Santeri Heikkilä

Verification of Large-Scale Agile Transformation: Case Nokia

Master’s thesis

ABSTRACT

Author: Heikkilä, Santeri
Name of the thesis: Verification of Large-Scale Agile Transformation: Case Nokia
Year: 2017    Place: Espoo, Finland

Master’s thesis. Lappeenranta university of technology, Industrial Engineering and management, GMIT.
77 pages, 13 charts, 8 figures, 8 tables and 1 appendix

Keywords: Agile, Scaling Agile, Large-Scale Agile, Large-Scale Scrum, Nokia, Measuring Software Development, Software Metric

In recent years, scaling agile practices has risen as one of the most burning research topics in software development management. This thesis focuses on providing ways to verify expected benefits of large-scale agile transformation in a case company Nokia. Selected framework for scaling agile is called large-scale scrum or LeSS. Key expected benefits for Nokia are shorter development lead times, increased quality and increased employee and customer satisfaction. One of the key results of this thesis are proposed, Nokia tailored, metrics for measuring benefits listed above. However, as LeSS pilot at Nokia was still in its early phases during the thesis process, deep down analysis using metrics developed was not possible to conduct within the time limits of the thesis project. None the less, from the preliminary results it is visible that adoption of LeSS way-of-work brought some of the expected benefits. These were, for example, introduction of a wider scope of development work in development teams, improved testing practices and improved employee satisfaction.
FOREWORDS

The aim of this paper is to provide concrete metric suggestions for Nokia Networks LTE R&D to measure the success of LeSS way-of-working pilots. Paper also discusses challenges of measuring software development. Improvement actions are suggested regarding these challenges. Extensive literature review and case company analysis were conducted for the thesis. Since author has previous working experience at the company in project management tasks he could suggest Nokia specific improvements regarding measuring practices, both in general and in pilot verification context.

Special thanks go to the thesis instructors, both at Nokia for Janne Kohvakka and at LUT for Ville Ojanen. With their support, help and knowledge in Nokia specific topics and in general research issues it was possible to finish the thesis within schedule and with valid results. Thank you for close family members and friends for motivation and support along the thesis process. Thank you also for managers at Nokia who gave the opportunity to conduct thesis project at the company. Hopefully this thesis will provide desired benefits for Nokia and much more.

Santeri Heikkilä
Espoo, Finland
26.6.2017
TABLE OF CONTENTS

1 INTRODUCTION ................................................................................................. 1
  1.1 BACKGROUND................................................................................................. 1
  1.2 RESEARCH GAP, OBJECTIVES AND QUESTIONS............................................. 2
  1.3 STRUCTURE OF THE PAPER ............................................................................. 3

2 METHODS OF SOFTWARE DEVELOPMENT AND MEASURING: LITERATURE REVIEW... 5
  2.1 TRADITIONAL WATERFALL MODEL ................................................................. 5
  2.2 NEW LEAN AND AGILE MODELS...................................................................... 8
    2.2.1 Scrum......................................................................................................... 10
    2.2.2 Advantages of agile development models ................................................ 13
  2.3 SCALABILITY OF AGILE MODELS .................................................................. 15
  2.4 MEASURING SOFTWARE DEVELOPMENT ...................................................... 18
    2.4.1 Checklist for developing software metrics ................................................... 20
    2.4.2 Key issues in measuring and comparing the status of software development .................................................................................................................. 22

3 INTRODUCTION TO NOKIA LTE R&D ................................................................ 25
  3.1 WAY OF WORK IN LTE R&D ......................................................................... 26
  3.2 MEASURING & KPI’S ...................................................................................... 34

4 LESS FRAMEWORK .................................................................................................. 36

5 RESEARCH METHODS ........................................................................................... 43

6 RESULTS .................................................................................................................. 46
  6.1 RQ1: HOW TO DEFINE AND MEASURE SUCCESS FACTORS FOR LeSS-ADOPTION WITHIN A FEATURE TEAM? ...................................................... 46
    6.1.1 Level of LeSS way-of-work adaptation within LeSS pilot teams .................. 47
    6.1.2 Development lead time ............................................................................ 49
    6.1.3 Usefulness of the product for a customer .................................................. 51
    6.1.4 Freedom from deficiencies ....................................................................... 53
    6.1.5 Employee satisfaction .............................................................................. 56
  6.2 RQ2: WHICH OF THE CURRENT NOKIA METRICS SHOULD BE USED TO DEFINE SUCCESS OF THE LeSS-PILOT? ................................................................. 57
  6.3 RQ3: WHAT ARE THE PRELIMINARY RESULTS AND IMPLICATIONS OF NOKIA’s LeSS-PILOT? ................................................................. 60
    6.3.1 Whodidwhat-analysis results .................................................................... 60
    6.3.2 Way-of-working survey results .................................................................. 61

7 DISCUSSION ............................................................................................................ 70
7.1 SUGGESTIONS FOR MANAGEMENT REGARDING MEASURING THE SOFTWARE DEVELOPMENT AND LeSS PILOTS. ........................................................................................................................................... 70
7.2 MAIN IMPLICATIONS OF THE WHODIDWHAT- AND WOW-SURVEY ANALYSIS. .............................................................. 74

8 SUMMARY ........................................................................................................................................................................... 76
REFERENCES ............................................................................................................................................................................. 79
APPENDICES ............................................................................................................................................................................. 85
LIST OF CHARTS

Chart 1. SW Feature activation rate in customer base stations (%) for features (1-32) in the example release (Nokia, 2017). .......................................................... 52
Chart 2. LeSS-team contributions to additional system components. ......................... 62
Chart 3. LeSS-team members’ base station level of understanding. ........................... 63
Chart 4. LeSS-team members’ contribution to entity testing before LeSS-pilot. ........... 64
Chart 5. LeSS-team members’ contribution to entity testing after first three iterations of LeSS-pilots. .......................................................... 64
Chart 6. non-LeSS team members’ visibility to ET cases and criteria. ......................... 64
Chart 7. LeSS-team members’ visibility to ET cases and criteria. .............................. 64
Chart 8. non-LeSS team members’ opinion about ET responsibility. ......................... 65
Chart 9. LeSS-team members’ opinion about ET responsibility. .............................. 65
Chart 10. Amount of communication between LeSS team members and relative change during first three iterations of LeSS pilots. .......................................................... 66
Chart 11. Change in lead times according to LeSS-team members. ........................... 68
Chart 12. Change in quality according to LeSS team members. ............................... 68
Chart 13. Suitability of LeSS-wow to Nokia LTE R&D according to LeSS team members... 69

LIST OF FIGURES

Figure 1. Phases of Waterfall model (Royce, 1970). .................................................. 6
Figure 2. The scrum model (Schwaber, 2004). ......................................................... 13
Figure 3. Risk exposure profile, where P(L) is probability of loss and S(L) size of loss (Boehm, 2002). ............................................................................................................. 14
Figure 4. MN business line sites (Nokia, 2017). .......................................................... 26
Figure 5. The setup around FOT. Colors mark team’s functional or component specialization. (Nokia, 2017). ........................................................................................................... 30
Figure 6. The scope of system testing and verification (eNodeB equal for base station). (Nokia, 2017). ............................................................................................................. 31
Figure 7. LTE metrics and use purposes (Nokia, 2014). ................................................. 34
Figure 8. LeSS huge framework (Larman and Vodde, 2017). ........................................ 37
LIST OF TABLES

Table 1. Summary of structure of the paper. .................................................................3
Table 2. Suggested research topics in the field of large-scale agile (Dingsøyr and Moe, 2013, p.38).................................................................................................................................16
Table 3. Classification of software metrics (Misra and Omorodion, 2011, p.2). .................19
Table 4. Roles and responsibilities between different way-of-working roles. ......................38
Table 5. Hartmann & Dymond's checklist applied to function point adjusted LeSS feature team development lead time metric. .................................................................................................................................50
Table 6. Hartmann & Dymond's checklist applied to functional defect density per requirement area metric. .........................................................................................................................54
Table 7. Pre-pilot whodidwhat-analysis results..................................................................61
Table 8. Whodidwhat-analysis results for first 4 LeSS-pilot iterations ..............................61

LIST OF APPENDICES

Appendix 1. Way-of-working survey................................................................................85
LIST OF ABBREVIATIONS

CFAM: Common Feature Analysis Module
CPU: Computer Processor Unit
CuDo: Customer Documentation
DoD: Definition of Done
e2e: End to End
EFS: Entity Feature Screening
ET: Entity Testing
FB: Feature Build
FBP: Feature Build Plan
FDD: Frequency Division Duplexing
FOT: Feature Owner Team
FS: Feature Screening (process)
GUI: Graphical User Interface
LeSS: Large Scale Scrum-framework
LPO: Local Product Owner
LTE: Long Term Evolution (Network technology commonly known as 4G)
LUT: Lappeenranta University of Technology
PC: Personal Computer
RAM: Requirement Area Manager
R&D: Research and Development
SFS: System Feature Screening
SPM: System Program Manager
SVN: Apache Subversion
TDD: Time Division Duplexing
URL: Uniform Resource Locator
WoW: Way-of-Work
1 INTRODUCTION

This thesis aims to produce concrete metric suggestions for verifying the success of a new way-of-work or WoW pilot at Nokia. This work is a multi-method research, in which systematic literature review, survey and version control statistical analysis are utilized. Metric suggestions and preliminary results of the metrics are described and analyzed in the results. Other issues that were discovered during thesis project are discussed as well.

1.1 Background

Nokia is currently one of the world’s largest mobile network manufacturers. In addition to mobile networks, Nokia operates in new technologies, such as healthcare, digital imaging and virtual reality. This work focuses on Nokia’s mobile networks business line and more specifically on the LTE or long-term evolution business unit. From now on, term Nokia is used to refer to Nokia’s LTE business unit, unless specified otherwise.

Current Nokia way-of-working model is based on the principles of agile software development. There are approximately 5000-6000 people working in Nokia research and development (R&D). R&D is currently organized around scrum teams, which are working independently on different system components of the product. Nokia’s mobile networks organization has grown overtime with numerous merges with competitors such as Siemens, Motorola, Panasonic and Alcatel-Lucent. This non-organic growth has had negative effect on the organizational structure. In fact, one of the biggest challenges that Nokia is facing in the R&D is the loss of customer focus due to the complex organization and coordination of organization itself. In other words, it is difficult for a huge organization to respond to constantly changing customer demands.

To address these issues, Nokia is piloting Large Scale Scrum or LeSS way-of-working model, which is applicable to multiple teams working together on one big product. LeSS promises to deliver the same benefits as scrum, but in a way that it can be also applied for large organizations with multiple scrum teams without losing efficiency. One of the objectives of this thesis is to suggest ways to monitor the effects of adapting LeSS way-of-work. The purpose is to examine what kind of concrete advantages, if any, adaptation of LeSS-framework will bring to Nokia.
1.2 Research gap, objectives and questions

There is not much information about adapting LeSS – framework in the literature except for the case studies found in the LeSS-framework home page (Larman and Vodde, 2017). Only one related scientific case study of Ericsson regarding release planning in LeSS was found (Heikkilä et al., 2013). There is, however, lots of information about the challenges and different methods of large-scale agile available. This clearly indicates relevancy of the topic. Also, the nature of LeSS framework, which emphasizes learning and continuous inspection and adaptation based on retrospectives, means that the framework will be most likely tailored to fit Nokia. The need for the thesis work is therefore obvious. Scientific contribution of this master’s thesis to the society is also notable in a way that it will be one of the first scientific releases analyzing the adaptation and use of LeSS framework in large organizations.

Main objective of this thesis is to provide ways to examine the effects of implementing LeSS framework in Nokia’s LTE business line. Thesis is done simultaneously with the LeSS pilot at Nokia. Therefore, thesis examines the preliminary results of LeSS adaptation. Thesis is analyzing changes in software development way-of-working practices. Hence, software development teams, called feature teams at Nokia, will be given especial attention.

The research questions for the thesis are set as follows

\textbf{RQ1: How to define and measure success factors for LeSS-adoption within a feature team?}

\textbf{RQ2: Which of the current Nokia metrics should be used to define success of the LeSS-pilot?}

\textbf{RQ3: What are the preliminary results and implications of Nokia’s LeSS-pilot?}

The aim of the first research question is to find suitable metrics for measuring the effects of adopting LeSS way-of-work. Second research question focuses on examining Nokia’s current metrics and checks if the current metrics could be utilized for pilot verification purposes. The third and the last research question presents the results and implications of the selected and applied metrics. Other findings related to way-of-working practices or measuring software development that were discovered during thesis project are also presented in this paper.
1.3 Structure of the paper

This sub-chapter explains the structure of the thesis. First, findings from the literature are examined in chapter 2. Software development models such as traditional waterfall model and scrum model are described. Also, the characteristics of measuring software development are discussed in chapter 2. Chapter 3 describes the way-of-work- and measurement practices at Nokia. Chapter 4 presents the LeSS-framework piloted at Nokia in more detail and compares it to the current way-of-working practices. In chapter 5, the research methods of this thesis are further defined and explained. Chapter 6 and 7 present the results of the thesis and the answers to the research questions posed above. Discussion about the limitations and suggestions for the management is presented in chapter 7. Table 1 summarizes the linkages between the chapters.

<table>
<thead>
<tr>
<th>Input</th>
<th>Chapter</th>
<th>Output</th>
</tr>
</thead>
</table>
| • Need for the work from company’s side.  
• Research gap. | 1. Introduction | • General introduction to the topic  
• Research Questions |
| • Research questions to find related info from the literature. | 2. Methods of Software Development and Measuring: Literature Review | • Key issues and solutions found from literature relating to the topics in research questions.  
• Overview of both traditional and agile software development methodologies.  
• Overview of scaling agile practices and measuring software development. |
|----------------------------------|-------------------|----------------------|
| • Internal Nokia material and discussions with Nokia managers. | • Overview of Nokia LTE R&D WoW.  
• Overview of the challenges within Nokia LTE R&D.  
• Overview of metrics within Nokia LTE R&D. | • The purpose and the scope of LeSS-pilot.  
• Overview of LeSS framework and principles. |
| 6. Results | 7. Discussion | 8. Summary |
| • Nokia LTE R&D WoW description.  
• LeSS framework literature.  
• Nokia LeSS WoW adaptation material. | • Answers to the research questions.  
• Results of the studies. | • Summary of the key findings and implications. |
| 5. Research Methods | 6. Results | 7. Discussion |
| • Research goals  
• Research questions | • Written and empirical material for the thesis  
• Research process and study analysis | • Implications of the study results.  
• Limitations of the study. |
| 6. Results | 7. Discussion | 8. Summary |
| • Written and empirical material for the thesis  
• Research process and study analysis | • Answers to the research questions.  
• Results of the studies. | • Summary of the key findings and implications. |
2 METHODS OF SOFTWARE DEVELOPMENT AND MEASURING: LITERATURE REVIEW.

Since the beginning of software development after WWII and after first commercial embodied software patent in 1968 (Con Diaz, 2015), various software development methodologies have been presented. Especially in the past few decades, significant effort has been invested in identifying good practices and models to improve the efficiency of software development. This has led to a birth of a large number of new software development methodologies (Jiang and Eberlein, 2009). These software development methodologies, both old and new, are usually split into two categories: traditional plan-driven methodologies, such as waterfall model (Royce, 1970), and more lightweight agile methodologies such as scrum (Schwaber, 2004) and extreme programming (Beck, 2004).

In this chapter, traditional waterfall and scrum methodologies are presented and described with the help of structured literature review. After this, the scalability of the agile models is discussed since traditionally agile software development models have been thought only to be applicable for small projects. The specialties regarding measuring software development are also discussed in this chapter.

2.1 Traditional waterfall model

Traditional software development can be characterized to have a heavy focus in long-term planning. The waterfall model was for a long time the most popular software development model before introduction of agile methodologies. The general opinion is that traditional waterfall model was first introduced by Winston Royce in the 1970. The model was created based on Royce’s own opinions and experiences from IBM about the way how large software development processes should be conducted (Royce, 1970). However, there already were some approximations of the waterfall model available before Royce’s paper was published (Boehm, 1987). It is also worth to be noted, that the term “waterfall model” was not used by Royce when he first introduced the model.

The waterfall model is process based and heavily emphasizing documentation. Royce splits his model into two fundamental parts that every software project has regardless of the size of the
project: analysis and coding. However, especially for bigger software development projects many additional steps are required (Royce, 1970). Different phases of the model are presented in Figure 1. As seen from Figure 1, the software development starts from determining requirements for the whole system and for the software. This is followed by the analysis of the requirements which is done before designing the program. After these phases are completed, the actual coding work can begin. After the coding, testing is conducted before implementing the program in practice. Hence, the name is waterfall model.

![Figure 1. Phases of Waterfall model (Royce, 1970).](image)

The model also includes iterations between previous and following phases, but only between these. For example, if problems are found during testing, the program is returned for re-coding. Royce himself notes that implementing the model as depicted in Figure 1 is “risky and invites failure” (Royce, 1970). As testing phase is at the end of the development cycle, it is possible that some of the faults that are caused by fundamental issues in program design are discovered only after coding work. This poses the risk of having to re-do the entire coding work if program design is needed to be changed fundamentally. Same is true for mapping the requirements. Some of the critical requirements for a program can only become visible during the technical design process of the program. This would mean that the requirements and analysis phase should be re-done.
In order to manage and avoid the risks, Royce presented five additional processes. One of these is to set-up a preliminary program design phase between software requirements and analysis phase. By doing so relative vacuum between software requirements and actual working software can be filled. This will prevent requirements to be irrelevant in real world. This phase could be done by utilizing program designers to check the requirements and by writing an overview document of the system so that all the members of the development process would have elementary understanding of the system. The role of documentation should be emphasized for other reasons as well according to Royce. By documenting the process of development, managers can focus on essential issues. Also, the dependency on certain people is reduced due to shared knowledge. Sharing of the knowledge also makes possible redesigning much more easy and manageable. In addition, Royce suggests that every new software being developed should be done at least twice so to speak, for example, with the help of simulation. By doing an overall simulation of the project, some of the hiding issues or problems could be discovered in an early phase of the program development work thus reducing the required re-work significantly. It is suggested to monitor testing and use specialized testers in addition to involving customer to make sure that the product meets customer’s requirements and needs. (Royce, 1970).

However, usually when waterfall model is described, the risk preventing steps presented by Royce are excluded and the model is only analyzed and used as in Figure 1. In a way, it could be said that the Royce’s original paper written in 1970 is somewhat misunderstood (Journeys of a young Software Engineer, 2017). Most of the steps to prevent risks presented by Royce are actually something that are nowadays considered to be agile practices. For example, customer involvement and making sure that all development personnel understand the principles of the program are both key agile principles (Highsmith and Fowler, 2001).

Partly because of this simplification of the waterfall model, the model has some notable downfalls as discussed earlier. The danger in waterfall model is that it drives development work to produce more documents with the aim of reducing rework between phases of the model instead of motivating software developers to actually solve the critical issues (Boehm, 1987). This means that the most severe problems might not be given enough attention and additional rework will occur as the software does not meet the requirements or is faulty. In addition,
learning to work and working according to the processes can be very time consuming. Some of the steps may not be even necessary to perform in every occasion. This in turn can lead to inefficiency in the development. More importantly, as the requirements for software are defined before the start of coding, the model cannot respond fast enough to changing customer needs during relatively long development periods that are typical for software development. In other words, traditional methods are not agile enough in today’s dynamic and connected world!

2.2 New lean and agile models

Terms “agile” and “agility” trace back to the beginning of 1990’s when the term lean development started appearing in manufacturing industries (Jiang and Eberlein, 2009). During that time, some of the most popular agile software development methods were introduced. Hirota Takeuchi and Ikujiro Nonaka introduced the ancestor of agile scrum method in 1986 (Takeuchi and Nonaka, 1986) which was later formalized, codified and named as scrum method by Jeff Sutherland and Ken Schwaber in 1995 in their work “SCRUM Software Development Process” (ScrumGuides, 2016). Sutherland and Schwaber have since published many papers about scrum and methodologies behind it. The fundamental difference with traditional methods is that traditional methods aim to produce one final working product, whereas agile methods aim to produce smaller individually working pieces of the final product that are incrementally completing the final product.

In 2001, a group of 17 agile practitioners gathered to create “Agile Manifesto” (Misra et al., 2012). They created the manifesto as they wanted to tell the software development communities that agile is not just an anarchist way of working as a software developer without no respect to long-term planning or traditional processes. The manifesto stated the purpose and benefits of working agile. It also gave the guidelines on how to work agile. The value statement of the manifesto is the following:

"We are uncovering better ways of developing software by doing it and helping others do it.

- Through this work, we have come to value:
- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

That is, while there is value in the items on the right, we value the items on the left more” (Highsmith and Fowler, 2001).

The four value statements of the agile manifesto show how agile practitioners acknowledge the importance of typical traditional software development practices, such as processes, documentation and planning. However, it also indicates how agile values should be considered as even more important. Out of these value statements practitioners derived a set of 12 formally written agile principles about working agile:

“We follow the following principles:

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity—the art of maximizing the amount of work not done—is essential.
- The best architectures, requirements and designs emerge from self-organizing teams.
• At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly. “(Highsmith and Fowler, 2001).

However, these agile principles were not new. Most of the software best practices, such as software architecture, continuous integration and incremental releases, were introduced long before agile manifesto (Janes and Succi, 2014, p.74). These best practices were then later combined to form what is now known as agile methodology (Bomarius et al., 2009). Some of these best practices were even recognized by the developer of waterfall model Winston Royce in 1970, as mentioned in the previous chapter. One of the reasons why agile methods have gained popularity in recent years is the development of technology and computers. During the 1960’s, when traditional waterfall model emerged, programming languages were still inefficient, computer processor units (CPU) slow and PCs had very limited memory. Waterfall model tried to minimize the time spent on coding i.e. time spent using the expensive and rare computers. Whereas nowadays PCs are on everybody’s desk. They are very powerful yet cheap. Programming languages are efficient and graphical user interface (GUI) has allowed fast implantation of prototypes. The development of PCs has enabled and at the same time required that the software projects grow considerably bigger and more complex. As the programs are more extensive with more possible functionalities in them, the customer requirements are changing more often nowadays than before. Increased complexity of developed programs means that developing organization is constantly discovering new aspects of the behavior of the program, also during development. All this means that software project in many cases cannot be planned ahead as the requirements for the software are constantly changing. (Jiang and Eberlein, 2009).

2.2.1 Scrum

In the article called ”The new new product development game” published by Takeuchi and Nonaka in 1986, authors introduced a new model for product development. The model was described as a more holistic approach to product development. The model was compared to a game of rugby where the ball gets passed within the team as the team moves as a unit up the field. The holistic model consisted of six different principles: built-in instability, self-organizing project teams, overlapping development phases, multi-learning, subtle control and organizational
transfer of learning (Takeuchi and Nonaka, 1986). These principles were then later codified by Jeff Sutherland and Ken Schwaber to tailor fit software development. They also named this new development practice as scrum in honor of Takeuchi and Nonaka’s work and metaphor about rugby. Scrum as a word is originally a rugby term that means the level of required co-work within rugby team in order for the team to succeed and win (Harper, 2012).

Software development can be characterized to be a wicked and complex problem (Schwaber, 2004) (Noll et al., 2014). In other words, due to changing requirements, there is no known solution for a software until the development work is finished. In the case of global software development, where the development work is spread around multiple locations around the world, the situation is even more difficult. These difficulties due to distance, time difference and cultural differences pose challenges for example in communication, differing working habits and different types of relations between managers and software developers (Noll et al., 2014).

Scrum, as well as other agile models, was developed to better organize way-of-work in the organizations to match the difficulties of software development. In order to manage wicked problems, an empirical process control needs to be employed (Schwaber, 2004). Empirical process control is one of the key principles of the scrum model and the theory behind it.

The scrum model (see Figure 2) is based on iterations called sprints, in which a feature team develops new functionality or functionalities. In other words, after each sprint the feature team produces one customer deliverable increment to the final product. After initial sprint planning meeting, where tasks for the iteration per team are decided, the team itself is responsible for producing the product increment for the rest of the sprint (Schwaber, 2004). This means that the team is responsible for designing, implementing and testing the solution to fulfill the requirement set to them in the initial meeting and product backlog. A prime example of empirical process control where the team itself controls the development process with the help of empirical findings discovered during development work. In case of unexpected incidents during a sprint, the team should still report these to a product owner so that he can keep a track of overall status of the product development.
Scrum has only three roles: product owner, team and scrum master. All management responsibilities in a project are shared between these roles. Product owner sets the initial requirements for the product, writes them into backlog and prioritizes them. In practice, this means that a product owner is also responsible for ROI (Return of Investment) and release plans, which product owner can affect, for example, via feature prioritization. The team is autonomously responsible for doing the development work. The role of scrum master is to help and teach team members to work according to scrum principles and company culture and values. Development work is done in sprints that last for 30 consecutive calendar days, or less. After the completion of the sprint, sprint review- and retrospective meetings are held. Resulted functionalities from the sprint are reviewed together with PO and other shareholders in sprint review meeting. In sprint retrospective team discusses together with scrum master about possible improvements to way-of-working practices for the next sprint. After these meetings, next sprint starts. (Schwaber, 2004).

Within the sprint, teams hold a 15min daily scrum meeting every 24hours. In the daily scrum meeting, team aligns and goes through the status of the development items (Schwaber, 2004). Instead of going through what individual team members did yesterday, what they are going to do today and what is blocking them for doing their job, the daily scrum should focus more on sprint backlog and its issues rather than individuals (Downey and Schwaber, 2013). The topics in the meeting could be, for example, what we achieved yesterday on priority 1 and what is our plan for completing priority 1 today? By doing this the group is working more as a team focusing on producing concrete items each day.
2.2.2 Advantages of agile development models

One of the biggest advantages agile methodologies can give to organizations, is the minimization of risks regarding changing requirements during product development. Incremental product improvements via functionalities produced in each sprint make inspection of the product in development easier and faster. The feedback of possible bugs or faults can be then sent faster to development teams and faults can be fixed before other functionalities are affected by these bugs and faults, thus avoiding rework. In addition, involving customers more in product development helps the development teams to maintain focus on customer needs and to get more complete vision of the requirements for the whole product.

However, agile models also have drawbacks and challenges. For example, the minimized documentation in agile models can pose serious problems. These can be seen when new people
are joining the development teams and teams need to consume a lot of other experienced workers’ time for teaching the skills needed for the position. Problems may also arise when employees leave the company with some information about the product that is not shared properly with colleagues (Cho, 2008). While the minimal documentation mentioned in agile values (Highsmith and Fowler, 2001), might not pose any problems for certain organizations, it can become an issue if the required documentation is left undone altogether or done incompletely using agile principles as an excuse. As said, scrum as well as other agile methods rely on the knowledge and skills of people rather than tacit knowledge of plans and documents. However, the methodology also acknowledges the importance of plans and documents (Boehm, 2002). Other challenges are, for example, the involvement of the customer into the development process. Usually customers do not have time to give feedback about the product often enough for the scrum process to work as efficiently as possible (Cho, 2008).

In real life, examples argue for and against agile methods (Boehm, 2002). Figure 3 illustrates risk exposure profiles for plan driven and agile software development methods regarding planning.

![Risk exposure profile](image-url)

**Figure 3.** Risk exposure profile, where P(L) is probability of loss and S(L) size of loss (Boehm, 2002).
In Figure 3, the black line illustrates the risk regarding planning and how it can be affected with additional work. If little architectural long-term planning is done, the risk for major problems, delays and rework is huge. Vice-versa by spending a lot of time planning, against agile principles, this risk can be lowered significantly. At the same time, the red curve depicts the risk of not being able to address changing customer needs fast enough due to plans driven development. In other words, if development is plan driven the risk is that produced software is outdated already by the time when its development is finished.

To summarize, by combining traditional and agile software development methods a sweet spot, where risks are minimized, can be found. With combination of agile and traditional practices, managing and coordination of large organization can be done with the right amount of planning, documentation and guidelines. At the same time, it is equally important to let developers find their own best way of implementing solutions to meet customer requirements. However, it is worth to remember that adoption of agile methodologies, such as scrum, has tripled the number of successful software development projects (Johnson, 2011).

2.3 Scalability of agile models

Most of the agile models and principles are designed to be used in small or medium sized software development organizations and teams (Boehm and Turner, 2005). As agile methodology relies less on prior planning, processes and practices, implementing agile development into large organizations that require more additional coordination and project management can be difficult. Especially inter-team coordination and interactions to other organizational units such as HR, marketing, sales etc. are known issues in implementing large-scale agile (Dikert et. al., 2016). Despite the fact, that agile methods are more difficult to implement in large scale projects (Dyba and Dingsøyr, 2009) agile methods have gained a lot of attraction within larger organizations and projects due to their potential in such environments (Dikert et. al., 2016).

A survey about state of agile, conducted by VersionOne to agile companies, shows that the use of large-scale agile has indeed become more popular in recent years. In 2006, only one third of the survey respondents said that they are working in organizations with 100 persons or more. Whereas in 2015, nearly two thirds of survey respondents said that they are working in
organizations with 100+ people and almost one third of the respondents said that they are working in organizations with more than 20 000 people (VersionOne, 2016).

In 2010 in the International Conference of Agile Software Development (XP2010), conference petitioners were asked, what they see as “the most burning research topic” on the field of agile software development. The most popular answer was large-scale agile and scaling agile methods (Freudenberg and Sharp, 2010). In 2013, in the XP2013 conference, the question was further examined by asking what particular fields regarding large-scale agile were seen as interesting topics. The results of the workshop can be seen from Table 2. In addition to inter-team coordination and handling large organizations, release planning and architecture as well as customer collaboration and agile contracts were mentioned as especially challenging fields.

Table 2. Suggested research topics in the field of large-scale agile (Dingsøyr and Moe, 2013, p.38)

<table>
<thead>
<tr>
<th>Rank</th>
<th>Topic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Inter-team coordination</td>
<td>Coordination of work between teams in large-scale agile development.</td>
</tr>
<tr>
<td>2</td>
<td>Large project organization/portfolio</td>
<td>What are effective organizational structures and collaboration models in</td>
</tr>
<tr>
<td></td>
<td>organization/management</td>
<td>large projects? How to handle a distributed organization?</td>
</tr>
<tr>
<td>3</td>
<td>Release planning and architecture</td>
<td>How are large projects planned? How can the scope be reduced? What is the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>role of architecture in large-scale agile?</td>
</tr>
<tr>
<td>4</td>
<td>Scaling agile practices</td>
<td>Which agile practices scale and which do not? Why and when do agile</td>
</tr>
<tr>
<td></td>
<td></td>
<td>practices scale?</td>
</tr>
<tr>
<td>5</td>
<td>Customer Collaboration</td>
<td>How do product owners and customers collaborate with developers in</td>
</tr>
<tr>
<td></td>
<td></td>
<td>large-scale projects?</td>
</tr>
<tr>
<td>6</td>
<td>Large-scale agile transformation</td>
<td>How can agile practices be adopted efficiently in large projects?</td>
</tr>
<tr>
<td>7</td>
<td>Knowledge sharing and improvement</td>
<td>When is the whiteboard not enough? How can communities of practice be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>established? What measurements are relevant to foster improvement?</td>
</tr>
</tbody>
</table>
Dikert et. al., 2016 review the success factors and challenges of large-scale agile adaptations through systematic literature review. The four biggest challenges identified in this research were difficulties in implementing agile, integrating non-development functions, requirements engineering challenges and resistance to change within organization. Also, inter-team coordination was mentioned as a problem in approximately in one third of the case studies. The challenges found from case studies are largely the same as identified by Dingsøyr et Moe (2013) in their workshop at XP2009. It therefore seems that the specific challenges in adapting large-scale agile are known, but there is no universal solution within agile models to face these challenges.

The success factors that Dikert et. al. (2016) discovered in their literature review also reflected the challenges discovered. The main mentioned success factors were management support, choosing and customizing the agile approach, mindset and alignment and piloting with training and coaching (Dikert et. al., 2016). Based on these results it seems that choosing the correct agile model for the specific company situation is critical. In many cases, companies combined different agile models, such as scrum, XP and lean software development (Dikert et. al., 2016) to match company specific situation and needs. Understanding the principles of the specific chosen model and working according to them is the key to success. Issues often arise when employees do not understand agile practices, and therefore do not want to work accordingly. Piloting with proper training and introduction can reduce the risk of misunderstood agile principles happening drastically. Success factors, such as team autonomy and engaging people (Dikert et. al., 2016), are addressing the problem of inter-team coordination identified by Dingsøyr et Moe (2013). For more detailed analysis of success factors, see rather conclusive work “Challenges and success factors for large-scale agile transformations: A systematic literature review” by Dikert et. al (2016).

Scaling agile is at the moment a "hot topic" amongst researchers and practitioners. The main issues regarding large-scale agile adaptation are known. However, the issues and solutions do
not seem to be universal. This is due to company specific agile models and business situations. Partly because of this, in many cases adapting and combining multiple methods to create a custom fit agile model seems to be fairly popular within large-scale agile adaptations (Dikert et. al., 2016). Nokia LTE R&D selected LeSS or large-scale scrum framework for large-scale agile piloting. LeSS is introduced in more detail in chapter 4.

2.4 Measuring software development

Measuring the performance of software development is important to be able to improve the quality of deliverables, increase the delivery rate and lower costs (Scacchi, 1995). Measurements can be also used to help quantify success (Misra and Omorodion, 2011). However, measuring software development can be difficult due to its complex nature. Often, validity of the results is underemphasized. It might even be that the measurements are not actually measuring desired parameter (Kaner and Bond, 2004). If companies are then led based on measurements with false validation, business plans, forecasts and other causal relationships will not be logical nor will not accord predictions. In other lines of businesses than software development, measuring does not necessarily differ that much from measuring software development. In software development and other businesses managers are interested in seeing the effect on revenue or profit, changes in lead times, development costs, product quality, customer satisfaction and employee satisfaction. However, in microlevel the data collection for software development metrics can be more challenging due to the intangible and complex nature of software.

Software metrics can be classified in many ways. Table 3 shows the metric classification by Misra and Omorodion (2011). Within the scope of this report it is sufficient to investigate two types of metrics and discuss about these: traditional and agile metrics. Even though the agile methods have become hugely popular in past years, the basic metric classes are still the same. The difference between traditional software development metrics and agile metrics is then in the subclasses of these metrics (Misra and Omorodion, 2011). For example, the emphasis on measuring teamwork particularly is missing in most of the traditional software metrics, which in turn is one of the key topics in agile.
Table 3. Classification of software metrics (Misra and Omorodion, 2011, p.2).

<table>
<thead>
<tr>
<th>Metric Class</th>
<th>Attribute they Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product Metrics</td>
<td>Complexity of design (Cyclomatic Complexity—v(G), Extensions to v(G), Knots, Information Flow), Size of final program, lines of code, bang, function points, Halstead’s Product Metrics (Program Vocabulary, Program Length, Program Volume), Quality Metrics (Defect Metrics, Reliability Metrics, Maintainability Metrics)</td>
</tr>
<tr>
<td>Process Metrics</td>
<td>Development time, Type of methodology used, Average level of experience of staff, Performance of development system</td>
</tr>
<tr>
<td>Objective Metrics</td>
<td>Identical values obtained by different observers for a given metric</td>
</tr>
<tr>
<td>Subjective Metrics</td>
<td>Non-identical values obtained by different observers for a given metric</td>
</tr>
<tr>
<td>Resource Metrics</td>
<td>Effort expended on tasks, Extra-project activities, Elapsed time, Computer resources</td>
</tr>
<tr>
<td>Project Metrics</td>
<td>Prediction and control of project size, effort, resources, budgets and schedules.</td>
</tr>
<tr>
<td>Direct Metrics</td>
<td>Length of source code, Duration of testing, Number of defects discovered during testing process, Time a programmer spends on a project</td>
</tr>
<tr>
<td>Indirect Metrics</td>
<td>Programmer productivity, Module defect density, Requirements stability, System spoilage</td>
</tr>
</tbody>
</table>

On the other hand, if the metrics are within same classes both in agile and traditional software development, why is there a need for specific agile metrics? As agile methods are based on delivering small increments of product in short iterations, measuring the performance between each and every iteration can help to identify and tackle problems in the development before they grow too large (Langr, 2007). The dilemma with measurements is that they do tend to affect people’s behavior, and therefore have an especially large effect on the empirically controlled agile development. Work is done so that the metrics looks good, not necessarily so, that the product itself is good (Langr, 2007) (Hartmann and Dymond, 2006). Hence, it is extremely important to consider carefully what to measure in the work of development teams.
For most of the types of metrics presented above, there are multiple variations, definitions and guidelines available in the literature. In 1984, IEEE standards committee gathered a working group to establish process standards for software quality measurements. As software quality is determined by the combination of certain attributes, the IEEE standard was created to define metrics to measure these attributes. These IEEE standard metrics were published in 1992 and then revised in 1998 (ISO/IEC, 1998). The attributes for quality in developing organizations and customers’ point of view are further explained in ISO9126 standard. ISO9126 standard splits quality from the developing organizations point of view to six sub classes: functionality, reliability, usability, efficiency, maintainability and portability. From the customer’s point of view the same kind of split is done to four subclasses: effectiveness, productivity, safety and satisfaction (ISO/IEC, 2011). However, measuring all of the attributes extensively and explicitly is not feasible. Instead, the metrics should be chosen according to company needs. It is important to remember that the used metrics depend entirely on the situation (Misra and Omorodion, 2011). There are also several other definitions for software quality. Most of them, such as Juran’s definition of quality (Juran et al., 1998), either focus on the customer and customer perceived value or on freedom of deficiencies or on both.

2.4.1 Checklist for developing software metrics.

Hartmann and Dymond (2006) have created a checklist for evaluating the potential metrics, which could be used to help determining appropriate metrics. The checklist addresses such issues as importance of metric name, purpose and the basis of measurement for the metric. According to the checklist, the metrics should be named clearly and the focus of measure should be well-defined. Also, the data measured should be specified clearly. This means that the metrics and their dynamics and purpose should be easily and unanimously understandable. It is also good to mention the possible assumptions done to maintain clarity of the metrics. According to the checklist, a good metric also has limits for usage. Use cases for the metric should be clearly specified to ensure that the results of the metrics are always valid. As mentioned earlier, measuring might also have undesired side effect that the work is done so that metrics look good, but the actual quality of the product is not according. When planning the metrics, the designers should not only have an understanding about the results of the metrics, but should also think how the metric can be fooled to prevent the gaming of the metric.
Metrics Checklist by Hartmann and Dymond, 2006, p. 4.

"Name: this should be well chosen to avoid ambiguity, confusion, oversimplification.

Question: it should answer a specific, clear question for a particular role or group. If there are multiple questions, design other metrics.

Basis of Measurement: clearly state what is being measured, including units. Labeling of graph axes must be clear rather than brief.

Assumptions: should be identified to ensure clear understanding of data represented.

Level and Usage: indicate intended usages at various levels of the organization. Indicate limits on usage, if any.

Expected Trend: the designers of the metric should have some idea of what they expect to see happen. Once the metric is proven, document common trends.

When to Use It: what prompted creation or use

When to Stop Using It: when will it outlive its usefulness, become misleading or extra baggage? Design this in from the start.

How to Game It: think through the natural ways people will warp behavior or information to yield more ‘favorable’ outcomes.

Warnings: recommend balancing metrics, limits on use, and dangers of improper use.

According to Downey and Sutherland (2013), it is crucial to have a lightweight yet reliable set of metrics to follow the performance of the scrum teams in order to keep the team performance up and unaffected by the negative effects of heavy measurements slowing down the development. Even though measuring software with applicable metrics can be difficult, the benefits of measuring and guiding the individual scrum teams with metrics are substantial. For example, properly coached and measured scrum teams achieved performance gains of 300-400% over the baseline (1st scrum sprint) compared to the average increase of 35% (Downey and Sutherland, 2013). Downey and Sutherland (2013) also discussed in their paper how the effective management of scrum teams should be done. The solution proposed in their paper was to use story points to quantify size of work to be done to enable objective couching of teams. Story points and their short comings as Nokia pilot indicator are presented in the next chapter 2.4.2.
To summarize, team centricity with emphasis on short delivery iterations in agile methods creates a special need for measuring the performance of the software development teams. Otherwise, traditional software performance metrics and agile metrics seem to be relatively similar. Special attention in software development measurement should be given to determining effective metrics that measure the desired parameter. Metrics should be also set so that the team does not change the way-of-work in a way that it only favors metrics. The metric checklist by Hartmann and Dymond (2006) addresses issues regarding the generation of software metrics.

2.4.2 Key issues in measuring and comparing the status of software development

One of the key issues in measuring the software is how to standardize the size of the feature. Some of the features are more complex and bigger than the others. Therefore, it is natural that it takes more time to complete these than it does to complete some of the easier and smaller features. The commensurability of the features is critical in order to produce viable comparable data. In general, it is believed that the size, cohesion and complexity are linked (Briand et al., 1996). However, there are only few general mathematically viable metrics for size of the software. This is due to the vague definitions about what the metrics are based on (Briand et al., 1996) and what they are actually measuring. Laird and Brennan (2006) split the measurement of software size into two: physical size of software and functional size of software (Laird and Brennan, 2006). Physical metrics that measure, for example, lines of code or LOC need to be carefully defined in order to LOC metrics to be comparable and valid. For example, the re-use ratio of code, the amount of comment lines, the amount of empty lines or the programming language used can have a huge effect on the validity and consistency of LOC metrics if no general rules within a company are set to tackle these issues in measuring LOC. However, the physical size metrics are only valid as a technical metric. They do not measure the value added or functions produced for the customer.

Scrum methodology presents story points as arbitrary metric for scrum feature teams themselves to measure how difficult and how much effort it takes for the team to complete one story (AgileFaq, 2007). One story point can be, for example, one item in the product backlog or a new value-added function for the customer. When team is estimating story points, they should consider the amount of work, complexity of work and risks and uncertainties regarding
the project (Cohn, 2017). However, as story points are defined by individual teams and not by unified standards, the usage of them as a viable commensurate size metric is questionable. It is possible though to set a baseline user story and agree on how many story points it is worth. By using this example user story, teams could adjust their story points to be roughly the same “size”. However, story points are generally intended to estimate the amount of work to be done (Cohn, 2017) and not to visualize how much work has been done. Also, the gamification of story points is relatively easy since they are solely decided by the team. Therefore, the usage of story points as a metric of what and how much work has been done is not necessarily beneficial.

In order to commensurate the size of the feature by its functionalities, function point analysis, first introduced by Allan Albrecht, can be used (Albrecht, 1979). Function point analysis measures the amount of inputs, outputs, interfaces, and databases in the program. By doing so, it is possible to measure the size of the software so that the size metric is comparable to features made by other feature teams. Function point analysis is even defined in ISO20926 standard (ISO/IEC, 2009). However, utilizing function point analysis is not easy. The method is semantically difficult. Analysis has to be done by hand and it requires a lot of effort and practice to master conducting reliable and consistent analysis (Laird and Brennan, 2006).

Complexity of the program is another property that affects the quality of the program in multiple ways. Frederic P. Brooks (1987) distinguishes two different types of complexity: essential complexity and accidental complexity. According to Brooks (1987), the essential complexity is something that is essential for the program due to the difficult and complex problem that has to be solved. Software typically is constructed of many different modules that have multiple ways of interacting with each other. As the size of the software rises, so does the essential complexity. The accidental complexity, according to Brooks (1987), comes from the ‘mistakes’ or bad solutions made by software developers when solving the complex problems. These bad solutions so to speak add unnecessary complexity to the program. The differentiation between essential complexity and accidental complexity should be done when measuring software development. It is natural that during the development of essentially complex programs there are more mistakes made than during the development of less essentially complex programs. Programs should be therefore made commensurate also in terms of essential complexity when comparing to other programs. In function point analysis, the essential or functional complexity
is needed to be estimated to gain reliable view of work required to be done to complete a certain function (IFPUG, 2017). With the size estimate and essential complexity estimate, the effort estimating function point index can be counted.

In essence, function point analysis is measuring the size of the software. It is not measuring development effort used, which is affected by such factors as quality of technical solutions for the feature and a team efficiency. Function point analysis is making development items commensurate in the eyes of customer more than ‘development efforts used’ is. However, creating rules for conducting function point analysis at Nokia requires too much effort for the scope of this thesis. Therefore, applying such rules, enabling and conducting such analysis is suggested as a further research topic.
3 INTRODUCTION TO NOKIA LTE R&D

Nokia Corporation is an IT-company providing telecommunications equipment, managing technology patents and developing new technologies such as virtual reality or digital health services and equipment. Key deliverables of Nokia are IoT technologies and equipment, wireless and wired communication technologies as well as cloud services for intelligent analytics within networks (Nokia, 2017). In contrast to the popular misbelief, Nokia is not currently manufacturing nor designing mobile phones. Current Nokia phones are manufactured and sold under exclusive licensing deal with HMD global Co. (HMD global Co., 2017). Net sales of the whole Nokia corporation in 2016 were 23,9 billion euros (Nokia, 2017), whilst the whole corporation was employing 104 000 people (White, 2017). The roots of Nokia extend to more than 150 years backwards to the year 1865 when Nokia started manufacturing pulp and paper (Nokia, 2017). Over time, Nokia has had many kinds of businesses from Rubber boots to television sets and mobile phones. This shows that Nokia has historically been extremely adaptive and still continues to be. Recently Nokia has acquired Alcatel-Lucent, a large French telecommunications manufacturer, to strengthen its position as one of the leading telecommunications manufacturers in the world. In addition to Alcatel-Lucent, Nokia acquired a smaller company called Withings to gain more footprint in the area of digital health services (Nokia, 2017).

This paper is focusing particularly on Nokia’s Mobile Networks (MN) business line and more particularly on LTE business unit. MN is the biggest business line of Nokia with 64 billion euro revenue in 2016 (Nokia, 2017). Geographically the company is spread across multiple continents (see Figure 4). Gartner (Gartner Inc, 2017) recognized Nokia as the leader in the LTE infrastructure for 6 years in a row during years 2011 to 2016 (Nokiamob, 2017)(Nokia, 2017). Therefore, it is safe to say that mobile networks and LTE business unit is especially important for Nokia as a business.
As mentioned previously, this work is focusing on finding ways to measure the success of WoW change. The need for the WoW change has risen due to the fact that “the LTE market is not anymore growing annually. This means that the customer and internal management needs are changing faster than before. This change also means that the mode of operations has to change.” (Nokia Manager, 2016). Decisions to begin piloting the LeSS WoW model depicts well the mentality of Nokia and the capability to change when needed.

The current WoW in Nokia LTE R&D will be described briefly in the next chapter. Special attention will be given to measuring software development at Nokia. In the following chapter, the new proposed LeSS WoW will be described in more detail.

### 3.1 Way of work in LTE R&D

Nokia LTE business unit is split into R&D, System Design and Architecture (SDA), Verification, Program Management, Business Product Management, Technical Product Management, Portfolio and Business, Operations and R&D and to corporate or other business unit or group functions (Nokia, 2017). The actual development work is mainly done within R&D organization. As seen from Figure 4 that is depicting Nokia’s MN business line sites, the

![Figure 4. MN business line sites (Nokia, 2017).](image)
LTE R&D is also split across multiple continents, countries and sites. System Design and Architecture is responsible for initial requirements analysis of features and system level i.e. cross network element-level specification. Verification department is responsible, for example, for system level testing and verification. Technical product management creates the technical roadmap for the products. Mission statement of the whole LTE business unit is to “define and develop state of the art 4G access systems for frequency division duplexing (FDD) and time division duplexing (TDD) based on Nokia’s Flexi multiradio platform.” (Nokia, 2017). Being flexible, agile and responsive to meet the customer needs better, is also mentioned within the mission statement. In practice, LTE development organization focuses on developing new LTE software releases for Nokia base stations. The LTE WoW itself is based on the Nokia product development processes, which are partially adapted for LTE (Nokia, 2017). For clarity, only valid practices for LTE are presented in this chapter. In short, the LTE development is based on continuous product development model, where product grows incrementally in features in parallel with other products and releases. The goal for LTE R&D is to do agile development in order to be able to respond to the changing customer needs. Hence, LTE product development is thriving to be incremental and continuous. This means that the feature development is started as late as possible and the priority features are developed first to be able to maximize customer value (Nokia, 2016).

As mentioned earlier, LTE product development utilizes continuous product development model. Multiple products and releases are developed in parallel (Nokia, 2016). The releases in development are called system programs and are led by system program managers or SPMs. SPM is responsible for the execution of system programs according to quality, content, cost and timing perspectives (Nokia, 2017). SPMs are working closely with requirement area managers or RAMs who are in charge of technically split requirement areas in which the feature development is done. RAM has the full responsibility over the feature development work (content, quality and scheduling) within their requirement area all the way from specification to verification (Nokia, 2017). Backlogs are used in LTE to monitor the status of software development. All the features are listed in LTE backlogs, where, for example, the priorities, splitting, effort estimation, status and lifecycle of the feature can be seen.
Feature development process starts from the feature screening (FS) process. In the FS process, main focus is to identify and understand business value of a development request for the customer and Nokia (Nokia, 2017). The purpose of FS is to select the most valuable features for fulfilling both customer and business needs. After the features are identified in FS process, they can be prioritized and split to releases by release planning. In this phase, the features are also assigned to corresponding requirement areas. The selected features are assigned to feature owners who then gather feature owner teams (FOT). FOT is a virtual team of representatives of development teams working together to develop a feature or a feature set (Nokia, 2017). The members of FOT consist of people from all functional groups of LTE, such as feature teams, specification, system testers, special entity testing teams, tools, platform, customer documentation, technical product management, etc. In other words, FOT has experts for all development phases from specification process (CFAM) to the system test (Nokia, 2016). Different development sites are often specialized to a certain system component, product platform or development phase. Therefore, FOTs are usually virtual teams working remotely.

FOT process starts by beginning of CFAM process. CFAM or common feature analysis module is a process created for the refinement of the feature and its scope (Nokia, 2017). More precisely, CFAM process produces definition derived sub-features with comprehensive analysis of the new functionalities and external impacts to other system elements. CFAM is done in a CFAM team, which consists end-to-end feature owners from each relevant requirement area (defined in the next paragraph) and feature team. As the FOT is created, the CFAM leader becomes an integral part of the FOT until the completion of the feature’s system verification. (Nokia, 2017). To summarize, features are specified more precisely with requirements in CFAM for the actual programming work to begin.

In order to be able to produce incremental functionalities continuously, features are often needed to be split. System level splits to “marketable sub-features” are done in CFAM process (Nokia, 2016). However, feature splitting is also done in lower levels, such as entity level (base station level) or development level. This is done to fit the feature to one iteration called Feature Build (FB). The continuous software development model of Nokia produces one FB every month. The monthly FBs enable the development work to be done according to scrum sprints.
Feature builds provide a new base station increments for testing phases which are not included in DoD. In the future, feature builds will also be continuously delivered to customers.

Development units are the “heart” of LTE R&D, with about 80% of all employees (Nokia, 2017). LTE development units are organized as scrum teams called feature teams (Nokia, 2017). Feature teams are responsible from base station or entity level specification done in CFAM to entity testing or ET. The scope of the work to be done within FB is further defined in the definition of done or DoD. Feature teams also follow the practice of continuous integration. This means that the team members frequently integrate their work to the automated build, which is then automatically tested for bugs. Advantages of continuous integration are numerous such as early finding of bugs or faults and enabling fixing them early on in the product development. This significantly decreases the risk of rework in product development.

RAM does the allocation of work for feature teams according to the team’s capacity and competence. However, currently feature teams cannot usually do all the required work for the feature (Nokia, 2017). Support from other teams is coordinated in FOT’s. Leading feature team for certain a feature is usually the one that has the biggest development effort. The leading team usually forms the core of FOT and is responsible for doing the entity testing for the feature. It also has overall ownership and responsibility of the feature. Whereas contributing feature teams contribute components to the main feature. These contributing teams are also sending members to FOT of the feature to ensure communication between the teams. Figure 5 represents the relations around FOT.
As can be seen from Figure 5, FOTs are also coordinated within development unit by local product owners or LPOs. LPO’s role is to lead the feature development work of one development unit. LPOs agree together with RAMs and other LPOs which development unit and feature team takes the leading role for a certain feature. LPOs not only coordinate the new feature development work but are also responsible for coordinating entity regression tests for the legacy (old, pre-made) features owned by the LPO or their development unit. (Nokia, 2017).

The feature teams are, according to DoD, responsible for feature development work until the end of entity testing. This means that unit-, system component- and entity testing are all done within FOT and are included into DoD for feature teams. However, additional testing efforts outside feature teams are still required before the feature is ready for shipping to the customers. All the test types mentioned before are done within the eNodeB base station. System testing and verification however has a much wider scope (see Figure 6). System testing and verification comprises of following test areas:

- "Network bring-up and output: integrated composition of network entities
- e2e Functional Verification
• Network performance verification (done partly at System I&V and partly at Entity I&V) ” (Nokia, 2017).

In addition to system test and verification, compatibility testing and interoperability testing are also performed by compatibility management and IOT management (Nokia, 2017).

Figure 6. The scope of system testing and verification (eNodeB equal for base station). (Nokia, 2017).

LTE product development also has several other managerial or administrative departments and roles in order to help the R&D with other managerial tasks such as reporting, measuring, gathering and analyzing data etc. Below is a list of key roles in LTE development. As can be seen from the list, the current setup of R&D has created a need for high amounts of coordination and middle management positions. Pilots for more agile and less governance needing LeSS WoW model are ongoing at Nokia R&D as the organization is trying to shift to more agile WoW.

List of key roles and responsibilities within LTE.

• "Anti-Product Core Team
• Anti-Product Technical Manager
• CFAM Team Leader
• Continuous Trunk Program Manager
• eNB Specification Development Team Leader
• Fault Coordinator
• Feature Owner
• Feature Owner Team
• Feature Team
• FOT Leader
• FOT Member
• Local Product Owner
• LTE R&D Release Manager
• LTE System Specification Project Manager
• Product Owner
• Product Owner Team
• Product Quality Manager
• Program Quality Manager
• Requirement Area Manager
• Release Product Manager
• Release Chief Engineer
• Scrum Master
• SFS Functional Feature Category Leader
• SFS Author And CoAuthors
• SW Core Teams
• System Program Manager
• Task Force Leader
• Technical Manager

(Nokia, 2017).
As can be seen, the list of key roles and responsibilities in LTE is relatively long. Most of these key roles are not directly involved in or are necessary for the value delivery process of developing LTE access systems for Nokia radio platforms. Only a small portion of the roles in the list was required to mention in order to explain the key mission of LTE product development. The existence of different management positions is mainly due to the complex nature of product being developed and due to a huge organization. As the organization and product have become more and more complex and bigger, various management positions have been created along the growth. Component and functional specification of development teams creates even more need for coordination and management.

Another reason for a high number of different management roles is somewhat a reactive management culture. When problem occurs, new position is usually established to handle the problem. As it is typical for software development to encounter new problems and surprises constantly, the number of managers or coordinators for problems also rise constantly. For these reasons, it might be beneficial for LTE R&D to revise the organizational setup and WoW to check if the organizational structure could be simplified.
3.2 Measuring & KPI’s

Figure 7. LTE metrics and use purposes (Nokia, 2014).

Figure 7 summarizes the metrics that LTE is currently utilizing to measure the status of product development. Quality or QA metrics are used to check how much defects are in the code and what is the status of the fix for the fault. At Nokia, fault reports or prontos are used to take a note of faults and as unit for fault metrics i.e. open prontos, closed prontos etc. Many metrics, such as defect density or reduction of new faults, can be derived out of the fault metrics. In addition to LTE metrics, global verification has its own set of system testing and verification related metrics that LTE is also utilizing, for example, in executive reporting.

Work in progress or WiP metrics are visualizing the feature flow in teams, measuring the feature lead time and R&D throughput (Nokia, 2015). WiP metrics help to identify bottlenecks that are slowing down feature development work within the development units. Currently WiP metrics are being used in three levels: system level, entity level and focal point level. System level WiP metrics show the split of undone work in development phases system wide for the mobile network product. Entity level show the same split in eNodeB or base station level. Focal point
WiP shows the amount of work done for features in Focal Point. Focal Point is tool for conducting feature screening process for new features.

R&D executive metrics are metrics that are used to report the progress of software development to stakeholders. These metrics consist, for example, of different types of quality metrics, such as trunk QA and feature QA, quartile metrics and various metrics measuring the status of continuous delivery.

Customer metrics are based on customer feedback. These metrics have a focus in explaining the quality of the product as customer sees it. The data for these metrics is collected through customer advocates.

Backlog metrics are measuring the status of development units and features & releases under development via backlog items. FB exit analysis focuses on measuring the items that have been finished by development units in certain FB, i.e. the items that have been marked as done to the backlog. This enables monitoring of the progress of feature/release development. Site evaluation metrics have been derived from FB exit analysis in order to compare different development sites.

To summarize, Nokia LTE has a lot of metrics in its disposal. However, as a certain R&D manager at Nokia said (2017), “we have a lot of metrics, but only very few systematical use cases for these metrics”. This finding partly supports the fact found from literature as how hard it is to measure software systematically due to the complex nature of software development. Related metrics for measuring LeSS pilot success are further discussed in chapter 6.2.
4 LESS FRAMEWORK

This chapter will examine LeSS framework in more detail. Roles and responsibilities as well as artifacts produced by LeSS are discussed. The new proposed WoW for Nokia based on LeSS framework will be also presented in this chapter.

LeSS or large-scale scrum framework by Bas Vodde and Craig Larman is the selected framework by Nokia to scale agile practices better. LeSS-framework is not trying to invent anything new. As Craig Larman stated (2017), scrum has become popular because “scrum hits an ideal balance between abstract principles and concrete practices”. By bringing in more concrete practices, LeSS tries to achieve the same balance within larger organizations (Larman and Vodde, 2017, p.1). However, the same as in scrum, the focus of LeSS is not to define all the processes and practices exactly. LeSS leaves lots of room for situational learning. As the authors said, ”a less defined process leads to more learning, more with LeSS!” (Larman and Vodde, 2017, p. 2). LeSS promotes itself as a customer centric, transparent continuously improving model per LeSS principles (Larman and Vodde, 2017). One of the key differences between the current agile minded WoW at Nokia LTE is that with LeSS adoption the agile change is also going to happen in product management (Nokia, 2017). Previously the focus has been more in creating continuous integration and test automation infrastructure and in the extension of the DoD for feature teams.

In LeSS, there are two different frameworks depending on the size of the organization: LeSS framework for 2-8 feature teams and LeSS huge framework for 8+ feature teams. This means that LeSS huge framework is suitable for Nokia.

The LeSS huge-framework consists of the following practices and positions (see Figure 8.)

- **Product owner**, who keeps the overview of the whole product’s future vision with the help of product backlog. Product owner also decides the size of the requirement areas and thus is able to prioritize certain parts of products.
- **Area product owner**, who keeps the overview of certain smaller customer concerning area of the product called requirement area. Area product owner uses area product
backlog for monitoring the progress within requirement area but also for prioritizing development items within requirement area.

- **Product owner team**, consisting of product owner and several area product owners to keep the product cohesive.

- **Feature teams**, usually 4-8 within one requirement area. These teams do the actual coding work. Teams are self-organized and they keep a track of their progress in sprint backlogs.

- **Scrum master**, one per 1-3 feature team, who helps feature teams in their never ending learning and adaptation process to Scrum.

- **One shippable product increment**, which is shipped to the customer after each iteration or sprint (Larman and Vodde, 2017). This means that the whole R&D organization has to work in a synchronized way in order to produce one harmonic integrated product increment.

![Figure 8. LeSS huge framework (Larman and Vodde, 2017).](https://www.example.com/figure8.png)

If Nokia LTE decides to continue the shift to organization model proposed by LeSS framework, the organization would not need as many middle management positions. Lean thinking behind LeSS emphasizes and respects people’s ability to learn and readapt to new skills (Larman and Vodde, 2017). According to LeSS, much of the management could be done with empirical
process control within LeSS feature teams. However, as the Figure 8 shows, some additional management roles would still be needed in order to maintain the direction and overall goal of the product development. Product owner would be responsible for the overall Nokia LTE product, whereas area product owners would be responsible for smaller areas of the product split based on customer interests. Table 4 shows the correspondence between roles and responsibilities in current WoW, textbook LeSS WoW and LeSS WoW at Nokia. For simplicity reasons, much of the current WoW roles and positions are left out from the table.

**Table 4. Roles and responsibilities between different way-of-working roles.**

<table>
<thead>
<tr>
<th>Roles and responsibilities in:</th>
<th>“by the book” LeSS</th>
<th>Nokia LeSS WoW vision</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>current WoW (partially)</strong></td>
<td>Head of LTE: Overall responsibility of all work done within product organization</td>
<td>Head of Product: Overall responsibility of all work done within product organization</td>
</tr>
<tr>
<td><strong>Release Product Manager:</strong></td>
<td>Product Owner: In LeSS there is one product owner per product who has the overall responsibility for the product being developed so that it maximizes customer value</td>
<td>Product Owner: In Nokia LeSS there is one product owner for LTE who has the overall responsibility for the product being developed so that it maximizes customer value</td>
</tr>
<tr>
<td><strong>Requirement Area Manager:</strong></td>
<td>Area Product Owner: Is responsible of the overall development work done within functionally split LeSS requirement area. APO Has 4-8 teams assigned to his/her requirement area and directly coordinates the work of teams by acting as their scrum product owner.</td>
<td>Area Product Owner: Is responsible of the overall development work done within functionally split LeSS requirement area. APO Has 4-8 teams assigned to his/her requirement area and directly coordinates the work of teams by acting as their scrum product owner.</td>
</tr>
<tr>
<td><strong>Local Product Owner:</strong></td>
<td>Coordinates development work with RAMs having a focus on one development unit. Development units are currently still system-component specialized. LPO has teams assigned to him/her. He</td>
<td></td>
</tr>
</tbody>
</table>
coordinates the work of those teams by acting as their scrum product owner.

**System Program Manager:**
Has the overall technical responsibility for one system release (compare product release manager)

**Release Product Manager, RAM, LPO:** They cooperate to map functionalities, features and development work between different sites and teams.

**Product Owner Team:** Enables communication between PO and different sub APOs to ensure that the common plans and architecture towards one unite product are followed.

**FOT:** Team of representatives from feature teams, testers and specification people doing work for one feature

**Product Owner Team:** Enables communication between PO and different sub APOs to ensure that the common plans and architecture towards one unite product are followed.

**FOT:** if learning required skill is impossible, then FOT can be used to transfer knowledge

**LeSS Feature Team:** Is e2e responsible of development work from feature specification to testing of customer marketable functionality.

**LeSS Feature Team:** Is e2e responsible of development work from system level specification to system wide testing for customer marketable functionality.

**Technical Product Manager:**
Cooperation with program management to provide technical support

**Technical Product Manager:** Helps APOs and FTs to follow the vision of PO by providing technical support
solutions to releases being developed.

**Scrum Master:** Has usually focused too much on teams without considering product owner, organization and technical practices.

**Scrum Master:** Helps feature teams to work in a scrum way within boundaries set by company culture/practices. Helps in continuous improvement. Scrum master helps also team members to see beyond their individual focus to help maintaining overall product vision.

**Scrum Master:** Helps feature teams to work in a scrum way within boundaries set by company culture/practices. Helps in continuous improvement. Scrum master helps also team members to see beyond their individual focus to help maintaining overall product vision.

**Programs Office:** Produces supportive materials such as cost calculations, schedules, reports etc. for better management of product development.

**Programs Office:** For now, still present to support the management.

**Other Supportive Functions:** Customer documentation, system design & architecture etc.

**Other Supportive Functions:** To be kept as minimal as possible, most of the tasks done within feature teams instead.

**Other Supportive Functions:** Are present, but in a considerably smaller scale than in Nokia’s current WoW.

---

Biggest differences between current Nokia WoW and LeSS WoW is the huge reduction of middle management positions. Many of the management tasks are done automatically within feature teams in LeSS WoW, since teams are given more authority and wider perspective of the work. In other words, feature teams can their selves produce a working piece of product for a customer with the guidance received from PO, APO and TPMs. The idea is that big part of the coordination responsibility for the development work is shifted from management to the feature teams themselves. Because LeSS feature teams are end-to-end capable cross functional and cross component teams, the amount of needed coordination is expected to be dramatically less.
than in the current LTE organization. According to the LeSS-theory, this would render the program or project management offices unnecessary. The feature teams themselves would be either created by moving specialists from different current organizations or/and by learning via continuous development.

Another huge difference is the absence of different system programs or product releases. Adaptation of LeSS framework itself is not the cause to this change. It is more of a subsequence of the shift to continuous (i.e. monthly) delivery model. However, this is the theory in ideal situation. The reality in Nokia is most likely some sort of hybrid solution between releases and continuous delivery due to customer requests and wishes regarding the type of product delivery. In current WoW, releases are released every 6 months. Instead of different product versions, there would only be one Nokia LTE product that would consist of the basic needed functionalities plus extra add-on functionalities that can be added to the base LTE software according to customer wishes.

The pilot is defined as follows in internal Nokia materials “LTE attempts to gradually, and permanently change the way it develops a product” (Nokia, 2017). Therefore, the purpose of the LeSS pilot is not only to “test LeSS WoW” but rather to gradually and permanently change the way LTE develops its products. As the change from old WoW to LeSS WoW happens in steps, there are numerous issues that need to be tackled in order to fuse the two WoW models together in order for LTE to continue developing products effectively during the transformation. This means that FOT is sometimes needed for LeSS teams to communicate with external teams and dependencies. Handling of faults, or prontos, is also one of the issues in gradual adaptation. In other words, which team is responsible for fixes? LeSS teams or some other part of the organization? From the system design and architecture perspective the problem is in finding a way to fuse current SDA organization with LeSS teams. As the specification work is done by feature teams in LeSS model, the proposed solution is to transfer specification experts from SDA to LeSS feature teams as needed. Customer documentation will be also done by utilizing external experts from CuDo organization. Also, the lack of system programs in LeSS WoW requires the organization to find ways of communication between LeSS feature teams and system program managers (Nokia, 2017). However, as noted in table 4, there is no system program managers in Nokia LeSS WoW vision. The communication between SPM’s and LeSS
teams is still needed due to gradual adaptation of LeSS WoW. LeSS is based on inspect and adapt method. This means that as organizational learning takes place, also the WoW practices will be changed accordingly.

This thesis focuses on evaluating the suitability of LeSS huge-framework in Nokia context. The size and complexity of Nokia LTE R&D might pose a need for some additional management in addition to Product Owner and Area Product Owners. Special attention will be given to the analysis of current Nokia metrics and adapting them to team and customer centricity and sprint emphasizing LeSS model to gain data showing whether LeSS improved the organizational efficiency or not. The aim is to yield more information for strategic management within Nokia.
5 RESEARCH METHODS

Chosen research strategy is a single case study. Single case study was selected because the research questions are Nokia specific. Single case study is done to map current situation at Nokia and present Nokia specific suggestions for improvements based on results. Literature review is conducted to map previous research and findings in scaling and measuring agile software development. Internal Nokia material is also utilized. Pilot interviews are conducted with Nokia managers and employees to gain further knowledge of the situation at Nokia. Source code commits to code repositories are also analyzed in order to get a low-level view over what R&D feature teams are actually doing.

The material for literature review was selected based on the searches with selected key words in various databases. Main search engine used was LUT Finna, which searches from multiple databases, such as EBSCO and IEEE. For full list of included databases, see LUT Finna online page at: https://wilma.finna.fi/lut/Browse(Database. Google Scholar was also utilized, but mainly in situations when certain article was not available from LUT Finna. The articles were searched with keywords such as: metrics, measuring, agile, software development, large-scale, transformation, scrum. More than 100 matched articles were first analyzed by their header and after that by abstract and actual content. In total 34 different sources were used for the literature review. Many of the sources were found from appendices of previous articles. The papers selected needed to be generalizable, not specific case studies. The selected search phrases were intentionally loosely defined as purpose of this literature review was to gain more understanding about the topic in general. Many other online sources were also utilized together to gain more extensive view or to get specific information. Online sources are also marked in reference list.

Secondary data collected from Nokia, to build a case study, contains meeting minutes, internal way-of-working guidelines, e-mails and planning documents regarding LeSS pilot as well as “elevator” discussions with Nokia managers and employees. Most of the internal Nokia materials are not available for public, therefore in the thesis they are only referred in text as “(Nokia, *year*)”. Author of this paper has worked at Nokia in project management tasks in the summer 2015 and then later from May 2016 onwards. This means that the secondary data
collection has begun already in summer 2016, as discussions about LeSS pilot were started and the leader of Nokia LTE R&D presented research problem for the thesis.

The survey (see Appendix 1) conducted for the thesis was done to gain developers’ perspective of LeSS adoption. The survey was conducted in Nokia intranet pages using Microsoft SharePoint’s build-in survey module. The questionnaire had 18 questions in total with additional sub-questions. Two versions of the questionnaire were created. One for the non-LeSS feature teams to get baseline and another for LeSS pilot teams. All the respondents were selected from the same development site to ensure that cultural differences are not negatively affecting the reliability of the survey. The version sent to the non-LeSS respondents consisted of 14 questions. Four remaining questions (Question 15-18) were sent only to LeSS-team respondents. These four questions were about opinions of LeSS usage in practice and about the effect LeSS adoption has had. First 14 questions were asked to gain further knowledge about the WoW practices in reality. Questions were formed around the key topics that LeSS framework is proposing to “solve”. Such topics are, for example, knowledge of customer needs amongst developers, lead times, quality and the role of management. Timespan in most of the questions regarding WoW practices was the past 9 months. To gain knowledge of how LeSS adaptation, or possible other reasons during LeSS-pilots, have changed WoW practices additional sub-questions with the same scope but with timespan of 3 months were asked. Survey answers were first codified per question if needed to gain statistical data. Thematic analysis was then done to gain further information behind the statistics. The response percentage amongst LeSS-pilot members was 58% and 34% amongst non-LeSS feature team members. 52 responses were received and analyzed in total. 21 responses were from LeSS pilot team members and 31 responses were from non-LeSS pilot team members.

In addition to WoW survey, additional questionnaire conducted in Nokia LTE by Bas Vodde, one of the co-creators of LeSS-framework, was used to gain more knowledge about the status of product development in general. Questionnaire conducted by Vodde (2016) consisted of two questions; “Name top 3 problems in LTE now” and “Name top 3 mistakes we have done in LTE in the past”. Due to the nature of the questions, the survey respondents and their replies are kept completely anonymous. However, the respondents were selected within LTE organization ranging from different level of positions from software developers to managers.
Thematic analysis was conducted on the questionnaire answers to identify the room for improvement in LTE R&D. Questionnaire answers are not directly presented in the thesis due to business sensitive information in them. Instead, the answers were mostly used by author to gain confirmation to his own findings and deductions of the state of Nokia R&D organization.

Tool called “whodidwhat”, found from Github (Kohvakka, 2017), was used to identify and separate the changes done to the code by LeSS team members. Tool requires a list of version control repositories and a list of usernames. The tool then looks for changes in given repositories done by the given usernames. In addition to list of usernames and version control repositories, dates for the analysis time span need to be specified as well. As an output, the tool produces a statistics file which shows: commit counts and committed lines per user, number of lines of code changed in folders/files and aggregated count of commits to files/folders. Tool also produces blame files, that are exact copies of the original source code files in which the lines changed or added by users on the username list are marked. Some statistics can be derived directly from the number of lines changed in a folder/file. However, more detailed analysis can be made with the blame files. For example, the tool itself is not able to leave out blank lines or lines containing only function start or end marks. By leaving these lines out of the analysis, the results can be unified as the usage of such lines varies from developer to developer. It is also possible to check the type of the code produced from the blame files. For example, it is possible to check was the produced code test case scenario configurations or new functions to the final product. Blame files can be even further utilized as input for static analysis tools such as simian, lizard or sonar. With the username information in blame files, it is possible to compare the quality of the code produced by LeSS teams to the code produced by non-LeSS teams.

In the thesis, all code repositories, from various system components, that LeSS-pilot teams could have changed were analyzed with whodidwhat-tool. During the analysis, the pilot was in its early phases with only four feature builds or iterations completed. Therefore, changes done to outside original system components of the component based LeSS feature teams were too small to be further analyzed and compared with static analysis. Results of the whodidwhat-analysis are explained in chapter 6.3.
6 RESULTS

In this chapter results of the thesis are presented alongside with answers to the research questions defined in chapter 1. One of the main purposes of this thesis is to provide metrics for measuring the success of LeSS-pilots at Nokia. The first research question was posed to address this need.

6.1 RQ1: How to define and measure success factors for LeSS-adoption within a feature team?

As mentioned in the internal Nokia materials, “Main goal of LeSS Requirement Area piloting is verifying whether the LeSS approach will give expected benefits in terms of:

- feature development time from conception to delivery
- overall usefulness of the features from customer perspective
- product quality
- the capability to respond to change
- increased transparency
- employee satisfaction” (Nokia, 2017)

Out of these feature development times, employee satisfaction and quality can be quantifiably measured within organization. Feature usefulness for customer is addressed later with functional quality of the product. Change of transparency within organization is needed to be checked with subjective evaluation. For example, an assumption that less management does lead to increased transparency is not necessarily true in all cases.

Various metrics to follow the status of software development have been proposed by software development community. However, for most of them there are no adequate guidelines on how to interpret the results (Rosenberg, 1998). Often qualifying and interpreting of the metric results is done by hand using past experiences and common sense (Mordal et al., 2012). Therefore, the metrics proposed to measure the success of LeSS are simple and easy to understand. By also making these metrics commensurate size wise and complexity wise, relative improvement with the adoption of LeSS can be observed as the dynamics of the metrics are easily understandable. Reasoning and importance for commensuration is further discussed in chapter 2.4.2.
Metrics proposed in this thesis are a combination of operational and managerial level metrics. As T. Toivola (2017) mentioned, complex organization has different levels of abstractions and each of these abstraction levels have their own requirements for metrics and management. It is important to note as well that the metrics and methods proposed in this thesis are meant especially for verifying if LeSS-pilot has brought the expected benefits to the organization. Some of the metrics are detailed low-level metrics that give accurate feature or even code module level data. Some of them are more high level general management metrics such as customer satisfaction. Metrics suggested in the following chapters are therefore not necessarily suitable for wide-scale use within Nokia LTE R&D. In addition, Hartmann & Dymond’s metric checklist, described in chapter 2.4, will be used to clarify and check the lead time and quality of proposed metrics.

6.1.1 Level of LeSS way-of-work adaptation within LeSS pilot teams.

Pre-requisite for measuring the success of LeSS pilots is to measure to which extent LeSS pilot teams have changed their way-of-working habits. If the pilot teams have not truly changed their way-of-working habits, there is little sense in trying to see the effects that LeSS pilot adoption would bring to organization by measuring such teams. The WoW survey and whodidwhat-tool can be used to see whether the way-of-working habits have truly changed or not. LeSS feature teams should be cross functional cross component teams that do all the feature development phases from SFS to system testing in iterations. In other words, LeSS teams should produce tested product increments in contrast to current LTE feature teams that produce increments to system components.

Whodidwhat-analysis shows reliably the concrete changes that feature teams are doing to different code repositories. Especially in the relatively early phases of the pilot, whodidwhat-tool can be further utilized to measure the level of LeSS WoW adoption in feature teams. The tool inspects the work done in a code level to enable instant and reliable feedback. From the results of the tools it is also possible to see the contributions of the individual team members and to identify what kind of code they have committed e.g. test cases, new functions etc. Whodidwhat-tool is introduced in more detail in chapter 5 Research methods.
In addition to whodidwhat-analysis, WoW survey can be used to get firsthand information from pilot team members about how the way-of-working practices have concretely changed. The survey (see Appendix 1) used in this thesis is focusing on visualizing following important aspects of LeSS WoW and also the changes in them due to the adoption to LeSS WoW:

- Level of understanding of the customer need for the features being developed
- Level of understanding of dependencies between network elements to implement the features
- Level of understanding of dependencies between system components within a base station to implement the features
- The scope of the work done. I.e. system components modified, stages of work such as SFS, EFS, code, testing etc.
- How are the feature decisions (splitting, content) done and how tasks are split within teams?
- How much do team members communicate daily with each other?
- How does the team react to surprises during iteration, such as change in scope, prontos etc.
- Are team members aware of entity testing criteria and who is responsible for doing it?
- Are the teams thriving towards continuous development?
- What do the teams think are the biggest bottlenecks and problems they face?
- What is the work satisfaction of the team members and what do they think about the suitability of LeSS WoW to Nokia LTE R&D.
- What is the effect of adopting LeSS WoW to quality and lead times in team members’ opinion?

The survey conducted for this thesis can be used, slightly modified, later on to analyze the status of LeSS adoption rate in LeSS-pilot teams. The list of important aspects presented above can still be utilized as a checklist, if new questionnaire is created. The results of the survey conducted for this thesis are presented in chapter 6.3.
6.1.2 Development lead time

The development phases that LeSS is affecting in the piloting phase need to be separated and identified to be able to measure the decrease in feature development lead times due to adoption of LeSS WoW. As LeSS pilots are ongoing only in a small part of the organization, the non-LeSS development phases need to be left out when comparing the effect of LeSS WoW on lead times. The definition of done (DoD) for the feature teams during pilot has been set from the beginning of SFS all the way to the end of system testing. Therefore, the feature development lead times comparison should be done from the beginning of SFS to the end of system testing. If features done per LeSS WoW are compared with features done per current Nokia WoW, the change in development lead times can be seen.

This metric is meant primarily for seeing the difference in development lead times between LeSS feature teams and non-LeSS feature teams. However, not all features are the same size or complexity. Therefore, it is logical that for some features the development times are naturally longer than for the others. Features or backlog items are also split differently in LeSS WoW and in current Nokia WoW. In other words, backlog items are needed to be made commensurate for comparison. Function point analysis is suggested to be used to estimate the size and essential complexity of the backlog items. If lead times are adjusted with the function point analysis results by multiplying development time with the number of function points, the backlog items can be compared. The details of making features commensurate are discussed further in chapter 2.4.2. Table 5 summarizes the proposed development lead time metric applied to the Hartmann & Dymond’s (2006) metric checklist. The metric checklist does not go into practicalities regarding how to implement the metric in practice. This is due to current Nokia backlogs and feature splitting practices that make it very difficult, if not impossible, to implement the metric. Solutions to solve these issues are proposed in chapter 7.1.
Table 5. Hartmann & Dymond's checklist applied to function point adjusted LeSS feature team development lead time metric.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function point adjusted LeSS Feature Development Lead Time</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Question</strong></td>
<td>What is the development lead time for a development item between feature development phases assigned to LeSS feature teams in LeSS DoD?</td>
</tr>
<tr>
<td><strong>Basis of Measurement</strong></td>
<td>Measures backlog item development lead time in days from the beginning of SFS until the end of system testing. Development lead time is adjusted with the number of function points for the item.</td>
</tr>
<tr>
<td><strong>Assumptions</strong></td>
<td>LeSS DoD is from the beginning of SFS until the end of system testing. Function point analysis is done according to the set rules and reliably.</td>
</tr>
<tr>
<td><strong>Level and Usage</strong></td>
<td>Suitable for comparing LeSS requirement area lead times to feature lead times of non-LeSS organization.</td>
</tr>
<tr>
<td><strong>Expected Trend</strong></td>
<td>Adjusted lead times should decline after possible initial increase in lead times due to adjustment to new way-of-work. Adjustment in practice means learning about new system components, iterative development and working as a team to solve customer problems instead of producing code for certain system component.</td>
</tr>
<tr>
<td><strong>When to Use It</strong></td>
<td>Verifying the LeSS pilot results in respect of improved development lead times. Comparing development lead times on a backlog item level.</td>
</tr>
<tr>
<td><strong>When to Stop Using It</strong></td>
<td>When the DoD for LeSS teams changes, the metric should also be adapted.</td>
</tr>
<tr>
<td><strong>How to Game It</strong></td>
<td>Conducting false function point analysis, marking the backlog item as 'system test: done' too early. Excessive preliminary work for backlog item before the official start</td>
</tr>
</tbody>
</table>
During pilot verification, it is important to check what phases of development work LeSS teams have actually done in order to have reliable and comparable results.

6.1.3 Usefulness of the product for a customer

There are many different definitions for product quality available in the literature. Joseph M. Juran (1998) splits the meaning of the word ‘quality’ into two critically important aspects regarding management of quality. According to Juran, quality is that product is not only free from deficiencies but also meets customer needs. It is not always possible to measure product usefulness for customer directly in a quantifiable and reliable way. When asked from a customer about the quality of delivered product in retrospective the response is usually someone’s personal opinion. However, it is still the customer’s opinion, which should be taken seriously, since customer is the one that will use the product and decide about potential re-orders. Measures should be taken to improve product even further based on a feedback from a customer. It is also important to let customers know that their feedback has been noted and acted accordingly. In addition to customer feedback, one potential way to monitor how well new functionalities address customer needs is to see how fast and how extensively customer rolls-out developed functionalities and software updates. Chart 1 shows feature activation rates for an example Nokia LTE product release. It shows the percentage of customer base stations where certain feature (features 1-32) is in use and activated.
It can be seen from Chart 1, that only 4 of the 32 features are activated in more than 50% of customer base stations. Majority of the features are activated in less than 30% of customer base stations. Of course, some of the customer required functionalities and features are not necessarily needed in all customer’s base stations. For example, some of the features might be only needed in rural areas to maximize the range of the base station. However, the chart also shows that 19% of the features are not activated at all in base stations. More importantly, the chart shows that 56% of features are activated in less than 5% customer base stations. Low activation rate for the majority of the features in the example release indicates that the feature content of the example release is not accurately responding to true customer needs. In the current setup, it seems that Nokia LTE R&D is spending a lot of resources in developing features that customer does not really need or use. On the other hand, some of the additional and un-needed features might be customer requests due to varying long-term business reasons. This means that the releases take longer to develop and deliver to the customer due to un-needed feature content and customers need to wait for the features that they really need. If product development times would be shorter with releases containing less features, customers could more accurately order features needed in near future instead of preparing for different long-term scenarios with lots of possible features.

As LeSS WoW should make the organization more transparent, the visibility over customer needs should improve and therefore the features should match better with true customer needs.
By following the feature activation rates, the improvement of customer focus with the introduction of LeSS can be monitored accurately on a feature level. However, it is worth to note that feedback loop for activation rates is relatively long. This fact makes it difficult to use such data as a basis for decision making.

6.1.4 Freedom from deficiencies

The second part of the quality according to Juran’s definition is the freedom from deficiencies in the product (Juran et. al., 1998). Defect density is a simple and comparable metric that can be used to measure the relative amount of deficiencies in a code. However, organization needs to have valid rules for determining the size of the object being measured for defect density to be a valid metric. The size could be measured in lines of code or in function points, depending on the situation. However, in addition to defects found, the quality of the code can vary a lot depending on how it was written. For example, poorly written code might be defect free, but it might be hard to maintain and update the code without inserting additional faults. Two different kinds of quality metrics and analyses are proposed below to measure the change in quality: functional defect density metric and static analysis to inspect the overall quality of source code.

Existing Nokia pronto system could be utilized for gathering the fault data for the functional defect density metric. Prontos are essentially faults found either from testing or by customer. Utilizing such fault data has the benefit that only faults that are visible or will be visible to the customer in a normal operation of the product are compared (Musa, 1989). In other words, this fault data can be used in measuring functional, customer perceived, quality and defect density. However, inadequate test cases can have a huge negative effect on the reliability of this metric. Also, mapping of failures of certain features can be sometimes difficult due to test methods used. For example, if a failure is found in system testing, identification of the source of fault in feature level might be sometimes impossible. Practices used to write down the pronto and pronto specification and details could be also beneficial to revise. Currently, most of the fields in the pronto tools such as “Feature ID” or “Found In” can be filled freely by hand. This causes inconsistency in the pronto data and makes it harder to use the prontos as a reliable feature level fault metric.
As a manager at Nokia said (2017), one of the shortcomings of the R&D is that the entity testing for the features is not done on time. This means that failures that would be discovered in entity testing are not discovered, marked as prontos and fixed on time. In other words, untested features are inserted into trunk, possibly with faults in them. In addition, long feedback time from customers due to gradual upgrade of the base stations with new software and features poses challenges in the usage of the functional fault density metric since not all functional faults are visible at the metrics always. Due to the limitations mentioned above, the functional pronto metrics are mostly valid for high abstraction levels such as functional LeSS requirement area. The prontos can be assigned to requirement areas using ‘pronto group’. It has been agreed in the LeSS pilot agreements that LeSS requirement area is responsible for fixing the prontos related to LeSS requirement area. ‘Group in charge’ could be used to identify which team has fixed the pronto. After the pronto has been finished the team or group that has been marked as ‘group in charge’ should be the one that has fixed the pronto. Similarly, it is possible to identify the prontos that have been caused and fixed by LeSS requirement area.

Table 6. Hartmann & Dymond's checklist applied to functional defect density per requirement area metric.

<table>
<thead>
<tr>
<th>Name</th>
<th>Functional Defect Density per requirement area metric.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Question</td>
<td>What is the density of test or customer discovered faults that have been found from trunk or product releases related to LeSS requirement area per release?</td>
</tr>
<tr>
<td>Basis of Measurement</td>
<td>Measures the number of bugs or functional faults found from trunk or product release. Only prontos related to features done by LeSS requirement area are being measured. The size of the development item is measured in function points. The unit of metric is defects/function points.</td>
</tr>
<tr>
<td>Assumptions</td>
<td>All the testing phases are done with sufficient coverage. Customer reports found faults back to Nokia. All the faults discovered are written down as prontos and assigned to a correct requirement area.</td>
</tr>
<tr>
<td><strong>Level and Usage</strong></td>
<td>Suitable for measuring LeSS requirement area functional defect density and comparing it to the level of defect density to other requirement areas, LeSS or non-LeSS.</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Expected Trend</strong></td>
<td>Functional defect density should decrease as the pilot grows and proceeds.</td>
</tr>
<tr>
<td><strong>When to Use It</strong></td>
<td>Metric should be used after the release containing the functionalities being measured is released and customer has done acceptance testing to it. This will ensure that most of the defects that are found are visible. Metric can also be used during development to gain trend information of the level of defect density. Comparison is feasible also between requirement areas.</td>
</tr>
<tr>
<td><strong>When to Stop Using It</strong></td>
<td>-</td>
</tr>
<tr>
<td><strong>How to Game It</strong></td>
<td>Conducting unreliable function point analysis. Not reporting all the prontos discovered.</td>
</tr>
<tr>
<td><strong>Warnings</strong></td>
<td>Accuracy is affected negatively by assigning the prontos to requirement areas and by late discovery of the prontos. For some faults finding the root cause can be difficult. The prontos that are discovered late, for example years after the release was delivered, are not taken into the metric.</td>
</tr>
</tbody>
</table>

The faults that are written down as prontos are either found by testing or by customer. However, factors such as code complexity or duplexity greatly affect the overall code quality as well. For example, less complex code is in general easier to maintain, update and modify without inserting additional faults to the code. Static analysis could be used to further inspect the change in code quality by monitoring the trends in source code complexity, duplexity and a number of semantic faults. As said, these are all attributes that affect greatly the overall quality of the code and, therefore, indicate whether the pilot has led to the improvement of the code quality or not.

In order to mitigate the challenges and limitations regarding the usage of functional defect density metric presented before, static analysis could be used in addition, not solely. From the
results of static analysis, the changes in quality can be seen quicker and more precisely than from pronto data. Static analysis tools are also practically impossible to fool or game and bad or inadequate test cases cannot affect the reliability of the analysis unlike in functional tests. On the other hand, static analysis cannot find functional faults since static analysis is only focusing on finding anomalies and analyzing the structure of the code. Testing, whether the program has desired functionalities or not is not feasible by using static analysis methods. The results of static analysis can be used straight away. With static analysis, software developers can improve their working methods and fix errors immediately. Whereas executives can monitor the technical quality of the source code itself almost in real-time. In order to gain the correct data for static analysis to see if the pilot has made any difference to physical code quality, Whodidwhat tool can be utilized to separate the changes done to code repositories by LeSS-pilot teams members. See chapter 5 Research methods for further details of the usage of whodidwhat-tool. One possibility would be to compare source code file complexity and duplexity without LeSS team contributions and with LeSS team contributions. By conducting such analysis, the change in complexity or duplexity can be isolated due to changes done by multicomponent LeSS teams.

To summarize, functional defect density metrics are more “agile” in a way that they focus more on customer perceived quality than on the quality of source code. On the other hand, getting a feedback from these metrics takes time. Long feedback times decrease the suitability of these metrics for agile way-of-working methods that are focusing on continuous improvement and continuous integration based on quick feedback. With static analysis, it is the exact opposite. The quality data can be collected quickly and effortlessly, but it does not represent the true customer perceived quality. By combining the two metrics, a balance could be achieved by utilizing the instant feedback of static analysis in management in continuous integration and improvement, and also by utilizing functional defect density metrics to maintain a focus in the customer and customer values.

6.1.5 Employee satisfaction

For measuring employee satisfaction regarding LeSS pilot, simple questionnaire can be utilized. Conducting regular and brief employee satisfaction surveys is a convenient way to gain more
knowledge about the opinions of the employees and their work satisfaction. By showing employees that concrete measures are taken to improve the situation based on their answers, the motivation to participate in future surveys can be increased and the effectiveness of the surveys can be improved significantly (Heryati, 2017). Employee satisfaction was a part of the survey conducted for the thesis. Results of the survey, including employee satisfaction, are presented in chapter 6.3. When conducting employee surveys in a global company, such as Nokia, the effect of cultural differences has to be considered when the results of the survey are analyzed.

6.2 RQ2: Which of the current Nokia metrics should be used to define success of the LeSS-pilot?

The aim of the second research question was to investigate whether current Nokia metrics could be used to define the success of the LeSS-pilot. By utilizing current Nokia metrics, many benefits could be achieved. These benefits could be, for example, possibility to compare pilot results to history data effortlessly and the possibility to use data collection methods that are already in place.

As discussed earlier, Nokia has already implemented metrics to measure feature development times and the number of failures in LTE R&D. Feature development times can be derived from WiP or work in progress metrics, feature build exit analysis or directly from LTE metrics data repository or MDR where all the LTE metrics data is stored. There are marked timestamps for the beginning and end of certain development phases within the MDR. For example, the start time of entity level feature specification and end time of system verification are marked in the MDR. By using the timestamp information to measure the effect on development times, it is possible to measure only the changes that happened in feature development work within LeSS DoD. This means that in the current hybrid model with old WoW and LeSS pilots, LeSS feature team starts work at the beginning of SFS and passes on the features after they are system tested and verified. Therefore, it is relevant to compare only this time period of the feature development work. Comparison can be conveniently done using the timestamp information in the MDR.
Nokia LTE R&D is already utilizing fault density metrics. More specifically the metric is faults per development effort. Fault density as a metric is a suitable pilot success indicator. However, size data for the metric is currently a development effort used for item. As discussed earlier, development effort used is not a valid way of measuring the size of the feature. It is at least not the valid way when the fault density metric is used to see whether the change in way-of-working has shifted quality to better or not. If the size would be measured in a different manner, for example, with functions points, as suggested in the previous chapter, the validity of the metric as a pilot success indicator would improve. As mentioned before, fault data for the new defect density metrics could be gathered from testing and observed failures. In addition, static analysis or similar inspection done to a code can be utilized to get faster feedback of the change in quality.

One of the findings during the research was that the measuring and reporting practices have changed relatively often. For example, within certain product functionality developed in multiple releases between 2014 and 2017, the reporting and measuring practices changed significantly. The feature in question was split differently to sub-features between different development organizations such as system design and architecture and integration and verification. This kind of excessive feature splitting reduces visibility significantly and makes measuring of the status of development systematically impossible. This ineffective practice has been since identified and some fixes have been implemented. However, the issue has not been completely removed with these fixes and it still persist in a smaller scale.

Nokia has also implemented many different milestone systems to track the status of LTE business line. Sheer number of different overlapping process milestones makes the visibility over product development even more complex and hard to understand. The usage, definitions and relations between different milestones can lead to a situation where milestones start causing confusion in readers of product backlogs, where the checkpoint statuses and milestones are stored. For example, Nokia CFAM process include both system level feature specification and entity or eNodeB level feature specification. CFAM checkpoint 3 should indicate the end of entity level feature specification. However, usually the CFAM status is not in harmony with the entity level feature specification status within all different backlog files used at LTE R&D. This mismatching or confusing data was one of the reasons why current Nokia metrics and statuses
in backlogs were not used to measure and compare the results of LeSS pilots against the old WoW.

In addition to shorter lead times, adoption of LeSS WoW was expected to improve product quality. Pronto metrics were planned to be utilized in the thesis to check the effect on quality. This plan turned out to be not feasible due to couple of reasons. One reason was that the identification of origin of fault from Pronto is not always possible. Also, the identification of product development or testing phase in which the fault was discovered is not clear. This poses problems in reliable comparison of fault numbers. Also, many of the faults are found only after customer pilots. In some cases, faults are found only a year after the end of actual coding work. As pronto tool is used by humans and most of the sections, such as the ones regarding origin and the phase in which fault was discovered, are free text boxes, building a reliable statistical comparison proved to be very difficult if not impossible.

Other significant issue related to reuse of Nokia metrics was the commensurability of the features. The start of LeSS pilot was postponed and only limited number of sub-features were done during the later parts of the thesis project. Therefore, it was decided not to start gathering statistical metrics to figure out the success of LeSS pilots. Without large enough sample of features, using the averages for lead times and number of faults the features cannot be averaged to compare with other features created in LTE R&D. Option of commensuration for features was also discussed but it would have required too much effort in order to be done within the thesis project. However, suggestions for metrics measuring the success of LeSS-pilots after the thesis without the time or resource restrictions of thesis are given in the results paragraph.

To summarize, Nokia has already implemented the same metrics that could be used to measure the success of LeSS-pilot. However, current metrics and data collection methods and principles need to be modified in order to get accurate, comparable and valid results to determine the status of the LeSS pilot. Currently, the metrics are used in a higher level, such as development site, product release level etc., than they would be especially during the first months of the LeSS pilots. This difference requires adaptation also to a level of accuracy of the data used for metrics. Lower level metrics require more detailed and accurate lower level data.
6.3 RQ3: What are the preliminary results and implications of Nokia’s LeSS-pilot?

As mentioned before, it is too early to derive concrete reliable and measurable results due to status of pilots. However, LeSS-adoption rate and software developers’ subjective opinion about LeSS was evaluated using the questionnaire and whodidwhat-tool. In this chapter, the results of conducted analysis and survey are presented. The implications of the results are discussed further in chapter 7 Discussion.

6.3.1 Whodidwhat-analysis results

Whodidwhat-tool (see chapter 5) was used to identify the changes that multicomponent LeSS feature teams had done to the relevant code repositories of different system components. Based on the results of the whodidwhat analysis, LeSS feature teams are slowly starting to shift from component specialized teams to multicomponent LeSS-teams (see Table 7 and Table 8). However, LeSS feature teams are still mostly working on their own specialized system component. In the first four iterations, one of the four LeSS pilot teams did not expand its scope outside of team’s specialized system component. Teams 2, 3 and 4 expanded their scope to one additional system component. When measured by LOC, team 2 contributed to the additional system component 6% of the total work done by team 2 (excluding ET testing) in first four iterations of LeSS-pilot. The same percentage for team 3 is 12% and for team 4 it is 8%. Results also show that teams 2, 3 and 4 started doing entity testing within teams. Before LeSS pilots, only team 2 had done entity testing contributions but in a significantly lower scale than after adopting LeSS WoW. In team 4, in which the increase and absolute number of entity testing lines committed is the highest, entity testing was committed by special entity testers transferred into the team in the beginning of LeSS pilots. However, according to these entity testers, the team was responsible for creating the test cases as well as executing and analyzing the results. This would indicate that building new teams by allocating specialists such as ET or SDA specialists into feature teams would be a faster way to adapt LeSS WoW than current way of learning the new competencies without additional expert team members.
Table 7. Pre-pilot whodidwhat-analysis results.

<table>
<thead>
<tr>
<th>Nr. of system components modified</th>
<th>Aggregated count of files committed to team specialized system component</th>
<th>Lines of code committed to team specialized system component</th>
<th>Aggregated count of files committed to other system components</th>
<th>Lines of code committed to other system components</th>
<th>Aggregated count of files committed to entity testing repositories</th>
<th>Lines of code committed to entity testing repositories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team 1</td>
<td>1</td>
<td>1031</td>
<td>4178</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Team 2</td>
<td>1</td>
<td>3795</td>
<td>40673</td>
<td>0</td>
<td>0</td>
<td>480</td>
</tr>
<tr>
<td>Team 3</td>
<td>1</td>
<td>2966</td>
<td>15833</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Team 4</td>
<td>1</td>
<td>436</td>
<td>3796</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

When team contributions to other system components are inspected in more detail, it is visible that a big part of the contributions is creation of test cases or configuration of test drivers. As new functional contributions of pilot teams to other system components were still relatively low, reliable comparison to code quality with static analysis methods was not reasonable to conduct. However, whodidwhat-analysis enabled accurate inspection of what kind of work feature teams had done in the first four iterations of the LeSS-pilot.

6.3.2 Way-of-working survey results

Way-of-working survey (see Appendix 1) was conducted to monitor the status of feature teams and effect of adopting LeSS WoW. The main results of the survey are presented below. Sign “-“ means that a respondent has not answered to the related question.
As can be seen from Chart 3, most of the LeSS team members still have competencies and understanding to modify code for only one specific system component. Only 14% of LeSS team members replied that they have code level understanding of at least two system components of the eNodeB or base station. None of the respondents replied that their level of understanding was adequate to make changes to all system components that participate in the feature that is currently being developed. This indicates that LeSS feature teams have not yet gained competencies to truly be multicomponent feature teams as defined in LeSS framework. The results of whodidwhat-analysis are corresponding to this finding. However, 48% of the respondents answered that their base station level understanding has improved during the first three iterations of LeSS pilots. Main reason for the improvement was the introduction of either MAC PS or ULPHY as an additional system component that team members could inspect or make modification to and thus gain more knowledge of other system components. However, 48% of non-LeSS team members reported also improvement in the base station level knowledge during the same time period. Most of the reasons for the improvement in base station level of knowledge amongst non-LeSS team members were related to gained product development experience in general. However, it can be assumed that LeSS pilot teams have also gained more experience during the timeframe in addition to learning by inspecting and modifying the additional system component (see Chart 2). Therefore, we can say that LeSS pilot has slightly increased base station level of knowledge of the pilot members.
From the results of the survey it is visible that LeSS pilot teams expanded their scope of development work to new types of development work. The expansion of scope was due to adopting parts of entity testing work. Chart 4 and Chart 5 show the percentage of LeSS-team members who had done ET before and during the pilot. As can be seen from the charts, only 29% of the LeSS-team members had done ET work before the pilot, whereas during the first three iterations of the pilot the same percentage was significantly higher reaching 52%. Increased ET work was also visible in whodidwhat-analysis results. From the results of non-LeSS team members, it is visible that there has not been any extension in the scope of the work. Based on the survey results, members of non-LeSS teams are still mainly doing either coding or system component testing work or both.

**Chart 3.** LeSS-team members’ base station level of understanding.
Results of the survey also suggest that LeSS-team members have a considerably higher visibility over entity test cases and acceptance criteria than non-LeSS team members. 76% of the LeSS-team members (see Chart 6 and Chart 7) replied that they have seen either entity test cases or acceptance criteria of entity tests or both. Correspondingly, only 37% of non-LeSS team members replied that they have visibility to entity test cases or acceptance criteria or to both.

Chart 4. LeSS-team members’ contribution to entity testing before LeSS-pilot.

Chart 5. LeSS-team members’ contribution to entity testing after first three iterations of LeSS-pilots.

Chart 6. non-LeSS team members’ visibility to ET cases and criteria.

Chart 7. LeSS-team members’ visibility to ET cases and criteria.
Chart 8 and Chart 9 show what both LeSS-teams and non-LeSS teams think about the responsibility over the entity testing work. As can be seen from the charts, LeSS teams have in general wider knowledge and bigger rate of commitment to entity testing work than non-LeSS teams. Reasons for high rate of empty responses from non-LeSS team members both to visibility and to responsibility of the ET work is not clear. The reasons could be that respondents do not know the answer or did not want to answer. Based on the responses of non-LeSS team members, ET testing has been done by separate entity testers. As with the introduction of LeSS WoW, LeSS pilot teams are currently trying to do the ET work within teams in oppose to utilizing separate specialized entity testers.

Similarly, LeSS teams have changed feature decision process (i.e. deciding content of features and feature splitting). Before the beginning of pilot, non-LeSS teams and LeSS teams reported that product owner and technical leader with architectures was in charge of deciding user scenarios and features. Teams could then take and in some cases do necessary sub-splits for features in grooming meetings in which the goal is to populate backlog for teams for the next iteration. After first three iterations of LeSS pilot, LeSS teams have more responsibility over feature decision process. According to set DoD for LeSS teams, teams should be responsible for feature development work from SFS until the end of system testing. Even though LeSS
teams started splitting features in e2e perspective themselves, they are currently not doing the feature SFS work themselves. This is partly due to the fact that there are no system architects in the teams to help with feature specification process. However, introduction of backlog refinement meetings in which the team together does e2e splits for features has increased significantly the amount of communication within LeSS teams. Another reason for increased communication is increased pair work within teams. In fact, 62% of LeSS team members who answered to the survey reported that there has been an increase in the amount of communication within the team. Chart 10 depicts the amount of communication in everyday work within LeSS pilot teams before LeSS pilots and the relative change (i.e. more, less or no change) in the amount of communication after first three iterations of LeSS pilots. The baseline non-LeSS teams reported higher amounts of communication within teams. However, there was no relative change in the results of non-LeSS teams during the first three iterations of the LeSS pilot whereas the communication within LeSS teams had increased.

![Chart 10](chart10.png)

**Chart 10.** Amount of communication between LeSS team members and relative change during first three iterations of LeSS pilots.

Biggest bottlenecks that both non-LeSS teams and LeSS teams reported were lack of system level knowledge and dependencies to other system components that cause delays. In addition, work done to fix prontos is seen as a nuisance as it is interrupting the development work and is
very time consuming. Some of the team members also replied that there are too many meetings interrupting development work. Average work satisfaction for LeSS team members was 3.8 on a scale from 1 to 5. For non-LeSS team members, the average work satisfaction was 4.3. On the other hand, 43% of LeSS team members said that their work satisfaction had risen during the first three iterations of LeSS pilot, whereas only 18% of non-LeSS team members replied that their work satisfaction had risen during the same time period. The main reason for the improvement in work satisfaction for LeSS team members was ability to learn more via increased communications with team mates. Also, learning from different types of development work and from different system components was seen as a positive change. Even though the change in work satisfaction was positive, one LeSS team member answered that his work satisfaction had decreased due to excessive communication in LeSS WoW model. To summarize, it seems that adopting LeSS WoW had improved work satisfaction in most of the cases.

Chart 11 shows the opinions of LeSS-team members on how lead times have changed during the first three iterations of LeSS-pilots. 48% of the respondents said that development times have gotten longer. The adaptation to new WoW and learning how to modify the other system components took and will take time. Also, additional unit tests were mentioned as reason for increased lead times. On the other hand, 29% of the respondents said that lead times got shorter. The reasons mentioned were lack of dependency to other system components and possibility to deliver e2e features within feature a team.
Chart 11. Change in lead times according to LeSS-team members.

Chart 12 shows the opinions of LeSS team members on how quality has changed in the same timeframe within teams. In general, quality got better according to team members. Main reasons for the improvement of quality was that team members understood more the functions of code due to e2e feature delivery model, cross-component work and increased communications. Moreover, the fact that items were split to entity testable sub items helped to improve quality hence it was possible to easily do entity testing to all development items.

Chart 12. Change in quality according to LeSS team members.
To summarize, it seems that LeSS WoW could be suitable for Nokia LTE R&D teams. As can be seen from Chart 13, 43% of LeSS pilot team members say that LeSS WoW is suitable for Nokia LTE R&D. However, many of the respondents emphasized that it is still too early to say whether LeSS WoW is suitable or not. Change to new WoW requires time as it makes the work more challenging as there is e2e scope for the work. On the other hand, such WoW fosters learning and pair work according to pilot team members, thus making work more enjoyable and efficient.

**Chart 13.** Suitability of LeSS-wow to Nokia LTE R&D according to LeSS team members.
7 DISCUSSION

Discussion part of the thesis is split into two different sub-chapters. First one, chapter 7.1, presents suggestions for management to improve measuring practices at Nokia, both for general cases and for measuring the success of LeSS pilots. The goal of the suggestions is to improve management visibility over product development thus make decision making process easier. Suggestions focus especially on enabling the use of proposed LeSS pilot success indicator metrics and on improving the measuring practices in general. Second part, chapter 7.2, summarizes and evaluates the results and implications of whodidwhat-analysis and WoW survey conducted for the thesis.

7.1 Suggestions for management regarding measuring the software development and LeSS pilots.

Commensuration of features is not always simple. However, creation of well-defined systematic rules for measuring both size and complexity of the software would be highly beneficial for Nokia LTE R&D. In order to have comparable data for feature development work, size and complexity should be considered in most of the metrics. Currently size and complexity standardization is not done sufficiently in LTE R&D. Features are made commensurate using efforts or man-hours spent for developing the feature. Man-hours spent is not a viable metric for commensuration of complexity or size as it does not take in account effectiveness or difference between essential and accidental complexity. One better commensuration could be, for example, function points. The benefit of function point analysis is that it is also considering functional complexity in addition to the size. By taking into the account functional complexity, commensuration of the features can be done even more accurately (Lavazza and Robiolo, 2011). Also, function point analysis is not tied to the performance of development teams.

However, for getting fast results from pilot success KPIs, a comparison feature or feature set could be used as a baseline instead of using commensuration methods such as function points. By selecting roughly same size and complex features as a baseline, the comparison of the improvement due to LeSS adaptation can be seen. However, this is not a solution that works for the whole LTE, but more of a quick fix to a special problem. Therefore, the commensuration of the features should be done none the less. Features made in LeSS requirement areas could
be, for example, compared to features made within FOT or group of FOTs. For both FOT and LeSS teams the scope of the work extends from feature specification to system testing.

In general, current metrics data looks viable over a large set of features. When inspected more closely on a feature and feature split level, various issues in metric reliability are visible in addition to non-commensurate features. One of the issues is that splitting of the features is significantly reducing visibility and accuracy. Currently, backlogs in LTE R&D are split between development ‘phases’. Features are often split differently in the system specification backlog and in the system testing and verification backlog. This means that automatic comparison of feature sub-splits between system level specification and -testing is not possible as the sub-splits are not matching between two backlogs. When the data from multiple backlogs is gathered to one single backlog file, i.e. LTE Data Repository, the data seems to be incomplete as system verification is missing for certain feature sub-splits due to different splitting of the features in specification and system verification backlogs.

The principle of having only one product backlog per requirement area seems to be sensible, as it would bring more clarity and help in the overall planning and monitoring of feature development work. If only one product backlog is in use, organization could work in a more harmonized and synchronized way even if features are required to be split into many small functional pieces. LeSS framework suggests that feature splitting should be done as minimally as possible and only with functional splits producing backlog items that can be completed within one sprint (Larman and Vodde, 2017). Doing only functional splits instead of special splits within different development phases would improve e2e visibility on its own and enable the use of single product backlog. This would further increase visibility over product development for everybody in the organization. It would also enable accurate and immediate following of development work progress for individual feature splits, for example, via timestamp information used currently in LTE Metrics Data Repository. As mentioned before, current Nokia metrics provide data that is valid only in feature and higher abstraction levels. By changing feature splitting practices to functional splits and shifting to one product backlog, as suggested also by LeSS framework, metrics could be made accurate in feature sub-split level as well.
To further increase the reliability of metrics used, a simple and general set of rules should be created for defining new metrics and for using these metrics. As mentioned earlier, too many measurements slow down the development work. Therefore, Nokia should think carefully what metrics to collect. It would be better to have less systematically utilized metrics than a huge amount of metrics with no systematic use. The metrics checklist by Hartmann and Dymond (2006) can be used to define rules for the metrics. As said before, complex nature of software development poses great challenges for measuring software development. It is important to assess when and for what purposes metric is used so that right data is collected to produce reliable overview. If the dynamics behind a metric are clear, management can set meaningful goals for software development teams. Otherwise, if goals are set unclearly, for example, to “reduce the number of open prontos by X percentage in 2 months”, there is a risk that metrics will only start to foster competition between teams. If a goal for a team is to reduce number of open prontos, the easiest option for a team is to transfer the responsibility of pronto to other teams or reduce test coverage to discover less new prontos. Even if the situation regarding bugs inserted in to the code has not truly improved at all, it will look it has in the team metrics that are measuring the number of open prontos. In addition, even though high management is driving agile change, lower management and development teams are usually afraid to make changes if the changes will first cause a drop in performance and thus cause additional work to teams.

The same can be said for LeSS pilot success indicators. Most important is to understand the dynamics behind the metrics. As mentioned before, metrics for monitoring the success of the pilots should vary between different levels of abstractions. Since the pilot is currently affecting only a small part of the organization, metrics to measure the effect of pilot can only focus on measuring the same small part of the organization. Therefore, inspection of source code properties (complexity and duplexity etc.) are currently viable statistics for the management to see and more importantly to understand the effects of pilots to the quality. As the scope of the pilot grows, the scope of the statistics that management follows should grow as well and move to more higher levels of abstraction. When the scope of the pilot is large enough to see the causal relation between LeSS WoW and overall functional quality of a product, such metrics as functional defect density or customer satisfaction can be utilized. For this same reason, the metrics used to measure the success of LeSS pilots are not necessarily directly suitable for
measuring the same attributes in a bigger scale within Nokia LTE R&D. Proposed metrics are further introduced in chapter 6.

One other issue that was discovered during the thesis process was the lack of commitment to agile transformation and practices. Based on discussions at Nokia, the issue is a lack of commitment to agile WoW. If the whole organization is not willing and trying their best to adapt to LeSS WoW, the change to a new WoW is extremely difficult if not impossible. Current WoW is focusing on shortening release cycles. However, it seems that organization is not yet ready for such transformation. This partly has made people even more skeptical and afraid of agile methods, such as LeSS. It seems that Nokia LTE R&D would need to radically change its WoW to be able to shift from current bi-yearly release model to a more agile continuous delivery model. However, the WoW cannot change if people in the organization are not willing to change. Also, for this reason metrics should not be used to foster competition in the organization. One of the biggest fears of conducting such change is deterioration of customer relations due to possibly reduced production performance during the transformation. Hence, it would be critical for Nokia to start including customer to development process even more strongly. As can be seen from feature activation rates (see Chart 1) there is a lot of room for improvement in responding to true customer needs, which is one of the main principles of agile methodologies.

Challenge in whole Nokia LTE R&D is that entity testing of the features is not done on time. This greatly compromises the quality of product during development as well as reliability of pronto data. The functional splitting of features in addition to using only one backlog would improve visibility over entity testing significantly, thus removing the need for separate management and controlling of entity testing practices. In the whole Nokia R&D context, if entity testing practices are left un-improved, using static analysis or other code inspection instead is just a way to heal the symptoms so to speak. The real improvement in quality and testing should be an increase in extensive on-time testing per DoD from unit testing to system testing. A problem is that if feature teams only perform system component test or module tests before committing changes to trunk, there is a risk that changes to trunk break the whole base station level build. Moreover, with the analysis of LeSS-pilot’s results, special attention should
be given for an inspection of test plans for features as well as the realization of these plans to make sure that teams test product extensively enough.

7.2 Main implications of the whodidwhat- and wow-survey analysis.

Based on the results of whodidwhat-analysis, way-of-working survey and the feedback from LeSS pilot area product owner, Nokia LeSS pilot is still in its early phase. From whodidwhat-analysis it was visible that LeSS teams had not yet truly changed to multicomponent teams. Only little changes to one additional system component outside of teams’ special knowledge was done. The findings from the way-of-working survey supported this finding from the whodidwhat-analysis. According to survey results, LeSS pilot teams were not yet able to do feature development according to Nokia LeSS DoD. Even though teams had started to do entity testing by themselves instead of utilizing special entity testing experts, the teams were not yet able to do feature specification nor system testing. Both of them are included in LeSS DoD. The lack of e2e scope is partly due to selected approach to use existing feature teams as a base for LeSS teams and slowly change them to multi-component LeSS teams via learning. Another option would have been to build new teams consisting of experts from different fields of development work. In practice, this means that change to true LeSS WoW practices takes time. Therefore, it is suggested that whodidwhat-analysis and wow-survey will be conducted again after the pilot has proceeded more and teams have had time to learn and expand their scope of work.

As mentioned before, whodidwhat-analysis results already showed improvement in the amount of entity testing and work done outside of team’s specialized system component. Increase in entity testing should improve source code quality as late entity testing has been a problem in Nokia LTE R&D before. Entity testing that is done on-time will also remove surprises and rework, thus making product development more efficient in a long run. From the survey, it was visible that non-LeSS teams had very limited visibility to entity testing cases and acceptance criteria. As a matter of fact, only 4% of non-LeSS respondents answered that entity testing is team’s responsibility. Based on the survey results, adoption of LeSS has truly shifted entity testing into development teams.
Survey also showed that LeSS team members learned more about dependencies between system components than their non-LeSS counterparts. Team members themselves mentioned e2e scope as primary reason for improved knowledge. The endeavor to split features and implement features with e2e scope modifying multiple system components was also visible in whodidwhat-analysis results. Lack of system level knowledge and dependencies to other system components were mentioned as biggest bottlenecks in WoW survey. By adopting multicomponent competencies within a team, these bottlenecks can be mitigated and thus development times can be improved. However, the improvement in lead times was not yet verified within the thesis due to relatively low contributions to other system components. However, it seems that LeSS WoW fosters learning through increased communication and multicomponent scope of the work within teams. Learning and increased understanding of the overall product also seems to increase work satisfaction, thus motivating team members to work and learn even more.

LeSS pilot teams seemed positive towards LeSS and its suitability to Nokia LTE R&D. However, many of the respondents replied that it is still too early to tell whether LeSS is suitable or not. To summarize, LeSS pilot has already shown positive changes in increase of entity testing practices, developers’ multicomponent knowledge and scope of work and in the amount of work satisfaction and team working practices. However, further analysis needs to be conducted later as the pilot proceeds, to see how LeSS affects to quality and lead times.
8 SUMMARY

This thesis project was a part of Nokia LTE business unit pilot project for new WoW practices for LTE R&D. Nokia LTE R&D consist of 5000-6000 people, which means that as a software development organization it can be classified as a very large organization. Traditionally, it has been thought that new agile software development methods are not suitable for such large organizations. As can be seen from literature, scaling agile methods to large organizations is one of the hottest topics currently amongst agile practitioners and academics. Based on findings from the literature, it seems that most of the challenges and issues regarding large-scale agile are known, but there is no universal solution to these problems. Therefore, most of the case companies found from literature had adopted custom agile way-of-work by combining practices from various agile models such as scrum and XP. Selected framework for Nokia LTE pilot is LeSS or large-scale scrum. LeSS is based on scrum, with prescribed way of implementing agile practices for larger organizations such as Nokia. LeSS is also emphasizing an inspect and adapt strategy to continuously develop WoW practices even more.

The desired benefits from implementing LeSS at Nokia LTE are improved quality, shorter lead times and improved employee satisfaction with more transparent organization structure. This thesis is focusing on providing ways to measure if LeSS actually brings any improvement in these topics. At the same time, thesis explained issues and challenges within Nokia LTE business unit that are affecting transparency and WoW methods. Current Nokia metrics were examined closely to check whether it would be possible to utilize them in verification of pilot results. Specific metrics were then derived to suit Nokia LeSS pilot verification process better. Thesis also presented preliminary results of two tools, that were suggested for Nokia to use in the future to evaluate pilot results.

In order to measure improved quality, the meaning of quality was split into two: functional quality and a freedom from deficiencies. For measuring functional quality, feature activation rates and customer feedback could be utilized. However, the feedback loop from functional quality is relatively long. Therefore, the results of LeSS pilot might be visible only after a significant amount of time. For measuring the effect on number of faults in the program, function point adjusted defect density metric was proposed. In contrast to current Nokia metrics,
proposed metric is utilizing function points instead of development hours used as a size estimate for the program. Function points are not affected by the efficiency of the feature teams and the suitability of technical solutions. With function points, defect density and many other metrics would be comparable organization wide.

In addition to defect density metric proposed, static analysis tools can be used to get instant low-level feedback of the quality of the code. The data from static analysis can be utilized both by feature teams and management. Using static analysis makes continuous integration much easier for feature teams as semantic faults can be detected right after committing the code. By following the trends in code complexity and duplexity managers can follow the change in code quality with much faster feedback than with fault metrics. Static analysis is especially suitable for LeSS pilot verification as it can produce instant low-level data about structure of the code as the teams start to work according to LeSS WoW.

For measuring the effect on lead times, it is critical to first separate the development phases of the features that LeSS teams are doing. According to LeSS DoD, teams would be responsible from system level specification to system testing. However, as the WoW survey and whodidwhat-analysis shows, teams are not yet doing the feature development with such scope. In order to have comparable data, function point analysis can be utilized to compensate differences in size and complexity of the features. Another option is to use certain selected features as a baseline, that match the features made in LeSS feature teams or wait until there is enough development items done to compare them with averages to non-LeSS development items.

To summarize, by making features commensurate, revising backlog and feature splitting practices and using fewer but better-defined metrics it is possible to follow the progress of LeSS pilot adaptation in an accurate way. Same suggestions would also increase transparency over the huge R&D machinery that Nokia has. In this thesis, function point analysis was suggested to make features commensurate as it seemed to be the most suitable way according to the literature. However, further research is needed to see whether function point analysis is feasible to implement in Nokia LTE R&D environment or not. Suggested further research would therefore be how to accurately commensurate features in Nokia LTE.
REFERENCES


APPENDICES


1. In the past 9 months, did you know the system level purpose of the features you were developing? Did you know how customer would/could utilize these features? Answer on a scale of 1-3.
   - I only understood the technical offering of the feature (i.e. input/output of the code)
   - I understood the functionalities of the features
   - I understood how customers would/could utilize this functionality and therefore knew how to maximize customer value

1a. In the past 3 months, has your level of understanding changed? How and why?

2. In the past 9 months, did you understand how different network elements needed to cooperate with each others, in order to implement the features you were developing?
   - I didn’t really understand.
   - I understood some of the key dependencies between network elements.
   - I understood basic interaction of different network elements and knew the basic parameters which relate to configuring these features.
   - I had a moderate understanding on the features in terms of messaging and protocols in different interfaces. I also understood how customer can change the configuration parameters related to these features in NetAct.
   - I had a detailed understanding on the features in terms of messaging and protocols in different interfaces. I also understood in detail how customer can change the configuration parameters related to these features in NetAct.

2a. In the past 3 months, has your level of understanding changed? How and why?

3. In the past 9 months, did you understand how different system components of eNB needed to cooperate, in order to implement the features you were developing?
   - I didn’t really understand.
   - I understood some of the key dependencies between eNB system components.
   - I understood code level implementation of one system component and interactions with other system components of eNB in terms of message sequences.
   - I understood code level implementation of at least two system components and interactions with other system components of eNB in terms of message sequences.
   - I had thorough code level understanding of all participating system components and I was able to make changes to all of them if needed.

3a. In the past 3 months, has your level of understanding changed? How and why?
4. In the past 9 months, to which system components you have committed code into?

4a. In the past 3 months, did you contribute to system components you haven’t been contributing before? If yes, to which ones?

5. In the past 9 months, what types of feature development work you have been contributing into? Choose below. (Mark also contributions, without having the main responsibility of the type of work)

☐ SFS
☐ EFS
☐ Code
☐ SCT
☐ ET
☐ SyVe

5a. In the past 3 months, have you done feature development work listed above you haven’t done before? If yes, what type of work?

6. In the past 9 months, how the feature decisions (content of features, feature splitting etc.) were done and communicated?

6a. In the past 3 months, has the feature decision process changed? If yes, how?

7. In the past 9 months, how have you split the development related tasks within your team?

7a. In the past 3 months, has the way of splitting development related tasks changed? If yes, how?

8. In the past 3 months, has there been any surprises in implementation of the feature (needs for change in scope, size estimate for the content proving to be wrong, etc)? If yes, what was the surprise and how it was handled?

9. How often do you communicate with your team mates about development work, such as discussing about technical properties and implementation of the code?
Once a week or more seldom.

Couple of times in a week.

Once a day.

Couple of times (5 or less) in a day.

Multiple times a day.

9a. In the past 3 months, has there been any change in the amount of communication and to what direction?

10. How did you show the results of your last sprint’s/FB’s work to your product owner and to other teams?

11. Have you seen Entity Test level test cases or acceptance criteria of the feature? Who is responsible of the Entity Testing related activities in your feature team?

12. Does your team thrive towards continuous improvement? If yes, how?

12a. In the past 3 months, has the situation regarding continuous improvement changed? If yes, how?

13. What are the biggest bottlenecks and problems you face in your work?

14. How would you rate your work satisfaction on a scale of 1-5, one being the worst and five the best grade for job satisfaction?

1, I don't like my job at all.

2

3

4

5, I love my job!

14a. In the past 3 months, has your work satisfaction changed? If yes, to better or worse and why?

15. In your opinion, how has adopting LeSS – WoW affected development lead times within your team?

I can't tell.
Lead times got significantly longer.
- Lead times got a bit longer.
- There was no change.
- Lead times got a bit shorter.
- Lead times got significantly shorter.

15a. Could you name the main reasons for the change?

16. In your opinion, how has adopting LeSS – WoW affected code quality within your team?
- I cant tell.
- Quality got significantly worse.
- Quality got a bit worse.
- There was no change.
- Quality got a bit better.
- Quality got significantly better.

16a. Could you name the main reasons for the change?

17. How has adopting LeSS – affected your work satisfaction?
- I like my job a lot less.
- I like my job less.
- Not at all.
- I like my job more.
- I like my job a lot more.

18. What is your opinion about LeSS-WoW? Is it suitable for LTE R&D at Nokia?