revisited: A system dynamics approach. In *Computer Software and Applications Conference, 1999. COMPSAC'99. Proceedings. The Twenty-Third Annual International* (pp. 370-375). IEEE.

Humphrey, W. S. (2001). *Winning with software: An executive strategy*. Pearson Education.

Hunt, B., Turner, B., & McRitchie, K. (2008). Software maintenance implications on cost and schedule. Paper presented at the 1-6. doi:10.1109/AERO.2008.4526688

Ishikawa, K. (1982). Guide to quality control (No. TS156. I3713 1994.).

Kim, J., & De Dear, R. (2013). Workspace satisfaction: The privacy-communication trade-off in open-plan offices. *Journal of Environmental Psychology*, *36*, 18-26.

Kemerer, C. F., & Slaughter, S. (1999). An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering*, *25*(4), 493-509.

Levy, J. (1999), "in my opinion", CIO, Vol.12 Nr. 23, p.10

Linders, B. (2016, March 2). Nurturing and Growing Agile Teams. *InfoQ*. From https://www.infoq.com/news/2016/03/nurturing-growing-agile-teams

Lindsjørn, Y., Sjøberg, D. I., Dingsøyr, T., Bergersen, G. R., & Dybå, T. (2016). Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*, *122*, 274-286.

Livermore, J. A. (2008). Factors that Significantly Impact the Implementation of an Agile Software Development Methodology. *JSW*, *3*(4), 31-36.

Lyu, M.R. (1996*) Handbook on Software Reliability Engineering.* McGraw-Hill, USA and IEEE Computer Society Press, Los Alamitos, California, USA.

Manawadu, C. D., Johar, M. G. M., & Perera, S. S. N. (2015). Essential Technical Competencies for Software Engineers: Perspectives from Sri Lankan Undergraduates. *International Journal of Computer Applications*, *113*(17).

Maylor, H. (2010). *Project management*. 4th ed. Harlow: Financial Times Prentice Hall.

Melo, C. D. O., Cruzes, D. S., Kon, F., & Conradi, R. (2013). Interpretative case studies on agile team productivity and management. *Information and Software Technology*, *55*(2), 412-427.

Misra, S. C., Kumar, V., & Kumar, U. (2009). Identifying some important success factors in adopting agile software development practices. *Journal of Systems and Software, 82*(11), 1869-1890.

Misra, S. C., Kumar, V., & Kumar, U. (2010). Identifying some critical changes required in adopting agile practices in traditional software development projects. *International Journal of Quality & Reliability Management, 27*(4), 451-474.

Nahapiet, J., & Ghoshal, S. (2000). Social capital, intellectual capital, and the organizational advantage. In *Knowledge and social capital* (pp. 119-157).

Nanni, A. J., Dixon, R., & Vollmann, T. E. (1992). Integrated performance measurement: management accounting to support the new manufacturing realities. *Journal of Management Accounting Research* (Fall), l-19.

Opdenakker, R. (2006, September). Advantages and disadvantages of four interview techniques in qualitative research. In *Forum Qualitative Sozialforschung/Forum: Qualitative Social Research* (Vol. 7, No. 4).

Pope, C., Ziebland, S., & Mays, N. (2000). Qualitative research in health care: analysing qualitative data. *BMJ: British Medical Journal, 320*(7227), 114.

Porter, M. E., & Heppelmann, J. E. (2015). How smart, connected products are transforming companies. *Harvard Business Review, 93*(10), 96-114.

Purna Sudhakar, G., Farooq, A., & Patnaik, S. (2011). Soft factors affecting the performance of software development teams. *Team Performance Management: An International Journal, 17*(3/4), 187-205.

Reiner, R. & Schaper, M. (2010). *Tackling IT complexity in product design* (Digital McKinsey, 2010:3). McKinsey & Company. From https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/tackling-it-complexity-in-product-design

Rivera-Ibarra, J. G., Rodríguez-Jacobo, J., & Serrano-Vargas, M. A. (2010, March). Competency framework for software engineers. In *Software Engineering Education and Training (CSEE&T), 2010 23rd IEEE Conference on* (pp. 33-40). IEEE.

Rostkowycz, A. J., Rajlich, V., & Marcus, A. (2004, September). A case study on the long-term effects of software redocumentation. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on* (pp. 92-101). IEEE.

Rus, I., & Lindvall, M. (2002). Knowledge management in software engineering. *IEEE software, 19*(3), 26.

Seaman, C. B. (1999). Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4), 557-572.

Schmid, K. (2013). A formal approach to technical debt decision making. Paper presented at the 153-162. doi:10.1145/2465478.2465492

Taba, S. E. S., Khomh, F., Zou, Y., Hassan, A. E., & Nagappan, M. (2013, September). Predicting bugs using antipatterns. In *Software Maintenance (ICSM), 2013 29th IEEE International Conference on* (pp. 270-279). IEEE.

Tirrell, C. (2018, January 26). Agile Development Teams: Size Matters. *Modus*. From https://moduscreate.com/blog/agile-development-teams-size-matters/

Tryggeseth, E. (1997). Report from an experiment: Impact of documentation on maintenance. *Empirical Software Engineering, 2*(2), 201-207. doi:10.1023/A:1009778023863

TT. (2017, February 24). Skenande brist på programmerare. *TT*. From http://nyteknik.se

Turley, R. T., & Bieman, J. M. (1995). Competencies of exceptional and non-exceptional software engineers. *Journal of Systems and Software*, *28*(1), 19-38.

Wagner, S., & Ruhe, M. (2008). A Systematic Review of Productivity Factors in Software Development. *n3n*.

Welch, C., & Piekkari, R. (2006). Crossing language boundaries: Qualitative interviewing in international business. *Management International Review*, *46*(4), 417-437.

Wolff, E., & Johann, S. (2015). Technical debt. *IEEE Software*, *32*(4), 94-c3. doi:10.1109/MS.2015.95

Yasin, A. (2015, October 27). Don’t Destroy Your Dev Team By Growing. *The Startup*. From https://medium.com/swlh/don-t-destroy-your-dev-team-by-growing-eef50d83090e