Aalto University
School of Economics

# Optimizing release planning under Scrum in complex multi-team software projects. Case: Nokia Siemens Networks

Technology Management and Policy

Master's thesis

Anatoly Tolstukhin

2012

Department of Information and Service Economy
Aalto University
School of Economics

# Abstract

Aalto University
School of Economics
Master's Thesis
Anatoly Tolstukhin

April 14, 2012

The aim of this thesis was to identify and explore existing challenges in large-scale agile projects and to provide recommendations on how to optimize release planning and tracking methods under Scrum in a complex multi-team software development environment. Importance of velocity and effort estimations, as well as visibility and allocation of user stories in a multi-team setting were particularly addressed in this study.

The results of this study include empirical findings related to agile release planning and tracking methods in a complex agile development environment to support more accurate decision-making. New management tools were developed for practical use, which support the main results of this thesis.

The findings are based on an actual case study and full-time thesis work at Nokia Siemens Networks. The results were presented to the leadership team and management of the organization where I worked, and demonstrated practical benefit for the case company. Additionally, this thesis includes a practical application of results to highlight how improved visibility could support better release planning and tracking in multi-team agile software development projects. The tools proved to be useful and were put to deployment in the case company.

**Key words**

agile release planning, effort estimations, velocity, multi-team agile software development, Scrum management tools, Scrum case study

# Foreword

I have applied for a thesis worker position at Nokia Siemens Networks to write my Master's thesis in the topic, which particularly interested me. From the beginning, I had high motivation to do my best and gain valuable practical experience in project management and agile methods. Looking back at my accomplishments during last six months, I am more than thankful to the company for giving me the opportunity and tools to develop further my personal skills and professional knowledge, as well as to complete the final stage of my studies. It has been particularly interesting and motivating to work on a task, which was of high importance for the company and that results had practical benefits.

I owe special gratitude to people who have helped me to succeed in this project, supported me along the way and shared their opinions and knowledge about current processes and management from the practical side, thus leading me to making right conclusions and practical recommendations.

I would like to thank NSN program management, who have been very supportive and guided me throughout the thesis work. From the beginning I was provided with all necessary information and kick-off sessions needed to start my work independently, as well as numerous meetings and discussions on the regular basis.

Also, I would like to thank Professor Matti Rossi for supervising my thesis work progress and providing feedback at all stages, as well as finalizing the published version of the thesis.

# Table of contents

# Table of figures

# Introduction

## 1. Introduction

In contemporary software development projects companies are facing challenges, such as constantly changing requirements, pressure to deliver faster, and the need to cut costs due to competition (Alleman, 2002; Huss, 2007). In an attempt to deal with these challenges new methods evolved, which became known as *agile project management* and *agile software development* (Cohn, 2006). The agile approach aims to produce high-quality software products faster, to create more value and to satisfy customers' needs better (Beck, 2001). Studies have been done, showing that agile methods improve productivity and project success in software development. The popularity of agile methods has grown since then and having initially proved to fit software development in smaller companies, currently many large organizations also started moving to agile approach. While in the early 2000s most literature has related to small-scale agile software development projects (Tolstukhin, 2009), only recently literature started shifting towards research of agile methods in complex projects and large organizations. (Stober and Hansman, 2009; Vähäniitty et al., 2010).

Planning and tracking development processes in agile software development are fundamental for successful projects (Chow and Cao, 2008), including efficient management of resources and continuous monitoring of development progress using the agile metrics. Multi-team context and large product size complicate planning and tracking processes in the agile environment. Failing to identify and address additional factors makes planning unreliable and visibility of progress ineffective, thus posing further risks on project success.

Therefore, applying agile methods in a complex environment requires a more organized approach than in simple agile software development projects. Apart from assessing uncertainty and risks, multiple additional factors need to be considered in agile release planning. These factors include prioritizing and estimating size of requirements, planning resource availability, as well as calculating velocities and tracking progress of different

teams. Combining and managing these factors in release planning is called *release planning optimization* (Heikkilä et al., 2010).

Since generally agile methods are still quite new, there is a growing demand to understand the challenges in complex and large-scale development environment. Heikkilä et al. (2010) mentioned that the existing studies on agile software development projects focus on discussions about the positive aspects of agile methods in an ideal agile world and highlighted the importance of additional studies related to challenges. Further, whereas earlier in the studies planning has not been considered important or useful in agile software development context, Mike Cohn (2006) wrote a famous book called *"Agile estimating and planning"* covering extensively agile planning methods on the theoretical level in a single-team context. Among other authors, Chow and Cao (2008) have carried out a literature review and a survey study to identify the success factors of agile software projects. Dybå and Dingsoyr (2008) have carried out a practical research on identifying agile methods' dynamics in a real business context. Recently, a simulation of software development under Scrum was carried out in an academic environment, highlighting the main aspects and success factors in implementing agile methods (Mahnic, 2011). Even though the results have been interesting, there are many limitations to apply these results in a real world situation. Heikkilä et al. (2010) carried out a case study on the adoption of agile methods in a multi-team development environment of complex systems. Vähäniitty, Rautiainen, Heikkilä and Vlaanderen (2010) have done an extensive research on current situation in the software development industry. The authors emphasize growing complexity of agile development projects and provide guidelines how the planning should be carried out in multi-team environment. One research was also found on how to approach release planning through statistical methods and simulations. Momoh and Ruhe (2006) provide an industrial case study analysis on strategic release planning through the use of the intelligent planning tool called ReleasePlanner. Overall, one can observe that the importance of planning in agile software development has been lately recognized.

It was identified that little practical research exists on agile release planning processes and challenges in the large-scale development setting. This thesis focuses on the study of Scrum release planning methods and metrics in large and complex software development

projects. Further, I will discuss my motivation for the study and the research objectives, as well as state the research problem, research questions, methods and the thesis structure.

### 1.1. Motivation for research

Having done my Bachelor's thesis on the values and success factors of agile project management (Tolstukhin, 2009), I got a good understanding of the agile processes and became interested to extend knowledge in this area also in my Master's thesis. Nowadays, agile methodologies are of high interest, as they offer more lean and rapid development methods in growing and highly competitive software markets. As discussed, while large organizations have adopted agile methodologies, few studies have yet been done about the applicability and specificities of these methods in large software development projects. Facing several difficulties in scaling agile methods to a more complex environment, this research on agile project management methods in a large-scale software projects should currently be of high value to many companies. Effective management in a complex environment requires better understanding about agile teams' dynamics, as well as knowledge how to adapt and keep agile methods and metrics under control when the characteristics of the development environment change.

Understanding the importance of release planning processes and the need for a tool to assist release planning and tracking, Nokia Siemens Networks showed interest in the case study of Flexi software product on the topic of agile release planning optimization. The company has initially expressed a view that more precise velocity calculations and effort estimations for planning purposes would be extremely useful. Through in-depth study of these parameters, together with the analysis of the case company's existing development structure under Scrum provided empirical findings about the specificities and challenges of agile practices, as well as practical recommendations to optimize release planning in such projects.

### 1.2. Research objectives and questions

While some studies have been done related to challenges in agile release planning, the existing models are designed based on theoretical assumptions and have not been

validated in real business case situations (Heikkilä et al., 2010). This thesis will deal with the actual problem faced by a global company in agile project management, where the main challenge is optimization of methods given complex and large development structure. Analyzing the case and identifying the parameters that effect the velocity and effort estimations in multi-team agile projects, a model will be created to perform release planning and tracking more accurately.

This thesis provides an empirical evidence-driven study about challenges of Scrum processes in a complex development environment. None of the existing research papers were found to address problems of agile release planning in a large organization's scenario through the study of velocity and effort estimations as the key measures. Also, resulting from the findings and extensive studies of the case company's agile development methods, this thesis sets the requirements needed in agile management tools to support planning and tracking of development in similar software development environments.

The objectives of this research are to improve the reliability of agile release planning at Nokia Siemens Networks by:

- *Determining what information is necessary in the product backlog to enable reliable forecasting and monitoring*
- *Investigating the velocity and effort estimations parameters across the teams in the company*
- *Performing analytical study of historical data to identify problems in earlier development under Scrum*
- *Proposing improvement actions and recommendations*
- *Creating management tool in Excel to improve accuracy of release planning and visibility of development progress*

As a result, the aim of this thesis is to provide empirical results on how release planning and tracking should be conducted when there are multiple development teams working on one complex software product. In particular, the main research questions are related to the velocity and effort estimations in a multi-team development environment, including

- *How to calculate velocity in order to make release planning more reliable?*

- *How can effort estimations be improved and standardized across the distributed teams?*
- *What are other methods to make planning and development of the Flexi more accurate and efficient?*

Since this study was facilitated and verified in a real business environment, it can also serve as a guide for companies on how to implement changes and to optimize Scrum methods to larger and more complex software development projects.

## 1.3. Methodology and structure of the thesis

The research methodology is the documentation of full-time work for Nokia Siemens Networks on agile release planning optimization in Flexi software development as the case study. This thesis is organized in three distinctive parts: Theoretical background, Analytical Part and Practical part.

The Theoretical Background is based on a *literature review* related to agile release planning and other parameters used to estimate velocity and plan project resources. This section is important to familiarize the reader with agile practices and to set the baseline for the analysis.

The Analytical Part includes practical work and familiarizes the reader with Scrum methods applied at NSN. Considering the complexities of the development environment in the case company, the following analysis methods were carried out:

- *Regular discussions* with the Flexi management at NSN
- *Data analysis* of product backlog
- *Questionnaire* across all development teams

Reflecting on the theoretical studies and the analysis results, this section aims to discuss the identified main problems and root causes in the case organization, as well as to provide practical recommendations and improvement actions.

The last section of the thesis, i.e. the Practical Part includes applying the results to build new agile management tools in Excel, which would provide higher accuracy and visibility in planning and tracking release development within the case organization. The tools were designed based on the analysis results from the study and thus should reflect on the main parameters which companies need to address in a similar agile development environment.

# Theoretical Background

## 2. Agile Methods

The aim of this section is describe the principles of agile development and Scrum methodology in particular based on the review of the existing literature. Firstly, the common practices behind the agile development will be explained including the values and principles. Further, within the scope of this thesis I will discuss agile planning methods in software projects and the challenges that exist when applying these concepts in a multi-team environment. This section aims to familiarize a reader with agile release planning processes through the literature review and serves as a basis for the case study and the empirical research.

### 2.1. Agile principles

For the past decade agile methods have gained a lot of attention and popularity in the areas of project management and software development over the traditional methods, which have had poor results. Agile methods were developed in attempt to perform better in the new market environment, where businesses are becoming more complex, requirements are changing quickly, and the pressure to cut costs and deliver results fast is growing (Augustine, 2005).

The values and principles of the agile methodologies have been documented in the "Agile Manifesto" in 2001. *Figure 1* illustrates the main values behind agile methods as explained in the purpose of Agile Manifesto.

*"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

1. ***Individuals and interactions*** *over processes and tools*
2. ***Working software*** *over comprehensive documentation*
3. ***Customer collaboration*** *over contract negotiation*
4. ***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"*

*Figure 1. Values of Agile Methods (Agile Manifesto, 2001)*

Agile software development clearly differs from the traditional methods. Under traditional methods, the whole project goes through the development process in one cycle known as the waterfall model, which includes extensive planning of the entire project, followed by design, development, testing, delivery and maintenance. The planning stage carries high investment, and the main assumptions under the traditional method are a stable environment and fixed requirements. Therefore, any changes will require returning to the planning stage and abandoning the completed work. Furthermore, if the project is abandoned before completion, there is no return on investment.

The main idea behind the agile method is that the whole project is split into parts called iterations or sprints, each lasting from two to four weeks. Every sprint involves work through all stages of the project including requirements analysis, planning, design etc. and delivers to the customer a set of working features. The features are prioritized in such way that the most valuable and profitable components of the product are delivered first while less important components are postponed until later. Importantly, the evolving project requirements are continuously re-assessed based on the feedback from the product owner (Alleman, 2002). Iterative development mode allows a team to continuously re-evaluate and improve the methods used. In such way, maximum value is delivered continuously throughout development, even when requirements change.

Due to the adaptive nature of agile projects, the overall risk is minimized and the total value is much higher. Under agile method the value is created continuously as the project evolves, whereas with traditional model the value is generated only upon completion. Higher value is also generated because the requirements are understood better and the parts of functionality are delivered faster to the end user. Since agile method allows flexibility and adaptability to changing requirements, the risk of project failure due to changing customer needs is lower (Augustine, 2005). Since this method provides first functional results early in the project, even if the project was cancelled there would be some salvage of value.

Agile principles are a set of methodologies, which address different areas in the software development. While the methods of how the development is approached differ, principles are fairly similar (Hass, 2007). The most common agile methodologies are Scrum, Extreme Programming (XP) and Feature-driven development. Due to the scope of

this thesis, I will focus on and discuss the Scrum methodology in the software development projects context.

## 2.2. SCRUM methodology

Scrum methodology is the most common practice used in agile software development nowadays (Vähäniitty, 2010). Scrum focuses on situations where it is difficult to plan ahead and the process can only be roughly described as an overall process. Since the activities in such settings are loose, rigid rules are used to keep the development process under control and to tackle possible risks. Scrum is a very flexible approach, where the overall project deliverables are partitioned into prioritized fractions. Fractions have a clean interface and are developed by self-organizing teams iteratively in sprints. Among others, according to Cohn (2006) and Larman (2006), using this approach one can test the feasibility and technology of the software requirements already after initial cycles and continuously through sprints.

In Scrum, the duration of a sprint is from two to four weeks long. Each sprint includes planning, design, development and review. The work is coordinated by the team members and the team manager, also known as Scrum Master, who is in charge of maintaining processes, assisting in solving problems and assuring that all tasks flow smoothly (Cohn, 2006). *Figure 2* below illustrates the cyclical process of software development propagated by the Scrum methodology and each step is discussed further.

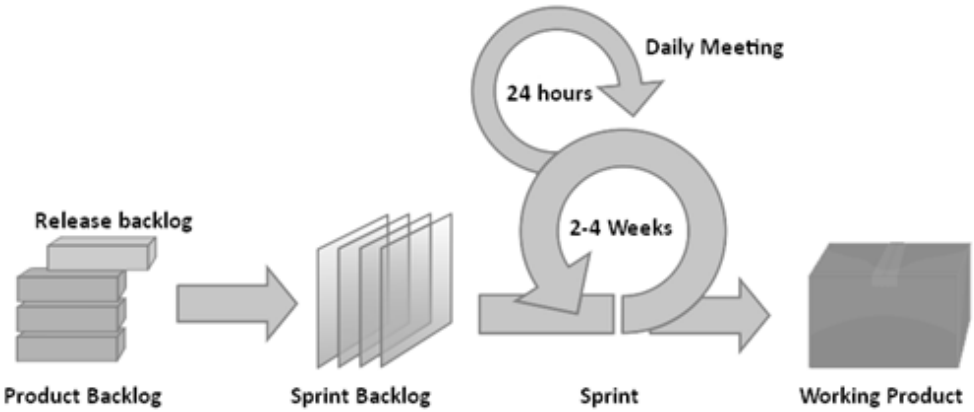


*Figure 2. Scrum cyclical process*

The Scrum development process begins with collecting all potentially relevant features into a common list called the *Product Backlog*. Product backlog is updated when the requirements change or need to be updated. Once the list is created, the team needs to identify which of the features should be included in the next release from their importance and value to the customer. Some unnecessary features may be excluded from the list if they are not feasible. It is vital to identify the core features (also called unbreakable features), which must be included in the release. Core features are the absolute minimum that has to be completed before the given release can be delivered to the customer. These requirements are listed in the *release backlog* together with other high priority supporting features. Release backlog is a subset of a product backlog, which includes the requirements only for one specific release. The product backlog and the release backlog are the most important elements used in planning and tracking the progress on each level.

When the release backlog is created, the highest priority items are assigned to earliest sprints. Each sprint should have a *Sprint Backlog* containing a set of requirements assigned. During each sprint teams break down the requirements into smaller tasks and use cards and task board to identify the type of task (e.g. coding, integrating etc.). Stick-on notes on a board are a common way to coordinate the sprint tasks. As work progresses the cards are moved based on the status, i.e. from pending to being completed. Task board assists team members to monitor tasks within the sprint. *Daily Scrum* meetings are brief stand-up meetings meant for checking up on project status and keeping it on track. In these meetings, the team members report their work progress during the last 24 hours, what is going to be done next and possible obstacles to reaching the goal.

On the *sprint review* meeting, sprint achievements and obstacles are discussed and shared among the teams and managers. In the end of each sprint a set of tested and working items need be completed, which could be potentially shippable to the customer. Also, uncompleted features are moved to later sprints and the release backlog is updated. Each sprint is summarized on a R*etrospective* meeting, where participants may suggest improvements based on lessons learnt during the previous sprint. As customer needs change the requirements may be re-prioritized and next top priority requirements will go through the development cycles in the following sprints. After all sprints are completed, the working product can be delivered to the customer.

### 2.2.1. Characteristics of SCRUM

Under the Scrum methodology, the processes are defined during the release planning and closure phases (Larman, 2006). On the other hand, during the development phase the project is highly responsive to changing requirements and the final deliverable can be modified throughout the development. Scrum has a high level of engagement with the user or product owner throughout all stages including design, development, test and maintenance until the product is stable and useful (Gill, 2011). Continuous collaboration with the customer and adaptive nature of development assure that the final product will have the highest value.

In Scrum, the actual development takes place during the sprints. The team is given full responsibility to perform tasks and follows certain rules, such as daily meetings, sprint duration. Thus, the development process under Scrum takes place in a controlled black box. (Cohn, 2006). In order for a project to be successful, team flexibility and creativity must be unlimited. Also, team members should be skillful and cross-functional, i.e. be able to perform various tasks across the project. Due to the low level of documentation, close collaboration and communication among team members is essential for the success of the project.

Team size also plays an important role and should be kept small to around 6 members, but there may be multiple teams for bigger projects (Cohn, 2006). Small and collaborative teams are better able to share tacit knowledge about the development process across team members. Larger team size, on the other hand, will decrease the productivity because team collaboration becomes too unproductive (Stober et al., 2006).

Since a project evolves throughout the sprints, the final product scope, cost and completion date become more clear during development. Therefore, while one of these parameters is fixed in the planning stage and other are flexible.

### 2.2.2. Management under SCRUM

On the product level, the management defines the initial content and timing of the release based on the metrics, which will be explained in more detail later in Section 2.4. Since the requirements are changing, unpredictability and complexity need to be controlled by tracking accurately project progress and changes of variables. Spreadsheets or agile

project management software are commonly used instruments for managing the requirements and monitoring project progress with charts. Due to minimal documentation, also face-to-face meetings are essential throughout the project to plan and track activities, and ultimately to control the project.

On the sprint level, the team is given the responsibility to manage the activities assigned for the respective sprints. S*crum master* is assigned to the team to assure that the team members don't face obstacles and the project runs smoothly. During the sprint, lightweight management methods are common, such as using post-it notes and whiteboard for listing features and their status. In larger projects, agile project management software is highly recommended because it makes communication easier and enhances project progress visibility.

Due to the nature of agile practices, tracking the progress is extremely important for the management. Since work progress can be tracked upon completion of the requirements, rigid rules exist regarding the status of different activities. Thus, a user story will get a status of being completed only if it meets all criteria pre-set in the *Definition of Done (DOD)*, which includes both development and testing activities. The duration of sprints is strictly set and uncompleted items will be moved to a later sprint or possibly canceled. Therefore, even if an item has been partly done during a sprint, it will not be counted in the actual work done for that sprint.

After every sprint, the teams reflect on the processes and suggest possible improvements, which are then included in *retrospectives.* Retrospectives serve as a feedback tool for managers to improve the processes and team spirit.

Therefore, communication and visibility of work performed by self-managing teams are the key aspects of management under Scrum development.

## 2.3. SWOT Analysis

Agile methods are a set of techniques used in current software development practices that apply a human-centered approach (Ceschi, 2005). These methods have proved to deliver products faster and with better quality. However, no method is perfect and apart from strengths, these methods also have threats and weaknesses. Shahir et al. (2008) have

discussed and suggested some improvement strategies for agile processes. In the following section I will cover the strengths, weaknesses, opportunities and threats based on the literature review. The aim of this section is to highlight problematic issues in agile processes, as well as the opportunities for managers and team members how to tackle those problems. The summary table of the SWOT analysis can be found in *Appendix 2*.

***Strengths***

The main strengths of the agile methods have been already addressed to some extent earlier in this thesis. Most importantly, agile methods allow the project to be flexible and adapt to changing requirements even in the later stages of the development process. This allows the results of the project to be relevant even if the development takes several months. Stakeholders and users are more satisfied with the results of agile projects compared to the traditional methods because under agile development the products match customer needs better. Additionally agile methods deliver higher value and more frequently, as well as the quality is higher since the increments of the final product are continuously tested after every sprint. Due to improved quality, flexibility, high level of communication and requirements prioritization, the overall delivered value is higher.

Shahir et al. (2008) also discussed other strengths of agile methods, including regular effective planning and visibility of the progression of the project. Everybody involved can see precisely the project status. Further, simplicity of the processes and design, as well as elimination of waste by doing only the required tasks - are collectively considered to be the strengths of the agile methods.

***Weaknesses***

Surprisingly only few sources discuss the weaknesses of the agile methods. No doubt, agile development is a major improvement over the traditional methods. However, like any other process it has drawbacks.

Since agile development is a human centered approach, the lack of documentation and inefficient communication may cause problems, including communication issues in globally distributed development teams due to limited communication possibilities, as well as cultural and organizational differences (Shahir et al, 2008). It is not as easy to delegate progress across multiple teams in different locations. While changes in the product backlog

have to be delegated to all teams, there is a risk that especially the time spent on meetings can grow out of proportion.

Secondly, the people-oriented approach causes heavy reliance on the development team. While it is assumed that all members in agile development teams are cross-functional, in larger projects it is not always the case. A crucial team member leaving in the middle of the project may pose a serious risk on deadlines and the project in general. Also, as the team size increases, agile mechanisms fail to act effectively. Thus, with multiple teams managing teamwork issues and keeping track of the individual teams' progress becomes more complex. This makes many agile projects difficult to control effectively.

Allowing frequent changes to the requirements is an agile principle. However, it complicates the estimation of time and cost, making it difficult to forecast and plan resource allocation. Lack of overall planning poses risk and bottlenecks, which may not be noticed initially, but the project may fail to fulfill the requirements. Even though agile development is a very useful technique, in developing large and complex software agile methods have limitations. Some large and complex systems require a central architecture and detailed initial planning. These estimations are usually very imprecise, especially as the requirements change. Additionally, as requirements become more complex and increase the workload, tracing dependencies between different items becomes difficult.

Lack of documentation and poor estimates cause limitations with contracts, where precise requirement specifications are needed. Problems also arise where guidelines for testing and "the definition of done" are insufficient or the guidelines are not strictly followed.

***Opportunities***

Having addressed the weaknesses of the agile processes, I will now aim to identify the possible opportunities how teams and managers can strategically improve their agile processes.

Both, inter-team and intra-team communication should be valued by all members. In distributed software development teams, teleconferencing and web-based development environments should be utilized to the maximum. Planning and forecasting are the core activities in controlling the development of the project. Thus, adding more functional

metrics and measures may improve planning and forecasting reliability, even with constant change of requirements.

Improved agile management tools can create an opportunity to better manage information and to apply agile methods especially in more complex and large projects (Tolstukhin, 2009). In this way organizational, people, process and technical aspects in agile projects can be further improved. Thus the use of tools should be considered as an opportunity for future growth in large companies who already use agile methods.

Finally, expert advice and knowledge should be utilized fully to adjust the agile methodology to fit the product development at hand.

***Threats***

The major threat of agile methods is lack of interest in utilization of agile methodologies in traditional organizations and failure to incorporate real changes to the processes. Companies may be reluctant to dramatically changes in their processes, underestimating the benefits of correctly implemented agile methods.

Companies that already utilize agile practices should continuously seek for possible improvements in the operational processes. Since the processes in agile methods are more or less based on trial and error, companies need to be innovative and try out new methods that might better suit their product and culture.

In large organizations the failure to adjust to changes, and to utilize new techniques and tools may become an obstacle to success. Large companies may start implementing agile methods in a small team, and after the initial success expand it quickly to the company level. There is a threat that when scaling up agile development, there is a need to change the techniques and adjust methods. Large companies may either not have the experience or time to switch to new tools. Utilizing less productive and potentially impractical methods poses a threat to the company.

## 2.4. Agile Planning

While agile methods are promoted by a lack of long-term planning, the truth is that the process of planning is extremely important in all kinds of projects. The process of agile

planning is carried out in a different manner as compared to the traditional methods, and thus is often misunderstood.

Estimating and planning are crucial to the success of any software development project. These activities affect the investment decisions and give information, which helps tracking project progress (Cohn, 2006). Planning is difficult and many projects fail to meet the planned deadlines. It should be carried out in right amounts because too little planning will not give the needed information, while too much planning will cause plan updates after every change in requirements. Progress tracking is strongly interrelated to planning because the actual progress is evaluated based on the plans providing hints to the corrective actions and decreasing uncertainty.

The following section will cover planning and tracking processes in single team and multi-team agile software development environments. Due to the scope of my thesis, the main focus will be on the release level.

### 2.4.1. Agile release planning

Agile software development consists of planning at multiple levels including strategy, portfolio, product, release, iteration and daily levels (Cohn, 2006). Different levels of agile planning are illustrated in *Figure 3*.


*Figure 3. Agile planning levels (Cohn, 2006)*

Strategy, portfolio and product levels are part of a company's long-term goals, which should provide a roadmap for product management i.e. an overview of product development in future planned and upcoming releases (Vähäniitty, 2010). According to

Cohn, long-term plans are very abstract, and agile teams plan in the short-term on release, iteration and daily levels.

*Figure 4* below shows the Agile project hierarchy where the software project is the highest level. The software project consists of a number of releases, from which some are minor updates and others are major softare improvements. Further, each release is dividied into sprints. Each level has a respective specified definition of requirements definition (Vähäniitty, 2010). On the program level, the requirements belong to *epics,* which are used to define different categories within the product. On the release level, the requirements are called *features*, which are part of epics. Further, features are broken down into *user stories* and added to the list of requirements on development level. In sprint planning, user stories are broken down into *tasks*. As we go down the hiearachy the size and complexity of the requirements is decreased. However, attention should be paid to tracking back the requirments up the hierarchy tree as well.

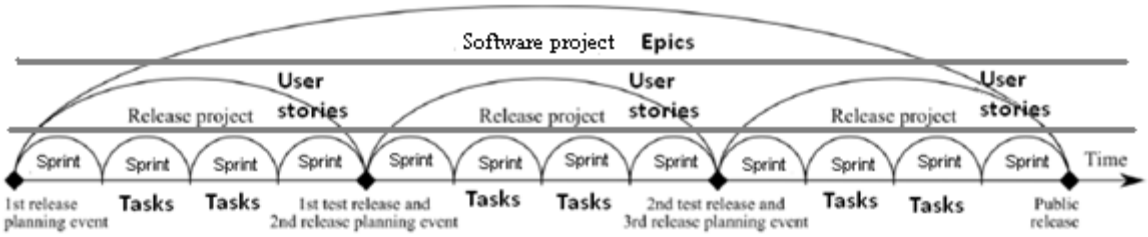


*Figure 4. Agile project hierarchy (Vähäniitty, 2010)*

Release planning deals with gathering and assigning the features to a deliverable package so that the business, technical and resource constraints are met (Vähäniitty, 2010). Planning the next product release is one of the most crucial success factors in agile software development projects (Rautiainen, 2010). The aim of the release planning process is to identify when a releasable version of a software product would be made ready and what functionality it should include (Logue & McDaid, 2008). The subject of release planning has been addressed in the literature as a challenging process. Failing to optimize the required features with the available resources commonly results in problems to meet deadlines (Kittlaus & Clough, 2009). In agile release planning, developers need to have deep understanding of the technical features required in a release to be able to make accurate estimations and balance the resources with the desired requirements.

Additionally, there is certain level of uncertainty in time and cost to develop the chosen functionalities, as well as in the value of features. Especially those releases, which are under contractual obligations, require accurate planning because failure to supply agreed functionality or to meet deadlines can be very costly.

#### *2.4.1.1. Release planning process*

Release planning process begins with identifying the most relevant features from the product backlog. Product backlog lists all potentially useful features for the future releases to succeed and contains up-to-date information on stories status, sprint commitments, size, value, etc. When all features for the next release are identified, they are broken down into user stories. A user story is a definition of the required functionality and is expressed in a simple sentence such as "As a <User Type>, I want <capability> so that <business value>". The user stories are included in the release backlog, which is a list containing all the needed functionality for the given release (Larman, 2006). When all candidate user stories have been identified, the development team estimates size of each use story. Size estimation is normally expressed in *story points*, which is an estimate of the amount of work needed to complete each user story relative to one another (Logue & McDaid, 2008). In other words, a story point is a measure for expressing sizes of different tasks proportionately compared to one another e.g. a task with size of 2 user points should be twice bigger compared to a task of 1 user point.

When user stories for the coming release were identified and their sizes estimated, the product owner prioritizes the user stories according to their value and size, and identifies the minimum marketable features which must be fulfilled before the release is ready to be delivered to the customer (Vähäniitty, 2010). Resource and time constraints are taken into consideration to decide upon the viability of features. Determining the business value is important because 80% of the project value may be derived from 20% of features. Thus, in larger projects, multiple releases are often planned simultaneously through the process called *joint release planning*. However, usually only the topmost items are prioritized and developed within current release because the requirements tend to change (Vähäniitty 2010).

Based on the information gathered about the requirements, the product manager either forecasts how long it will take to complete the required features or estimates the amount of user stories that can be completed by a specified date. Scheduling a release requires estimating the size and duration of the desired features, as shown in the *Figure 5* below.

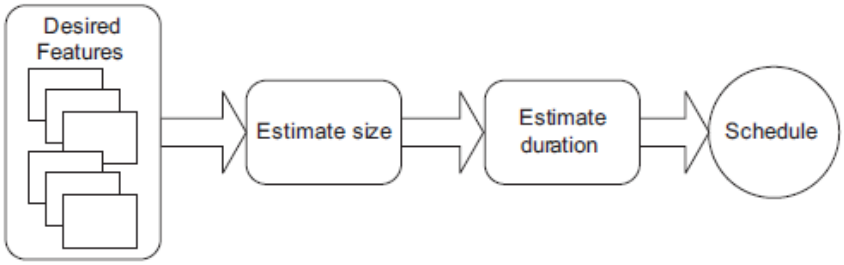


*Figure 5. Release scheduling. (Cohn, 2006)*

To estimate the duration, the product manager needs to know the *velocity* of the team or teams carrying out the development task. Velocity is the speed or the amount of work a team completes in one sprint. Velocity is the main parameter used in the planning process and will be discussed in more detail separately in the next section (2.4.2.).

The development process of the release can be started after the release parameters or deadlines are agreed among the product owner, the line manager as well as the development teams assigned for the release. The plan should include the scope, schedule and resources for the given release (Cohn, 2006).

Given the scope and velocity, the release is broken down into sprint cycles of a pre-specified duration. The features are denoted to iterations based on their priority and each feature should fit into one iteration. If a feature is too large for one sprint, it should be broken down into smaller tasks (Cohn, 2006). As stated earlier, a release consists of a set of requirements that must be completed before the release can be delivered to the customer. Release planning includes the following activities:

1. Determining the Scope (user stories that must be developed)
2. Estimating size of user stories
3. Composing the release features given the available resources
4. Estimating the release date

Since the release consists of a number of sprints, sprint planning and monitoring deliverables is part of the release planning process. Sprint planning usually takes place on the development team level where the team members commit to the tasks and confirm the

size and scope of the requirements. During the daily stand-up meetings team members plan activities for the day and present the results from the previous day. The goal is for everybody to know the current state of the project and what should be done next.

In agile planning, certain level of uncertainty is accepted. The uncertainty decreases as the project progresses and the original estimates are improved. Boehm (2002) developed a concept to explain that estimations are very vague in the beginning of the project and that estimations need to be redone on the regular basis. As the project proceeds the estimations become more certain. This concept was discussed by Cohn (2006) and Larman (2006) and named as the *cone of uncertainty*. Also, Cohn argued that the accuracy of estimates can only be done for a couple months ahead, long-term estimations are rough and the plan needs to be refined after every sprint (Cohn, 2006). While it is impossible to get accurate estimations, a good planning process reduces risk and uncertainty, supports better decision making, and conveys information.

### 2.4.2. Velocity

Velocity is a measure of a team's rate of progress in a given sprint and is the main parameter used in release planning. This measure shows the amount of story points a team can complete on average in one sprint. Velocity is calculated based on the team's historical performance. For example, if during one sprint a team completes 10 story points, its velocity for that sprint is 10. Based on this information and all other factors constant, one can estimate, for example, that during the next sprint the team will have the same velocity and will also complete 10 user stories.

Given the overall list of user stories with the relative story point estimations, one can calculate the expected total size of the release summing all user points. Further, the duration of a release, i.e. number of sprints needed, can be calculated dividing the estimated total size by expected team velocity. For example, if release size is 60 and team velocity is 10, it should take 6 iterations to complete the release. One should keep in mind that the estimations are not accurate and need to be re-assessed after every sprint.

However, expected velocity may vary and thus should rather be considered as a range. There are different options concerning how a velocity can be estimated. Cohn suggests

three approaches to estimating velocity: using historical data, running a sprint and making a forecast (Cohn, 2006).

#### 2.4.2.1. Using Historical data

Historical data of velocity may be extremely valuable in situations where little has changed between the old and the new team and project. Cohn suggests considering whether technology, tools, teams, product owner, working environment or people making estimates have changed before making velocity estimates based on the historical data. Even if these factors don't change much between releases, it is good to express velocity as a range. Further, if some factors have changed, the range of uncertainty can be bigger or alternatively other approaches for estimating velocity can be used.

#### 2.4.2.2. Running a sprint

Cohn suggests that the best approach to estimate the velocity is to run 1-3 sprints and use the observed data to estimate the velocity for the release. Since it takes certain time to plan the release and finalize the requirements, it may be practical to make a team complete a few sprints right away and based on the observed velocity plan the release date.

Running only one sprint is usually not reliable (especially for new projects) because the teams may concurrently do preparations or learn to work together. Thus, if it is possible to hold off giving the estimated release date for at least two sprints, the observed velocities together with the range of uncertainty can provide a good estimate. Additionally, if a team can run three or more sprints, the velocity can be forecasted, for example, using the average or median of the observed values.

Running initial sprints makes more adequate forecasts regarding how quickly a team can progress and allows addressing the potential risks.

#### 2.4.2.3. Making a forecast

In some occasions there is no appropriate historical data available or it is impossible to run the initial sprints to observe the velocity, for example because the project is not starting soon, or the contract has to be signed prior to the beginning of work. Thus, forecasting the velocity may be the most feasible solution. Forecasting the velocity involves:

- Estimating the number of hours that each person will be available on each day,

- Determining the total number of hours that will be spent on the project during sprint,
- Selecting user stories and expanding them into tasks to determine how many tasks can fit into one sprint
- Converting the velocity into the range

Cohn suggests that the observed velocity is the best method and regardless of which method was initially used and recommends switching to using actual values and its likely range of completion dates.

### 2.4.3. Effort estimation

Effort estimation is a measure of the amount of work needed to complete a user story. Effort estimations show the relative size of different user stories and are measured in story points (Cohn, 2006). A story point is defined by a team and is translated into the amount of effort needed to complete a story, for example, 1 story point equals 1 ideal day of work. Story points can be estimated with any unit of measure, but have to be clear and consistent across different user stories so that the estimated amount of effort needed for different user stories is proportionate. For example, a task that is twice as big as another task should have twice more story points.

In agile methodologies the experience of the team represents the basis for estimating the effort needed from the high-level requirements (Buglione, 2007). Since a user story comprises multiple technical tasks carried out by different team members, the experience of all members should be considered and required effort estimates should be done by the whole team (Cohn, 2007).

Accuracy of estimation directly depends on team members' knowledge of the technology and past experience with similar tasks (Larman, 2006). Also, the amount of time spent on effort estimation affects how accurate the estimate would be. According to Cohn, accuracy and time spent on estimating effort act according to the law of diminishing returns, meaning that at a certain point estimation will not become more accurate. Instead, Cohn suggests spending just enough time for estimating effort and use corrective actions to re-estimate in the process.

Literature suggests that higher priority items may need more accurate effort estimation than lower priority items because lower priority items might change before being developed. Importantly, expected effort of a user story done by the team carrying out the actual development is the most relevant (Cohn, 2006).

Generally, effort estimation is an important parameter in planning because it measures the total size of a release. Inaccuracies in size estimations of user stories lead to time and cost overruns.

#### *2.4.3.1. Deriving an Estimate*

Estimation can be done using a certain set of techniques such as expert opinion, analogy and disaggregation (Cohn, 2006). Expert opinion is based on the experience and is an opinion-based approach. Asking an expert may be helpful, but in agile projects developing functionality happens in a team. Thus, teams who have the most experience in similar tasks would make the most accurate intuitive estimates. The analogy technique is an alternative to expert opinion and assumes comparing the story size being estimated with relative size of other stories, for example this story is approximately twice bigger than the previous story. This method is useful because estimating relative size is easier than estimating absolute size (Cohn, 2006). It may be helpful, to estimate the smallest and biggest user stories first in order to select the range. Disaggregation refers to simplifying estimation by splitting a story into smaller tasks. It is difficult to make accurate estimates for large figures, thus big user stories could be broken down into smaller tasks make estimation easier.

The methods suggested can be used separately or be combined to maximize the accuracy of effort estimated.

#### *2.4.3.2. Planning poker*

Planning poker is a technique of combining effort estimates introduced by Grenning in 2002. All team members participate in planning poker. In case of multiple teams, each team will estimate independently user stories assigned to them. The product owner participates in the game but does not estimate. This is a relatively new lightweight technique with face-to-face interaction and discussions.

In the beginning a deck of cards is given to each team member. The product owner reads a user story and then there is a discussion between the participants clarifying the requirements. After that the developers write their own estimate on the paper, but not discussing it with other participants. If there is an agreement, the estimate is recorded and the discussion moves to the next story. However, if there is a disagreement, the discussion goes on in an attempt to clarify the differences and come to a consensus (Buglione, 2007). This technique has been compared to the unstructured group estimation, when group members have a discussion about user stores where decision is made at the end. The results showed that planning poker provided more accurate estimates for familiar tasks while the opposite was found for unfamiliar tasks. Further, results from unstructured group estimates were more realistic than individual estimates mainly because of increased task awareness after discussion and identification of additional activities (Moløkken-Østvold and Jørgensen, 2004).

Planning poker is a powerful estimation technique because it brings multiple experts together to share their opinions and who will eventually contribute to those user stories. Justifying the estimates decreases uncertainty and combining the individual estimations forms a good average realistic figure. However, it should be kept in mind that more time and effort in estimation may not necessarily increase the accuracy of the estimations. Thus, the actual benefit of planning poker is difficult to measure. Additionally, in some occasions it may not be known who will eventually carry out the task (Moløkken-Østvold and Jørgensen, 2004).

#### *2.4.3.3. Re-estimating*

Re-estimation is a common issue, which arises after the initial effort estimation. Re-estimation is needed when the initial effort was poorly estimated. However, since the effort measures the relative size of a user story, longer implementation does not necessarily mean that the size has changed. Thus, re-estimation is only needed when the size estimate of a user story appeared to be relatively bigger than other user stories' sizes. In other words, re-estimation of an effort for one user story should not cause other stories to be re-estimated as well.. Also, re-estimation may be carried out for partially completed stories in cases when the story cannot be completed in the next sprint. However, partial credit is not

generally recommended and most development teams count only if a user story is fully done (Cohn, 2006).

Re-estimating is important only for obvious cases to correct the consistency of the estimates. One should rather observe and learn from mistakes to improve estimates in the future.

#### *2.4.3.4. Areas for improvements in effort estimation*

Effort estimation may prove to be difficult and the benefits can be minor compared to the invested resources. The following suggestions offered by Cohn (2006) are useful to deal in situations where estimating the effort was difficult.

Time spent on initial estimation can be decreased. Instead, more time can be allocated to the feedback about the accuracy of the estimate and based on the feedback increasing the frequency of estimating (Cohn, 2006). For example, unfamiliar and low priority features' efforts can be estimated later as the team goes on with the project. In case the requirements are unclear, teams might also consider carrying out multiple estimates. The developers need to clarify all details about unfamiliar features before doing the detailed estimations.

Estimates can be validated and standardized by comparing them with the estimates of similar tasks and past experience, using simple rules and intuitive decision-making (Larman, 2006). The agile principles reply on self-organizing teams as well as learning from feedback and review sessions.

### 2.4.4. Release tracking

Scrum methodology emphasizes the importance to monitor the progress of the project. Tracking is needed to see whether the project is proceeding according to the plan (Cohn, 2006). The progress is tracked and reported through gathering suitable data and visualized by graphs such as release burn down charts, effort development charts, cumulative flow diagram etc.

Collecting the important data and information on the release status takes place in the end of each sprint. There are many existing metrics, but with agile practices "just enough" approach is recommended. Cohn (2006) recommends to use only a limited number used

for planning and tracking. This section will cover the current methods used in tracking and visualization of the data.

#### *2.4.4.1. Metrics*

As discussed earlier, velocity is the most important metric used in agile release planning. Planned velocity is tracked against actual velocity to see whether a release is progressing according to the forecast. Other useful measures and data include (Larman, 2006).

- The number of story points completed
- Total story points in release
- Story point start and end sprints
- Size of each user story
- Value of each user story

The number of story points completed shows the updated amount of work done for a given release. At the end of each sprint the number of story points completed shows the actual team velocity for the given sprint and thus allows to track planned versus actual velocity. Further, changes to velocity estimates should be tracked and updated. For either release or iteration plan, there must be a specified milestone criteria which tells the conditions of satisfaction of the task, i.e. defining the status "done" (Larman, 2006). The criteria should be tolerated because a completed story should potentially be ready for delivery and should not require any additional work effort.

The total number of story points in the release is needed because it allows tracking the actual size of the release. As the requirements change constantly the total size of the release needs to be tracked against the deadline. With fixed velocity, an increase in the total number of story points will postpone the release date.

The size of each user story can be verified by the duration. For example, a user story lasting 2 sprints should be twice the size of a user story, which took one sprint to be completed. Because in the product backlog the user stories are prioritized, the value of each user story needs to be tracked to adapt and to deliver the highest value to the customer.

#### 2.4.4.2. Visuals

Visualizing data and project status is helpful because it is more effective in interpreting data than numbers. One of the most important graphs in agile projects is the burn down chart. The burn down chart reports how much work is left and identifies at what stage the project currently is and whether it has progressed at a constant rate, i.e. it represents the planned and actual velocity. The burn-down charts are illustrated in the *Figure 6* below.

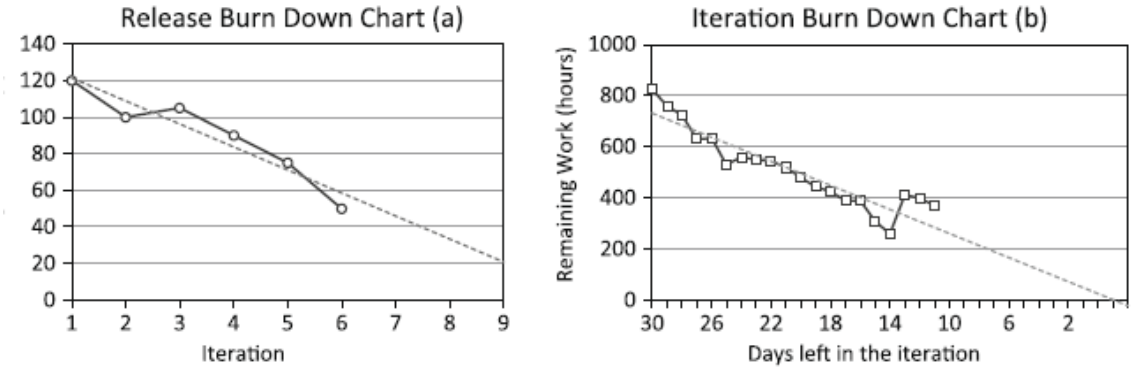


*Figure 6. Release Burn Down Chart (a) and Iteration Burn Down Chart (b) (Miranda & Bourque, 2009)*

*The release burn down chart* is used to monitor and report the progress of the project. It has two indicators: the overall rate of progress and the amount of work remaining. The rate of progress allows forecasting the time of completion. *The sprint (iteration) burn down chart* is derived from the task board information and shows the amount of hours versus days remaining for the sprint, showing whether all of the work of the iteration could be completed on time with the current pace (Miranda & Bourque, 2009). Both charts are updated as soon as new data is available, usually in the end of each sprint.

Another useful visual is effort development. It is a histogram illustrating the project or the release development status illustrated in Figure 7 below.

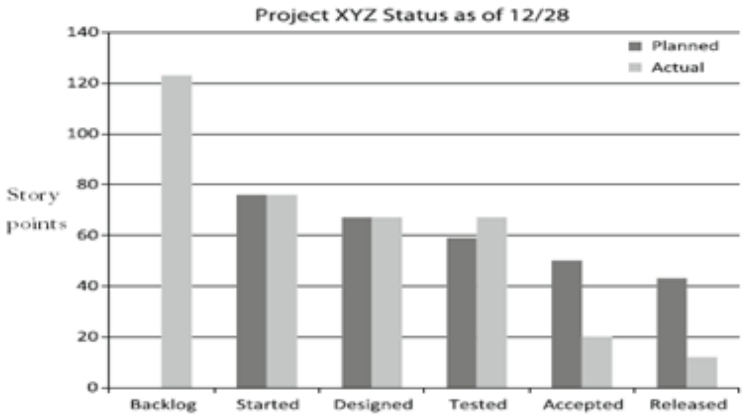


*Figure 7. Effort development graph showing (Miranda & Bourque, 2009)*

The purpose is this graph is to ensure that the planned amount of user stories matches the actual amount completed to ensure that features are developed at a pace that allows an even flow and right speed to meet the goals set in the plan (Miranda & Bourque, 2009). On the sprint level, visibility of planned versus actual story point completion over sprints on different levels allows to identify feature development progress on different levels and to identify current results. Such graphs may also be combined with the burn-down charts to support factual decision-making. This is a good way to see whether in a certain sprint plans were met on different stages of development.

Cumulative flow diagram illustrates the status of all user stories for the release over the time of development, as illustrated in Figure 8 below (Andreson, 2004).

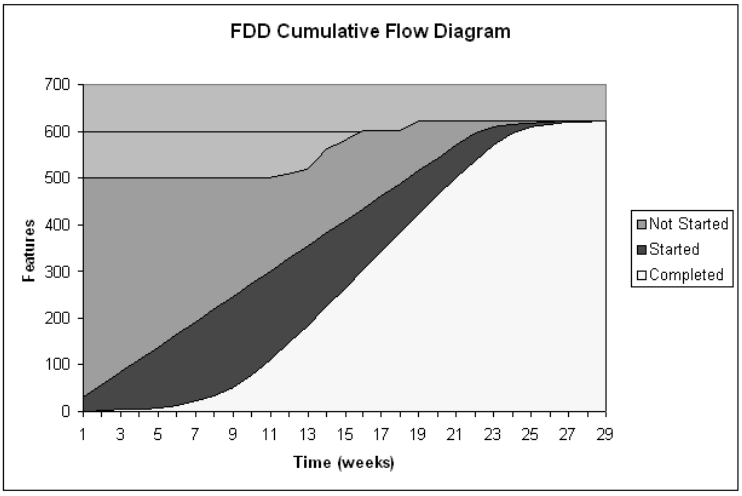


*Figure 8. Cumulative flow diagram (Anderson, 2004)*

The following graph is very useful to see how many user stories are “Done”, “In Progress” or “Not Started” and thus is a strong management tool for tracking the release

development. Unlike the burn down chart, the cumulative flow chart also graphically shows how much work is in progress. This tool is especially powerful to track and correct situations when too many user stories are "in progress". In some projects this can be a significant share, which would not have been presented otherwise.

The release tracking tools, which were presented in this section, support managers to monitor that the development proceeds according to the plan and when needed take corrective actions.

### 2.4.5. Multi-team agile environment

Agile methodology was initially designed for small teams with strict constraints regarding team size, location, presence of customer, informal communication etc. (Larman, 2006). However, some projects are too big for a one small team. Therefore, while following the agile principles and keeping team size small, multiple teams can be formed to work on the same product backlog instead. Depending on the company and the type of software there can be additional roles such as a software architect or a usability expert. They don't belong to any one specific team but have highly specialized skills and may act on various tasks, e.g. architects may act as high-level technical experts and estimate the total effort of the project to assist management in investment decisions (Heikkilä et al., 2010). Multi-team agile release planning and tracking have certain extraordinary challenges and methods. In this section, I intend to cover these differences and provide advices how to improve the reliability of planning and tracking processes under multi-team environment.

#### *2.4.5.1. Multi-team agile planning*

In a multi-team agile environment planning becomes more challenging and needs certain techniques to be incorporated. These techniques establish a common basis for estimates, adding detail to their user stories sooner, performing look-ahead planning and incorporating feeding buffers (Cohn, 2006). I will cover each of these techniques below.

*Establishing a common basis for estimates*

Even though each user story needs to be estimated by only one team, the estimates need to be equivalent and comparable across different teams. Cohn suggests two ways how a common basis can be established for different teams. In case the teams have worked

together before, they can meet and discuss the estimates of some user stories from the past and agree upon the estimates for them (Cohn, 2006). The teams need to identify stories with size of one story point and two story points. Once these baseline stories have been identified, the teams would be able to estimate new stories based on the analogy technique discussed earlier. Also, the teams should periodically verify the common baseline by randomly checking upon some user story estimates.

*Adding detail to user stories sooner*

Under multi-team agile development, the user story requirements need more definition before the start of the sprint as compared to a single team project. This includes clearly defining the conditions of satisfaction for the user stories, i.e. "definition of done". The developers are responsible for communicating user story requirements with members of the other development teams instead of asking about them indirectly from the team's product owner (Heikkilä et al., 2010). Additional details about requirements are needed in order to coordinate work across teams better and to see interdependences between different requirements.

*Visual traceability matrix*

The overall progress should be tracked and communicated in such way that all participants could see the dependencies of tasks. Heikkilä et al. (2010) suggests to use the traceability matrix which is a visual planning board showing interdependencies between user stories.

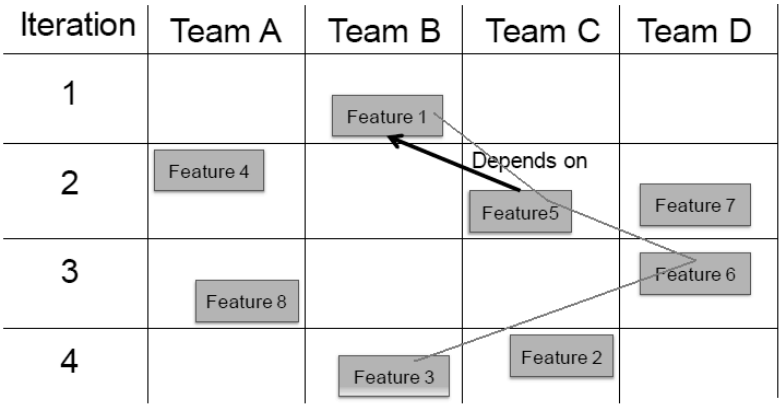


*Figure 10. Visual traceability matrix (Heikkilä et al., 2010)*

The board shows estimated start and end points of user stories. When multiple teams work on a release, they might face interdependencies (Heikkilä, 2010). These interdependencies between user stories need to be taken into account. Further, this visual can be very helpful in identifying the critical path and later tracking the project according to it. The critical path shows the earliest possible completion route illustrated as a line going through the features 1, 5, 6 and 3 in *Figure 9.*

*Look-ahead planning*

With look-ahead planning teams coordinate work in a span of a few sprints, where the release plan serves as the basis for that span. In the beginning of every sprint, teams meet to share information and commit to tasks they can complete. The development teams should also make sure that any progress information is up to date and that their team's velocity for the coming sprints have been calculated, taking into account any irregularities such as vacations or other planned absences of team members (Heikkilä, 2010).

*Incorporating feeding buffers into the plan*

Feeding buffers are also used not only in situations with interdependencies, but which are too complex to use look-ahead planning. If reducing interdependencies is not possible, a feeding buffer needs to be inserted into the sprint to protect on-time delivery (Cohn, 2006). This can be done by deliberately underestimating velocity for the developing team in the given sprint. However, one should keep in mind that adding feeding buffers will naturally extend the expected duration of the project in a reasonable manner.

Feeding buffers should be allocated only between the critical dependencies, so allocation of user stories among teams and sprint is a necessary pre-condition. Buffer feeding is not needed if the other team is able to work on another valuable feature or with partial deliverable. In sizing the buffer feeding, it is essential to follow incremental delivery, so it should not exceed from 50% to 100% of sprint length per one feed (Cohn, 2006).

Teams can also deal with interdependences by prioritizing the user story development order. For example, a less important story can be scheduled earlier than a more important story if another team's very important feature has a dependency to the less important story

(Heikkilä, 2010). Dependencies can be traced using the unique user story identifiers which need to be included in the product backlog.

#### 2.4.5.2. Multi-team agile release tracking

Release tracking in a multi-team environment follows the same principles as explained previously for the single-team release tracking. As discussed earlier, after the development process is initiated, the progress will at some point deviate from the initial plan. Monitoring the progress of the release in a multi-team project requires more factors to be considered and the status information must be kept updated after every sprint and the scope of the release updated accordingly. The metrics, such as size and availability need to be aggregated from across the teams (Cohn, 2006). Different teams have their own velocities, thus when tracking the overall progress of the release, each team's speed needs to be considered separately. Additionally, since there are a lot more requirements per release, it is harder to track the development progress accurately causing uncertainty of estimations in planning. More resources need to be spent on communicating the release status and the organization of sprint meetings between the teams and the management, especially if the teams are in different locations. According to Cohn (2006), in a multi-team development environment effort estimations need to have a common baseline and metrics across all teams.

Inter-dependencies between teams may cause bottlenecks and affect the schedule of the whole release, so it is important to monitor activities of different teams and decrease the risk with inserting the feeding buffers. Cohn suggests to use traceability matrix discussed earlier to track and eliminate potential bottlenecks. In more complicated projects with many interdependencies planning activities may become extremely complicated and thus agile management tools are deployed to support planning and tracking activities.

#### 2.4.5.3. Challenges

Similar to a single team release planning, in a multi-team environment the idea is to gather all development teams to perform the release planning together. However, multi-team release planning becomes more challenging due to increased number of people, more complex user stories and a network of dependencies between the requirements (Heikkilä

et al., 2010). This also affects the implementation order of the requirements. As discussed in the SWOT analysis, globally distributed development teams is a weakness in a multi-team agile environment because communication between the management and development becomes more challenging. Release planning and tracking require more complicated and rigid methods to be used. Since in large development projects the complexity of the tasks grows with the increase in project size, teams stop to be completely multifunctional and different teams and team members specialize in certain tasks. Additionally, with human-centric agile methods it may become a challenging task to plan activities for teams who have, among other factors, different working culture, experience in agile processes and technical expertise. This again complicates the planning of iterations and scheduling. Team effort estimates need to have common basis across teams and the planned activities are followed.

In addition to multiple teams, most often multitasking is almost unavoidable in larger companies. In situations where teams develop multiple releases simultaneously, additional metrics are needed in order to plan and monitor each release separately. It must be kept in mind that the velocity for each release will be slower because the total velocity of each team will be split. Also, when teams develop multiple releases it can be challenging to plan the distribution of resources and task selection. Compared to a basic Scrum model, picking tasks from multiple releases is more difficult. Thus, teams' actual proportion of effort spent on each separate release may differ from the estimations significantly, making planning and tracking even more challenging than it already is (Vähäniitty et al., 2010).

Common basis for effort estimation, correct forecasts of teams' velocities tagged to actual proportion of effort spent on each release within the sprint - become the center tasks in multi-team multiple-release planning and tracking. Also, prioritization and distribution of effort should be balanced correctly for successful development of all releases.

Based on the literature review, planning and tracking in multi-team agile release development environment is more challenging compared to a single-team agile release development scenario, and requires taking additional factors into account. An in-depth literature review sets a solid base for further analysis of parameters needed in building a tool to optimally plan releases.

# Analytical Study

## 3. Case Study: Nokia Siemens Networks

Nokia Siemens Networks (NSN) is a global company in the telecommunications industry with a total revenue exceeding 12,7 billion Euros in 2010. It was formed as a result of a joint venture between Nokia and Siemens, which began its full operations in April 2007.

My thesis work at Nokia Siemens Networks was based on the case study of Flexi software product development. Flexi is an LTE (Long Term Evolution) gateway product in mobile broadband technology, offering efficient management of mobile network traffic to customers, which includes over 50 large mobile operators around the world. Software is developed in continuous releases to match customer needs with improved or new functionality.

Flexi software development takes place under the Scrum methodology. The company has adopted agile methods in 2008 to mitigate high risks and business pressure, as well as to make development more efficient. However, technical complexity of product characteristics together with a globally distributed multi-team development structure has caused difficulties in planning and delivering agreed functionalities within the expected time frame.

This thesis work was facilitated by the need for more structured agile release planning and development methods in the case company. My main task was to analyze existing agile methods applied in Flexi, to identify problematic issues and to provide recommendations on how to optimize release planning with agile methodologies. Based on the results, my responsibility was also to create new product backlog and management tools in Excel to support more structured planning, tracking and development methods.

Through Flexi case study, this thesis attempts to identify existing issues and to provide practical recommendations on optimizing agile project management in large and complex software development projects. The study provides empirical findings about challenges and methods of scaling agile methods in large multi-team agile projects.

In this section, Scrum methods applied in the case company will we discussed to the extent required for understanding of existing problems and setting the grounds for further analysis. This section will be concluded with improvement propositions to optimize agile project management methods and will be further extended with practical part, where the results will be applied to create the required management tools for the organization.

## 3.1. Agile Software Development in Flexi

This section will cover Scrum methods applied in Flexi software development and will identify the main challenges that the company faced as a result of quickly growing size and complexity of software development.

### 3.1.1. Release development under Scrum

Like most companies developing software under agile methods, NSN adopted Scrum methodologies for the following common reasons.

- High risks from the technology and requirement complexity point of views
- High business pressure to get early results out of the development process.
- Combining knowledge of engineers with different backgrounds

In Flexi, Scrum follows general principles and values of agile development processes with the emphasis on the importance of common practices. Software development takes place in continuous release cycles in respond to changes in the customer market through delivery of the new functionality. The content on each subsequent release includes components from the previous release, together with new and updated features. Feature development further takes place in an iterative mode over continuous sprint cycles, as illustrated in *Figure 11* below.

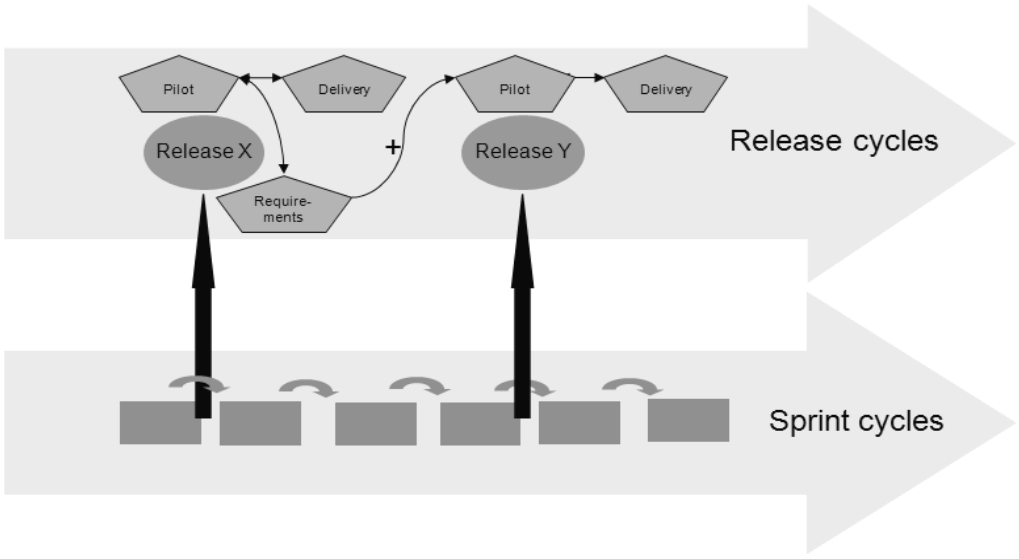


*Figure 11. Continuous release and sprint cycles*

Release planning and an overview of release features is done several months prior to the main development period of a release. Since the content of each release is limited, only the most important features are developed in the upcoming release. New profitable features are identified through preliminary evaluation of effort, market value and price, and may replace old functionality or form new areas. Requirements are managed in a common list called a product backlog. When new requirements are identified and approved, release content and delivery date are fixed prior to the main development period.

The development follows agile methods and takes place in cycles, known as sprints. In Flexi development, sprint dates are synchronized across all development and the sprint duration equals to two weeks (14 calendar days). During every sprint, daily Scrum meetings take place within each team, where members briefly share their achievements and plans. In the end of every sprint, there are regular sprint review, sprint planning and retrospective meetings to exchange information on the things that went well or need improvements based on the previous sprints. Meetings last approximately 2 hours, where each team delegates their achievements and goals to the management.

In sprint planning, teams meet in workshops and break user stories into smaller fragments called tasks. Fragmentation into tasks takes place on the technical level and the development takes place with a black box approach. Team members manage tasks and

upon completion tasks are combined back into the user story. A user story is completed when it meets all requirements of the "Definition of Done", including implementation, customer documentation, testing and review. A release is completed when all required user stories are developed and approved with successful testing and documentation.

All deliverables are stored in one place and several practices are deployed to enable frequent and systematic communication. Technology is successfully utilized to communicate and share data across different sites, basically eliminating the need for physical presence.

### 3.1.2. Scrum teams

While teams are located in multiple sites, most experienced development teams and management are located in the same site. This form of distribution makes the main site responsible for most complex and interdependent activities, whereas other sites carry a supportive role in development. Therefore, while distributed development structure naturally poses challenges in knowledge sharing, as well as cultural and organizational differences, it does not affect development significantly.

Further, due to the nature of the product, in addition to traditional agile Feature Development teams, also other team categories exist performing distinct activities, including feature testing, verification, architectural and release activities. Each team category performs unique activities in a specific order. Feature Development teams take a major part in developing new features in the form of user stories. When features are completed, they are passed to Performance Testing (PET) and Network Verification (NEVE) teams, who consequently test performance and compatibility of the completed features. Architecture teams are responsible for integrating features into the program and finalizing the release package. Different team categories exist due to the complex nature of the product, and while activities of feature development teams follow routine Scrum development methods, activities of the other teams are quite unique and result from feature teams' work. While interdependencies across teams are not significant, the main challenges result in planning activities of feature development.

Since the development progress mainly depends on the work of the feature teams, the focus of this thesis in on improving the release-planning activities in the following team

category. The results, however, should also be beneficial in improving the performance of other team categories given their development practices also follow Scrum methodologies.

Rapid growth in the Flexi development environment between 2008 and 2010 caused significant changes in the team structure. Apart from distributed multi-team development environment, various software components belong to technically different and complex functional areas. Initially, the product has been smaller and few cross-functional teams developed requirements across different functional areas. However, as the size and complexity of the software increased, input of each team became more limited. Also, development in each area currently requires distinct set of skills and expertise. Thus, the overall development currently results from the collective contribution of different teams working on small limited parts of the release. Next, I will explain the structure of the Flexi software consisting of separate functional areas.

### 3.1.3. Functional areas

Flexi requirements belong to diverse functional areas called *epics*. Epics are based on different complex codes and the development of software components within each area requires specialized knowledge. Some areas are generally larger than others. Additionally, due to rapid innovations in the telecommunications industry, also new areas may emerge and old ones may become obsolete.

Since epics have quite different technical characteristics, each area requires specific knowledge and experience. Given the size and complexity of the program, having fully cross-functional teams proved to be unfeasible. Therefore, teams are assigned to certain areas based on their skills.

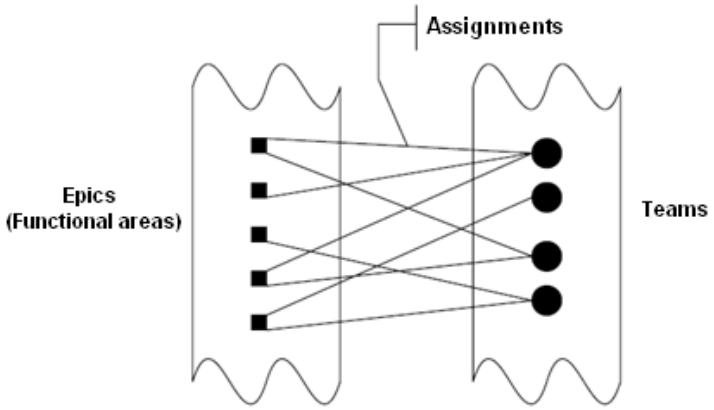


*Figure 12. Epics and teams relation*

As one can see from Figure 12, team knowledge is limited to certain areas. While knowledge of some teams may overlap in some areas, the expertise level of each team differs. As a result, currently the total effort in each area is distributed unevenly across teams and functional areas, as illustrated in Figure 13 below.

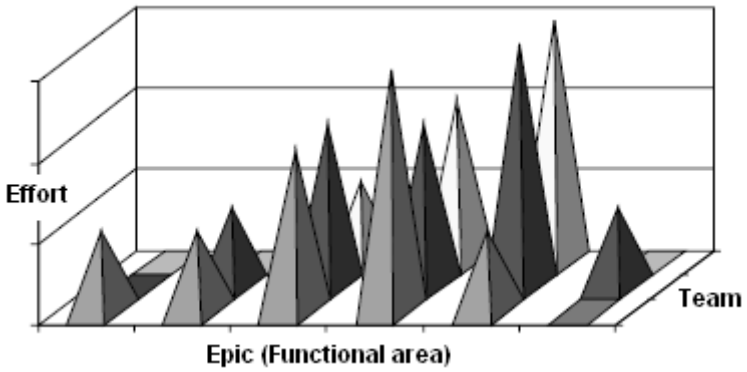


*Figure 13. Required effort of different teams within different functional areas*

Initially the structure of the development processes has been simpler, a few cross-functional teams were located in one site and user stories were re-factored to fit into one sprint. Epics were not considered important in release planning and development, and the aim was to develop team expertise across functional areas gradually. However, the rapid growth of the development structure offset the initial setting, in which all teams could develop functionality across all areas, resulting in new issues and challenges. The complexity of requirements in different areas required higher expertise levels from new teams and longer time to develop. As a result, user stories were allocated to teams based on expectations about requirements complexity level and team expertise in the area.

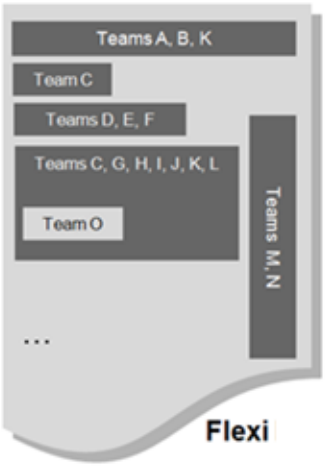


*Figure 14. Team contribution in Flexi release development*

Further, raising the overall level of expertise of teams across multiple functional areas efficiently proved to be challenging and most teams were assigned to have a core functional area as their prior responsibility, but would also have limited expertise in few other functional areas to have the supportive role and to allow certain level of flexibility.

As stated earlier, in Flexi there is common list of requirements, which is managed in the Product Backlog.

### 3.1.4. Product Backlog

In Scrum, the product backlog is the most important artifact, which is used extensively for managing requirements, planning and tracking development. The product backlog contains a comprehensive list of completed, ongoing or planned user stories, which would be developed in the feasible future.

In general, the structure of the product backlog should be simple and contain all data needed for planning and tracking of releases. Too little information in the product backlog may limit the usability of the data and thus affect the accuracy of release planning.

The content of the existing product backlog of Flexi was found to be quite comprehensive and it has been extensively monitored throughout the development. The summary and structure of the current product backlog content is presented in Table 1.

| Data | Content Definition |
|---|---|
| **Req. ID** | Feature or User Story ID |
| **Target Release** | Identifier of intended target release |
| **Compulsory** | Is the US compulsory or optional content of a release? |
| **Sprint Start** | In which Sprint a story started |
| **Sprint End** | In which Sprint a story will be/was completed |
| **Epic** | Functional area a story belongs to |
| **User Story** | User story itself<br>As a <Type of User>, I want to <goal> so that/because <reason> |
| **Link to Wiki** | Link to additional information on the story |
| **Feature value** | Story priority in monthly packages |
| **Lead Customer** | Key customer for whom the user story is intended |
| **Status** | User Story status (Completed, Ongoing, Not started, Canceled, Re-factored) |
| **Team** | Allocated Team |
| **Size** | Effort in story points |
| **Comments** | Additional notes are provided optionally |

*Table 1. Current Product Backlog structure*

The data included complete information about each user story including starting and ending sprints, epic, story description, team assigned and effort estimation. In Flexi, agile release planning and tracking was mostly based on the product backlog data.

In addition to the product backlog, a Web 2.0 portal has been used for storing and sharing more detailed information related to user stories and release development. Data included a comprehensive description of user story content, technical documentation, as well as retrospectives and other useful information updated by teams and management.

## 3.2. Release planning in Flexi

In Flexi, prior to the main development period, release lifecycle includes various preparatory activities, including planning, documenting and scheduling. Scrum methods are applied during the main feature development period, lasting approximately 3 months. The aim of this section is to discuss current release planning and tracking methods, as well as to identify main problems in applying Scrum due to rapid up-scaling of the development environment.

### 3.2.1. Milestone activities

In Flexi, the main planning activities take place before the actual development begins based on pre-defined milestone criteria. Each milestone is accompanied with a set of activities, which need to be completed before a milestone is declared as completed.

Since the focus of my study is on planning activities of feature development, the main activities related to planning need to be clarified to the reader. While release lifecycle begins with initiated need for new features, actual preparations for development begin during an M1 milestone. The key activities related to detailed planning and scheduling of development include:

- Estimating initial effort of features (size)
- Re-factoring features into user stories
- Assigning user stories to teams
- Implementing key parts of architecture
- Filling Product Backlog with user stories

Table 2 below summarizes the activities, which are in the zone of interest for this study. The content has been modified and irrelevant data removed for confidential purposes.

| Milestone | Main activities |
|---|---|
| **M1** | **Project Plan Confirmed**<br>- Delivery content verification<br>- Re-factoring features into user stories and updating to Product Backlog<br>- Resource allocation and scheduling up to delivery date (M4)<br>- Assigning user stories to teams<br>- Implementing key parts of architecture<br>- Customer documentation<br>- Product Backlog complete including team assignments and effort estimates<br>➢ *Commitment to delivery date* |
| | **Main Feature Development period**<br>- Development of user stories<br>- Maintenance<br>➢ *Measure of velocity* |
| **M2** | **Development completed** |
| **M3** | **Network Verification completed** |
| **M4** | **Ready for Delivery** |

*Table 2. Summary of milestone activities*

The M1 milestone is achieved after commitment to the release date and content are agreed, and deadlines for milestones are set up until milestone M4 when the release should be ready for delivery. The main feature development period begins right after M1 and lasts until all required features have been developed and tested successfully by M2. Further, the feature development period partially overlaps with Performance Testing and Network Verification activities. Since the latter activities depend on readiness of feature development teams, the main testing period begins when substantial amount of the user stories have been completed and lasts until the deadline for milestone M3. After the new content has been verified with a pilot customer, the release is finalized and prepared for delivery on M4.

The delivery date and release content are fixed already at M1, prior to the main development period, due to high pressure from the market and the need for rapid development. Therefore, inaccuracies in planning put threat on delays and may result in bottlenecks during the remaining release development lifecycle.

In agile methodologies, planning is based on effort estimations and team velocities. Therefore, current practices of estimating effort and calculating velocity in Flexi need to be discussed in more detail.

### 3.2.2. Effort estimations

In Scrum, effort estimations provide information about the relative size of the requirements. Given that the requirements and the development environment are similar to the past, an expected duration to complete a new release can be calculated from the estimated size of the requirements and the average velocity of the team from the past.

Under the current organizational structure, effort estimations are done on two levels: feature level and user story level. Effort estimations are done on a high feature level prior to the main development period due to the product and the environment specificities:

- ✓ Feasibility of a release content needs to be evaluated before the main development
- ✓ Commitment to deadlines has to be done early in the release development cycle

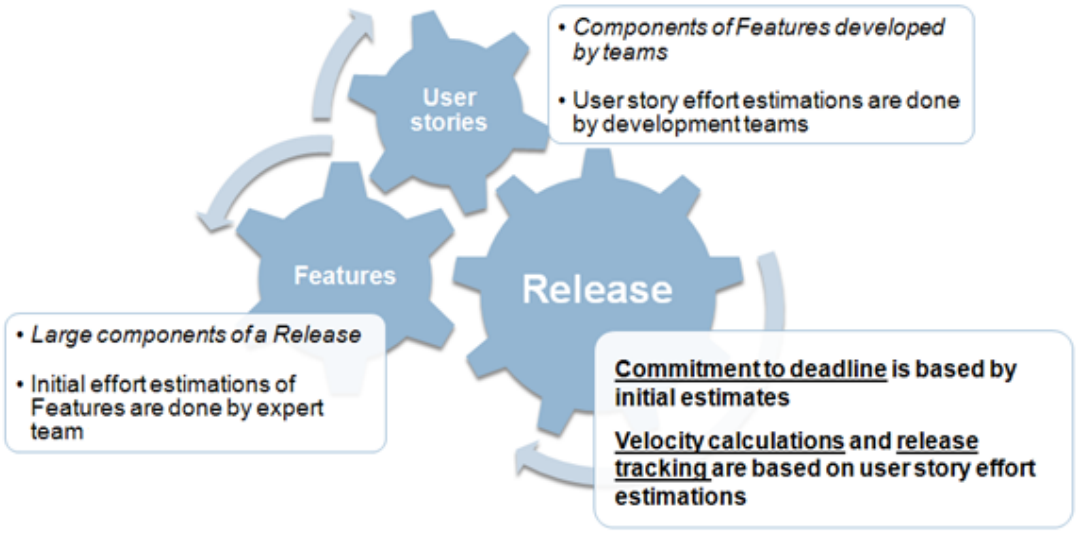


*Figure 15. Release and effort estimations*

In order to make initial effort estimations, the expert architectural team explores the content of high-level features. Despite strong expertise of the team, the accuracy of initial effort estimations depends on the complexity and uniqueness of the requirements, as well as quality of the feasibility studies. In Scrum, initial estimations are not required to be precise and these natural inaccuracies should be considered normal.

More accurate size estimation of requirements is possible during the actual development. Therefore, as features are re-factored into user-stories, development teams make effort estimations after brainstorming user stories and requirements more thoroughly. It was assumed that if feature requirements changed during the main

development period or if initial estimations were incorrect, development teams' effort estimations on user story level would correct inaccuracies in initial estimations.

However, even though requirements did not change, effort estimations done by development teams were found to be inconsistent. It was seen that methods to manage and control effort estimations on two levels were insufficient and inaccuracies in estimations could not be separated from changing requirements. An example is illustrated in Figure 16 below.

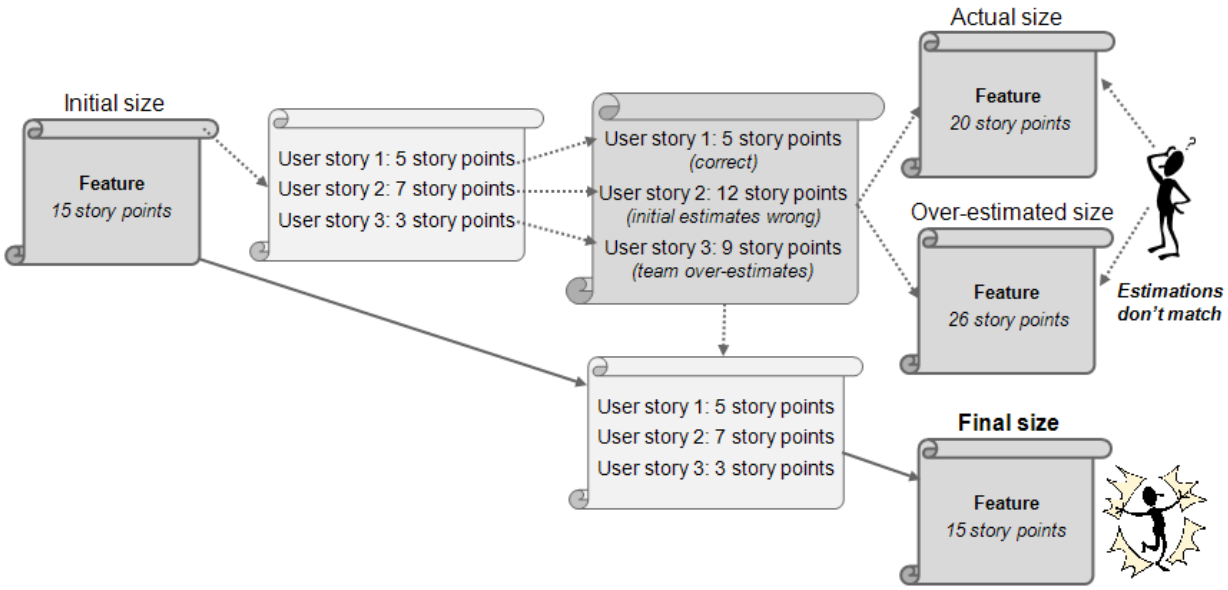


*Figure 16. Inaccuracies in effort estimations*

To keep initial estimations of release content and schedule under control, during development size estimations of user stories within each feature were not allowed to exceed initial feature estimations. While keeping user story effort estimations within limits, inaccuracies in initially under-estimated content (especially complex and unfamiliar features), as well as changes in requirements and team assignments, caused complications in planning and tracking release progress. Complexities also threatened testing and piloting deadlines, thus putting pressure on in-time delivery of releases.

Problems with the accuracy of effort estimates turned out to be only partial problem. While critical path of feature development could be estimated based on the major features of a release and teams assigned, additional problems arose from inaccuracies in planning of the overall development progress. Since in Scrum the effort estimations provide

information about the relative size of requirements, planning is based on the accuracy of past effort estimations and the accuracy of velocity calculations.

### 3.2.3. Velocity

In Scrum, velocity is a measure, which tells the average amount of story points developed per sprint. Velocity calculations are based on the historical data, i.e. story points completed in the past. Since effort estimations are the main component of velocity, inaccuracies in the prior naturally cause problems with the latter. However, apart from effort estimations, the methods applied to calculate velocity are also important. Having discussed effort estimations, the aim of this section is to discuss the assumptions and methods currently used in Flexi to calculate velocity.

Until now, calculations in Flexi have been based on the total amount of story points completed by all feature development teams over each sprint. Since most user stories last multiple sprints, only some teams complete user stories in a given sprint, as is illustrated in Table 3 below.

| Feature Development teams | | | | | | |
|---|---|---|---|---|---|---|
| **Sprint** | | **Team A** | **Team B** | **…** | **Team Z** | **Velocity** |
| **1** | | 0 | 0 | … | 10 | **73** |
| **4** | | 13 | 0 | … | 0 | **61** |
| **5** | | 0 | 8 | … | 0 | **105** |
| **6** | | 0 | 0 | … | 0 | **52** |
| **7** | | 47 | 13 | … | 0 | **121** |
| **8** | | 0 | 21 | … | 0 | **160** |

*Table 3: Current methods to calculate sprint velocity.*

While duration varies across teams depending on the estimated size and complexity of user stories, it has been assumed that, on average, the differences across teams and user stories would cancel out. As a result, completed story points in each sprint were added to calculate overall sprint velocity for feature teams. Further, moving average velocity and control limits were calculated and applied in planning given high uncertainty and were supported by rational judgment.

Additionally, some team-specific data was calculated separately, including

- *Average duration of user story completion,* showing how many sprints it took on average for each team to complete user stories.

- *Remaining number of story points in each release* showing the total amount of release-specific story points either ongoing or not started.

These metrics were used to assist specialists with decision-making together with knowledge about team general performance, release components on technical level and information gathered during sprint meetings.

Generally, in software release development under Scrum, re-planning needs to be done after every sprint based on new information about velocity and burn-down of story points as compared to initial plans. Visuals and tools should facilitate factual decision-making, especially when the development environment is large and complex.

### 3.2.4. Release planning methods

Components of Flexi software require expertise knowledge across different functional areas. In addition to complexity, product size is large and requires multiple teams to work on development simultaneously. Teams specialize in specific functional areas and overall development is based on compound effort from multiple teams having different areas of expertise. Requirements within each functional area vary in complexity and more complex user stories take longer time to develop and require higher expertise in the areas. Planning new features is less accurate prone to unfamiliar content. Additionally, during development certain tasks affect velocity and duration, e.g. maintenance work and possible changes in requirements.

While complexity of Flexi development environment and high pressure from the market put limitations on the accuracy of planning, problems also resulted from insufficient Scrum methods and practices of release planning.

Due to inaccuracies in effort estimations and velocity calculations, decisions related to release planning and development are strongly based on the expertise knowledge of key specialists the content of critical features and earlier performance and skills set of experienced teams. While based on the knowledge, realistic duration of critical path features is estimated, decisions are hard to support by facts and figures.

Having estimated the duration of the critical path separately, development of the remaining features is planned based on the total effort required and average velocity of

feature development teams, i.e. burn-down of release is based on historical data about average total size completed per sprint.

Inaccurate effort estimations and velocity expectations put pressure on planning future releases. Also, treating different teams and functional areas as one caused lack of visibility on release development progress. Based on the discussions with the key personnel involved in release planning and development, the general opinion has been that "*existing velocity was too optimistic, especially in the Feature Development"* (interviews).

Software planning and development under Scrum in small projects is relatively simple. However, agile planning and development require a different approach in large and complex development environments. While agile practices favor the lack of detailed planning, the methods how planning should be carried out are commonly misunderstood. In Scrum, planning and tracking development progress go hand-in-hand. As the development environment becomes more complex, the methods to plan and track development also become more challenging.

## 3.3. Analysis of current problems

Due to rapid growth of the Flexi development structure, NSN faced challenges with maintaining accuracy of release planning based on established agile planning methods. In this chapter, identified problems related to accuracy of release planning will be discussed. Further, data analysis will be carried out to recognize root causes of problems. "5 Whys" method will be used to support proposed improvement actions to identified root causes of problems in Flexi release planning.

### 3.3.1. Identified problems

In general, Scrum is more suitable and efficient method to support the development of technically complex software, especially compared to the waterfall model because it provides flexibility to changing requirements and facilitates faster feature development. Agile planning and tracking methods, however, need to be scaled when the development structure expands. In Flexi, growth of development environment caused multiple complexities and problems in planning and development efficiency. This section will

discuss identified issues affecting accuracy of planning and tracking, as well as efficiency of development. Also, hypothesis for further analysis will be set.

#### *3.3.1.1. Allocation of user stories*

In Scrum, planning is based on velocity of development teams and relative size of user stories. However, current allocation of user stories across teams complicates velocity calculations and decreases efficiency of development.

Flexi software is based on combination of different functional areas, which varying in complexity and required skills. Features are quite unique and requirements belong to one or more functional area. After features are re-factored into user stories, each story belongs to a certain functional area. Therefore, release content consists of a set of requirements in different functional areas. Some areas are larger and have more features than others.

Since release content is based on the total content of different functional areas, the critical path of a release should come from the duration to complete all requirements in each area. Since functional areas vary, different number of teams and velocity is needed in each area, depending on complexity of requirements and expertise level of teams. Currently, user stories within each functional area are assigned to multiple teams and each team is assigned stories from multiple areas.

Apart from developing within the core expertise area, teams also hold supportive role in other areas. The main idea behind current method of user story allocation has been focused on gradual increase of team competence across multiple functional areas. However, since each functional area requires different set of skills and team expertise varies in each area, such allocation of user stories complicates planning. With the existing allocation of user stories, combined average velocity of a team in a release depends on the effort assigned across multiple areas. As a result, estimating overall velocity to complete each area becomes challenging and inaccurate.

Since velocity depends on task complexity and expertise level of the team, more complex user stories will be developed slower. Further, because the expertise of the teams varies in each area, the complexity of tasks assigned across teams is not the same. Therefore, velocities of different teams cannot be compared or combined to estimate overall duration required to complete the total content of each functional area.

Also, current allocation of user stories poses additional challenges in planning and estimating the actual critical path. Release duration depends on complexity of tasks and load of different teams. While the critical path will depend on the total size and complexity of user stories assigned to the most loaded team, developing across multiple functional areas makes planning based on velocity more challenging.

Further, since teams generally develop across multiple functional areas, overall growth of expertise and efficiency is lower. While many teams can develop easy user stories in supportive areas, they are not competent enough to develop more complex requirements. Additionally, building the same level of expertise is not feasible across multiple areas due to overall program size and complexity. As a result, more complex requirements are assigned to the core team and narrow overall knowledge in a certain areas may cause bottlenecks.

In order to identify the current development structure and processes in more detail, differences across functional areas and teams need to be analyzed. Also, as outlined in earlier discussion about development structure, releases vary in content, i.e. each release contains a set of features belonging to different areas. This implies that the velocity in each functional area will vary depending on the release content, as well as other activities outside the main release. In order to identify the, release-specific content will be analyzed and compared with the overall development during the selected period.

Thus, the following hypotheses were formulated and are to be verified through more detailed analysis.

**Hypothesis 1.1: Functional areas vary and are developed by multiple teams**

**Hypothesis 1.2: Teams develop across multiple areas in different quantities**

**Hypothesis 1.3: Teams differ in structure and expertise**

**Hypothesis 1.4: Release content differs from overall development**

#### *3.3.1.2. Effort estimations*

In Scrum, effort estimations are the main metric used for calculating velocity. Therefore, accuracy of the prior naturally effects the latter. Accuracy of effort estimations depends on the size and complexity of the requirements, team expertise level in the area and earlier

experience with similar features. In the case company, problems with methods to manage effort estimations across multiple teams were identified.

As discussed earlier, while effort estimations are done on two levels, the consistency between initial and second-level estimations has been missing during the development. Lack of common agreement about effort estimations across teams resulted in safeguarding the initial estimations. Since initial effort estimations are done prior to the actual development, such estimations are inaccurate due to the size and complexity of estimated features. On the other hand, during development smaller user stories are work-shopped more thoroughly and thus secondary estimations should correct possible inaccuracies in initial estimations, given that teams have common baseline for estimations and agree with the requirements.

Even though feasibility studies have been carried out sufficiently to understand the requirements of features and to perform initial estimations, the content of some features could change during development for multiple reasons. Firstly, certain requirements could be missing from initial estimations, which were later identified by development teams. Secondly, some requirements could be added or excluded thus changing the planned release content. Additionally, the complexity of requirements could have been initially under-estimated. As a result, during development such requirements would be either be re-assigned to a more experienced team, or will have lower velocity than initially planned. While the actual size of release could differ from initial estimations, with existing methods changes in content were not visible due to lack of direct link between initial and final estimations during development.

In agile methods, re-planning after every sprint is essential to track changes in initial estimations and schedule. Re-allocation of stories to another team will affect the initial burn-down and may compromise the critical path. Thus, failing track changes in effort estimations on two levels could be compared to driving a car while looking in the rear mirror.

Based on investigation, the main problems with effort estimations were identified, including inconsistency of methods, as well as failure to distinguish changes in content and load of individual teams during the main development. Wrong expectations from the previous release also effected accuracy of estimations and planning of future releases.

Since it was identified that initial effort estimations were not tracked after features were re-factored, changes in content could not be compared. Questionnaire results will aim to identify issues related to metrics and methods of doing effort estimations across teams.

**Hypothesis 2.1: Effort estimations are misunderstood across teams**

#### *3.3.1.3. Velocity calculations*

In Scrum, velocity is the main metric used for planning duration of release development. Velocity measures the average amount of story points that a team can develop in one sprint based on previous sprints. Existing methods to calculate velocity were identified to be wrong for multiple reasons.

Firstly, as discussed earlier, functional areas and teams differ. Thus, team velocity depends on the complexity and size of requirements in different functional areas. Since teams also differ, the velocity also needs to be calculated separately for each team in each functional area individually.

Secondly, it was identified that most user stories don't fit into one sprint. For the reason that velocity measures the amount of completed story points per sprint, calculations should also include user story duration in sprints. In other words, if user stories last more than one sprint, completed size needs to be divided by the amount of sprints it lasted.

Developing multiple user stories with varying duration significantly affects the accuracy of team velocity. Thus, the duration of individual user stories needs to be considered within each team and the actual sprint velocity is more accurate when none of user stories are ongoing in parallel. Since the content of releases changes and multiple releases may overlap, keeping track of a single velocity figure for the overall development causes additional inaccuracies in planning specific features and future releases.

**Hypothesis 3.1: Team velocity varies across functional areas**

**Hypothesis 3.2: Total velocity differs across functional areas**

**Hypothesis 3.3: Teams don't know their velocity**

**Hypothesis 3.4: Release velocity differs from velocity in the overall development**

#### *3.3.1.4. Tracking release progress during development*

In Scrum, release planning and tracking go hand-in-hand with velocity, effort estimations and visibility of release progress against planned on the burn-down. Given complexities of the development environment and problems with current release planning methods and metrics, so far decisions have been mostly based on expert knowledge of specialists involved in planning. It was identified that current methods to track release progress lack sufficient visibility to support factual decision-making.

Firstly, while planning is based on initial estimations and tracking development progress is based on user story estimations done by the teams, existing tools don't support tracking changes to initial content.

Also, functional areas and expertise level of teams vary. However, the existing tools lack burn-down charts for individual teams and separate functional areas thus limiting the visibility of development progress. Since agile methods value flexibility and adaptability, changes in requirements and feature content during development result in the need for re-planning. Thus, in order to be able to carry out re-planning, it is important to see changes in initial and final content based on new information about effort estimations and velocity during development.

Based on the identified problems with methods and metrics, current tools need to be adjusted to support current development processes. Having identified and set hypothesis for further analysis of the development processes, the aim is to identify root causes of problems and to provide recommendations on improvement actions. Further, in the Practical Part of my thesis new tracking tools will be built to support improved release planning and development methods.

### 3.3.2. Analysis methods

In Scrum, effort estimations and velocity are the main components of release planning. Therefore, the main focus of my analysis will be related to factors, which affect these metrics. In addition to regular discussions with Flexi management and expert personnel, my analysis will be solidly based on the following two methods:

- Product Backlog data analysis
- Questionnaire across all development teams

Before proceeding to hypotheses formulation, I will discuss methods and data used in the analysis process.

#### 3.3.2.1. Product Backlog Analysis

As was discussed earlier, the Product Backlog is the main artifact under Scrum to support planning and tracking of development. Therefore, the analysis of historical data is an important part to identify and verify and illustrate root causes of problems in Flexi. While internal analysis was based on existing figures and data, the public version of the thesis excludes actual data, whereas presented figures serve only to illustrate the methods and findings.

In general, one should keep in mind that historical data is the most relevant given stable environment. If the environment changes (e.g. team structure- or functional areas change), historical data becomes less reliable. Also, in planning the environment is assumed to be similar as in the past.

Product backlog data analysis was carried out in Excel and calculations were done for individual teams and functional areas separately. Single- and multi-dimensional analyses will be illustrated to provide better visibility of the development structure and processes.

*Single-dimensional analysis*. A data set which consists of all user stories developed by individual teams in different functional areas until the last completed sprint. Cross-table calculations will provide the visibility about the size, number of teams and general allocation of user stories across different teams and functional areas, as shown in Figure 17 below.

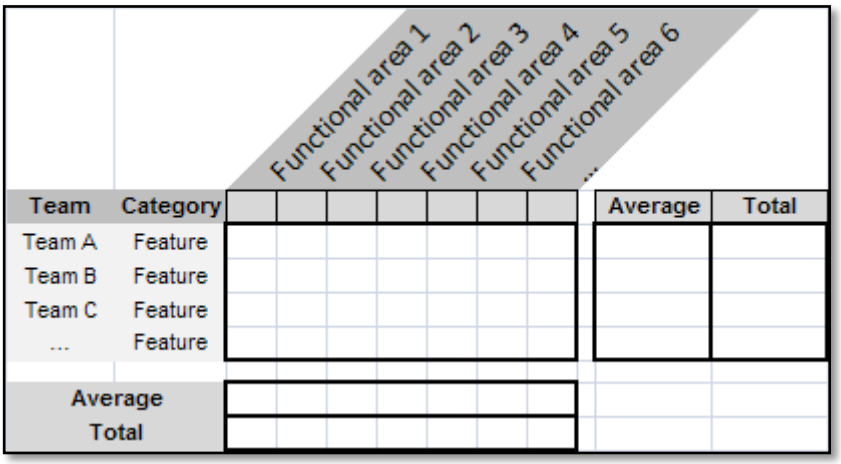


*Figure 17. Single-dimensional analysis of Product Backlog data sample*

*Two-dimensional analysis.* A data set, which consists of the number of, completed user stories completed by individual teams in different functional areas in each sprint separately. These calculations provide visibility of how the development progressed over sprints by different teams. Two-dimensional analysis also supports velocity calculations from the previous sprints. Analyses were carried out for each team separately, and the output of each team was added together to verify the data with the total development progress as in single-dimensional analysis. Cross-table calculations of each team and figure were done as shown in Figure 18.

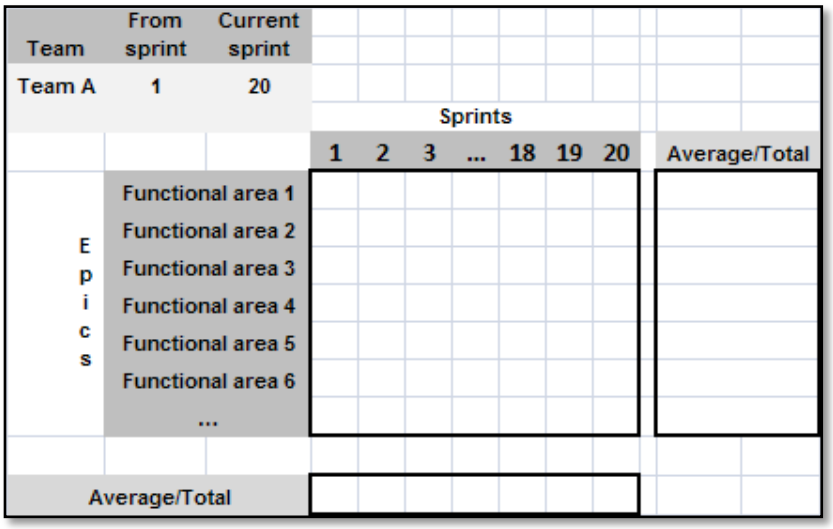


*Figure 18. Multi-dimensional analysis of Product Backlog data sample*

Single- and multidimensional data analysis provided empirical calculations and figures for separate functional areas and individual teams, including completed effort, number of user stories, user story duration, as well as the average velocity in each functional area for individual teams.

#### *3.3.2.2. Questionnaire*

Apart from Product Backlog analysis, a questionnaire research across all teams located in different sites was carried out. The results were collected using a web-based survey tool and the link was distributed by email through Scrum Masters to all teams. Full questionnaire can be found in Appendix 2.

The goal of the questionnaire was to gather direct feedback from teams regarding:

- ✓ General sprint activities, team structure and expertise

- ✓ Effort estimations
- ✓ Velocity

The questionnaire included a set of multiple-choice and opinion-based questions. In multiple-choice questions, respondents had to choose one or more options from the given alternatives. In opinion-based questions, participants had to provide a response based on a 5-point Likert scale ranging from Strongly Agree to Strongly Disagree. Additionally, the questionnaire included open-text answer fields for any additional comments related to each of the areas stated above.

The response period to the questionnaire was one month. Responses were treated anonymously across teams and the amount of responses received (N, x%) was considered to be reliable.

### 3.3.3. Analysis

Analyzing and confirming/rejecting the hypotheses should help identify root causes of problems in current development structure and support suggested improvement actions. In this section, I will discuss findings and results from each method. **Data and figures are modified for confidentiality reasons.**

#### *3.3.3.1. Product Backlog data analysis*

As discussed earlier, different skills and level of expertise are required for various functional areas. Differences across functional areas were identified through calculations of total size in story points, number of user stories and number of teams in each area.

**a). Single-scale analysis of the functional areas**

Functional areas vary in size, i.e. the total amount of effort required in each area differs. As illustrated in Figure 19 below, functional areas differ significantly in the total effort required.

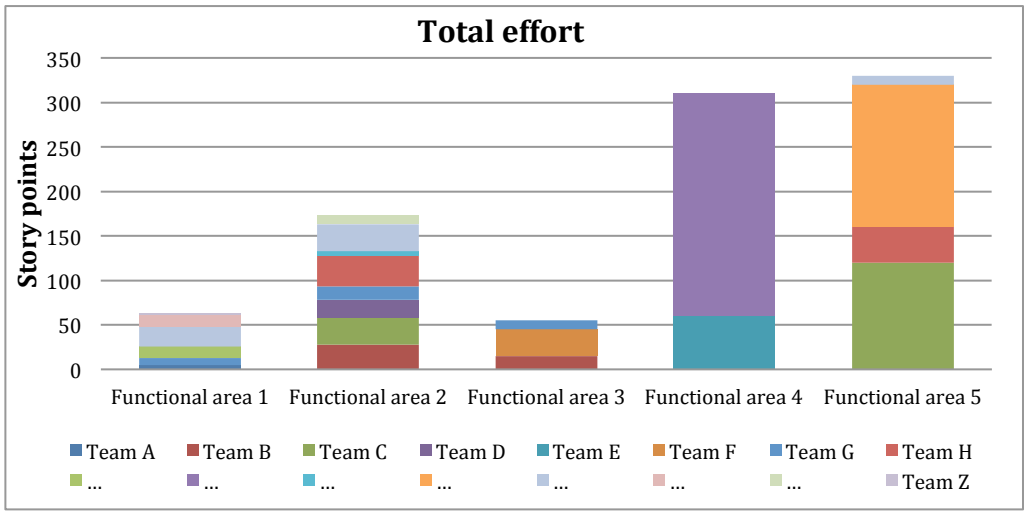


*Figure 19. Total effort across functional areas*

For example, in Functional areas 4 and 5 there were more story points completed than in Functional area 3, which implies that some areas of the program have more requirements and thus have higher total effort. Also, one can notice that currently the total effort in each area comes from the combined contribution of multiple teams. While total effort within each area is distributed across multiple teams, contribution of different teams is not even.

Effort is a measure of relative size of different requirements in story points. Requirements are developed as user stories, thus larger areas should have more user stories. Further, the number of user stories in each area completed by separate teams was calculated from the sample data.

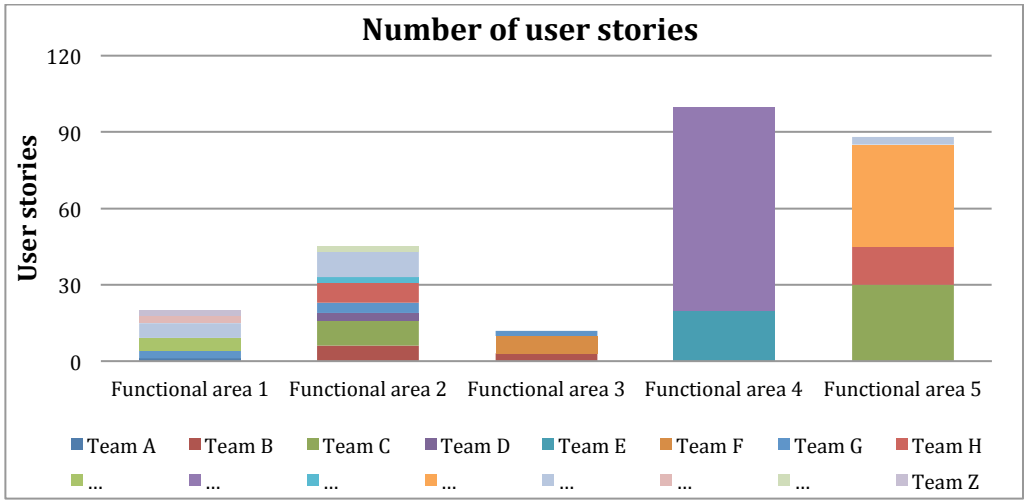


*Figure 20. Number of completed user stories across functional areas*

It was identified that even though number of user stories in some areas were not large, user stories are assigned to several teams (e.g. Functional areas 1, 2 and 3). Also, as can be seen from Figure 20, and the Functional areas 4 and 5, teams completed unequal number of user stories within each functional area. Given that competence comes from experience, knowledge of teams therefore varies across different areas.

Average user story size of each functional area was calculated based on total size and number of user stories.

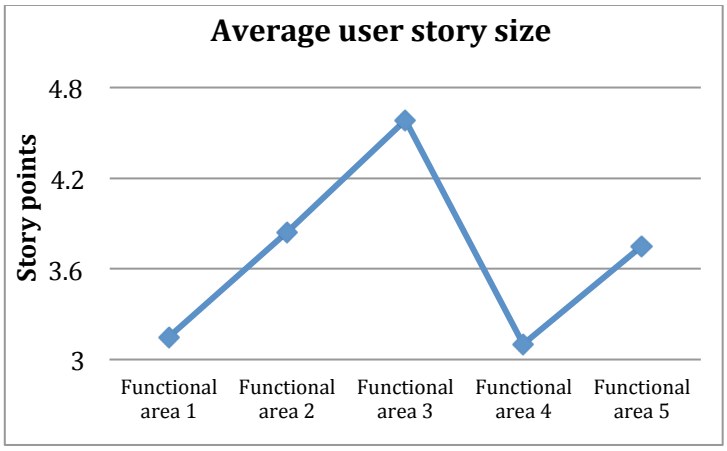


*Figure 21. Average user story size across functional areas*

Additionally, the average user story size in each functional area was calculated for each team separately. The average user story size varies across different functional areas as well as across teams within each area. For example, as in Figure 22 below, while the average user story size of Functional area 1 is 3,15 story points, average user story size across teams within the area varies between 0,5 and 4,7 story points.

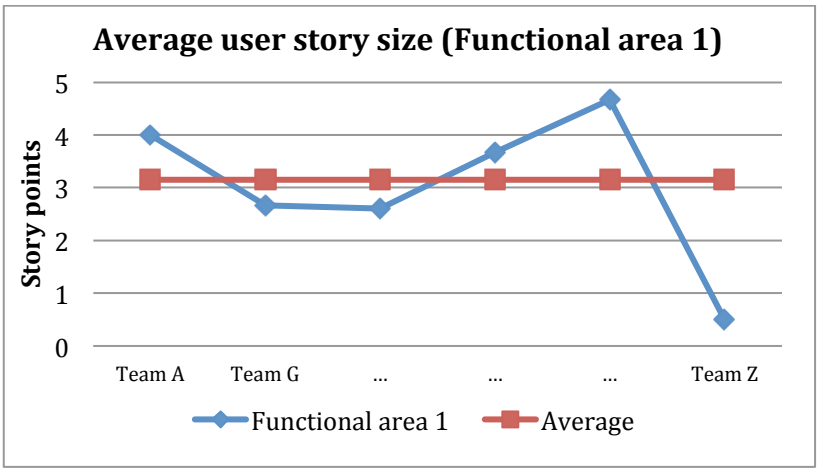


*Figure 22. Average user story size in Functional area 1*

While larger user stories require more time to be developed, the duration will also depend on competence and level of expertise of each team. Further, average duration of completed user stories was calculated.

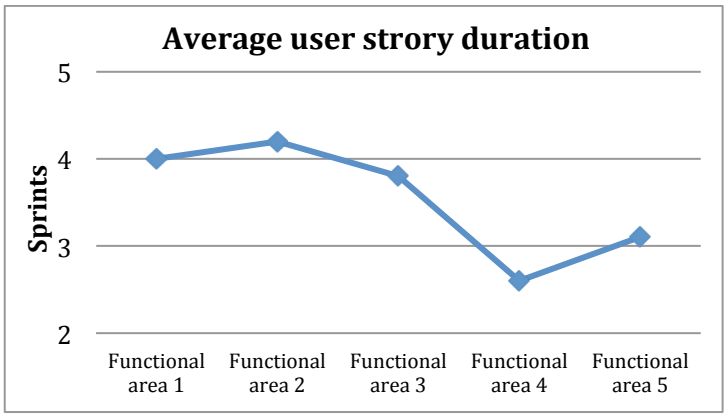


*Figure 23. Average user story duration across functional areas*

As shown in Figure 23, average user story duration varies and is mostly more than one sprint. Also, comparing the average duration and average user story size, it was identified that duration is not proportional with the size. Since duration comes from story complexity and team expertise, the results show differences across different functional areas as well as teams.

User story duration also varies within functional area depending on the team. Figure 24 illustrates these differences using Functional area 1 as an example.

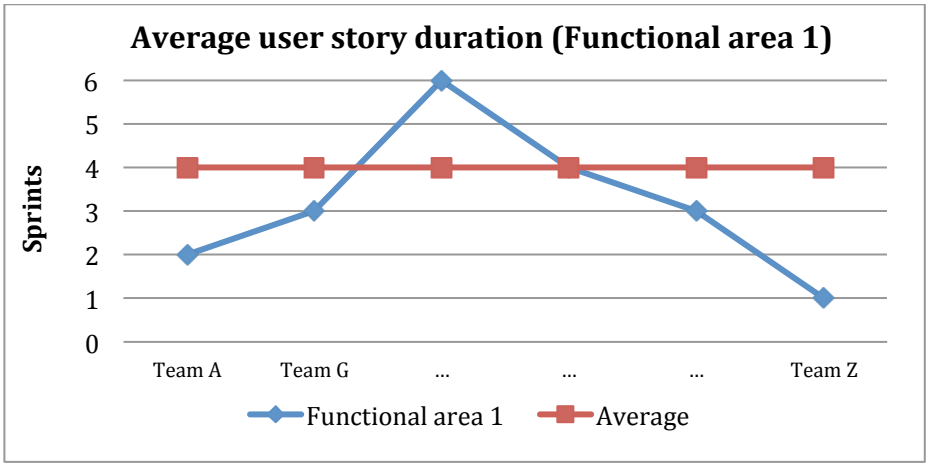


*Figure 24. Average user story duration in Functional area 1*

Additionally, it was observed that story duration was not proportional with the user story size. This means that apart from size, the duration of user stories also depends on the complexity of requirements, as well as expertise and average velocity of the teams.

Differences in use story size and duration were identified. Further analysis showed that multiple user stories were developed in parallel within and across different areas.

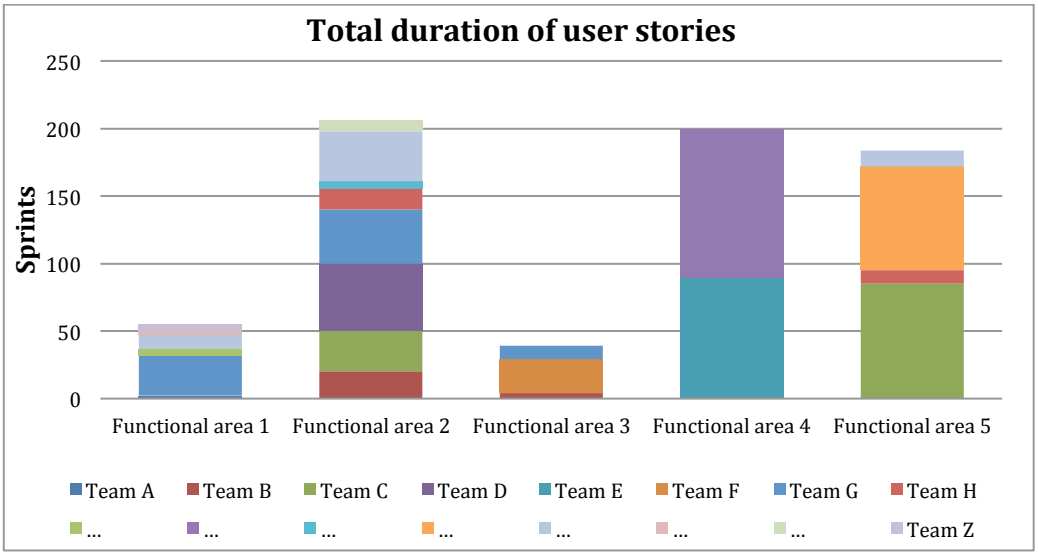


*Figure 25. Duration of user stories across functional areas*

As in Figure 25, the total duration of user stories in varies across different areas. Total effort within each area is distributed across different number of teams. Thus, the total duration depends not only on the number of teams, but also on expertise level of the teams. While Functional areas 4 and 5 have approximately the same size (Figure 19), comparing completed size with total duration, it can be seen that while total size of Functional area 5 is bigger, the duration is shorter. Therefore, Functional area 4 is more complex or expertise level is lower.

**To sum up, single-scale analysis of functional areas shows that total effort, average user story size and duration differ across teams. Thus, each area requires different velocity and number of teams. Also, differences in the velocity and expertise level across teams working in same areas were identified. Thus, the current allocation of user stories decreases efficiency and complicates planning because overall the expertise level is unevenly distributed across different functional areas.**

### (b). Single-scale analysis of the teams

Since teams are assigned user stories across multiple areas, current team structure will be further analyzed to identify complexities in team structure and consequences of such allocation.

Single-dimensional calculations for individual teams and functional areas showed that the overall contribution of each team consists of multiple functional areas.

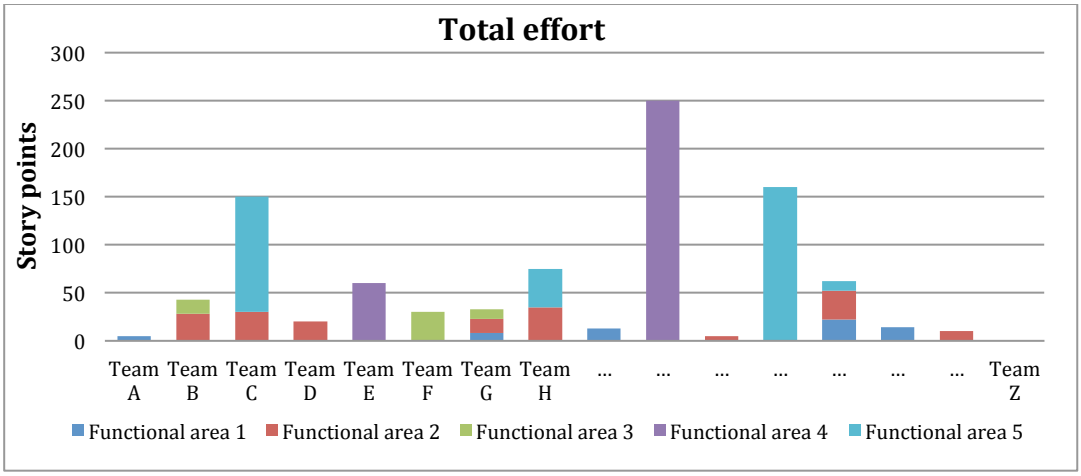


*Figure 26. Total effort across teams*

Total effort completed by the teams is accumulated from multiple functional areas (Figure 27). Also, total size completed by a team across areas varies. Therefore, generally the areas of expertise and competence of teams varies in different areas.

Additionally, since functional areas vary in complexity and user story size, the number of user stories completed by each team need to be calculated.

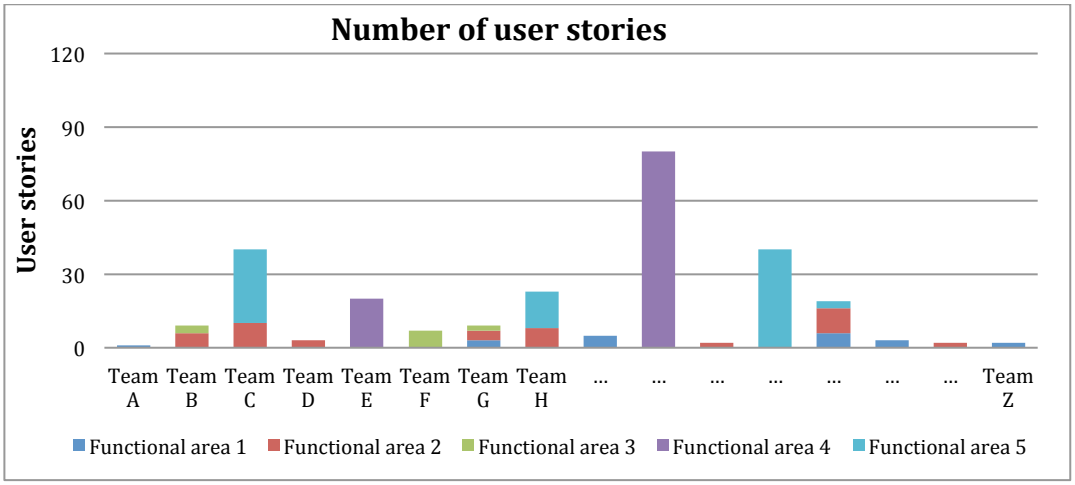


*Figure 27. Total number of user stories completed across teams*

As illustrated in Figure 27, teams have developed user stories mostly within one area and additionally a few user stories across multiple other areas (e.g. Team C). Also, there are certain teams whose core area of expertise is difficult to define because a small amount of user stories was completed in different functional areas (e.g. Team G). Further, none of the teams developed user stories across all functional areas during the studied period. This implies that due to the large scale of the program total effort of individual teams is limited only to a few areas.

From user story distribution it was identified that most teams were gaining wider expertise across different functional areas instead of deeper expertise within a few specific areas. Also, user stories completed across different areas within each team suggests that team expertise across areas is unbalanced. While the following allocation of user stories may make teams more multifunctional, overall team expertise is lower, especially in supportive areas.

Earlier it was identified that the duration of user stories differ. Further, average user story duration in individual teams was calculated. As one can observe from Figure 28 below, the average user story duration varies across different teams.

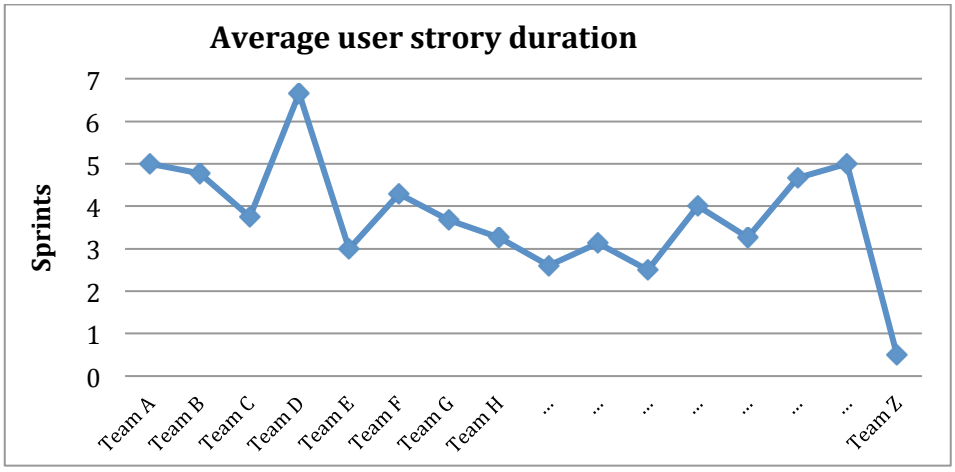


*Figure 28. Average user story duration across teams*

Because the average duration varies, on average some teams complete user stories more often than others independently from velocity. Average user story duration varies due to differences in user story complexity and team expertise across functional areas. The following also implies for a single team across functional areas. Figure 29 below shows varying average user story duration within Team G.

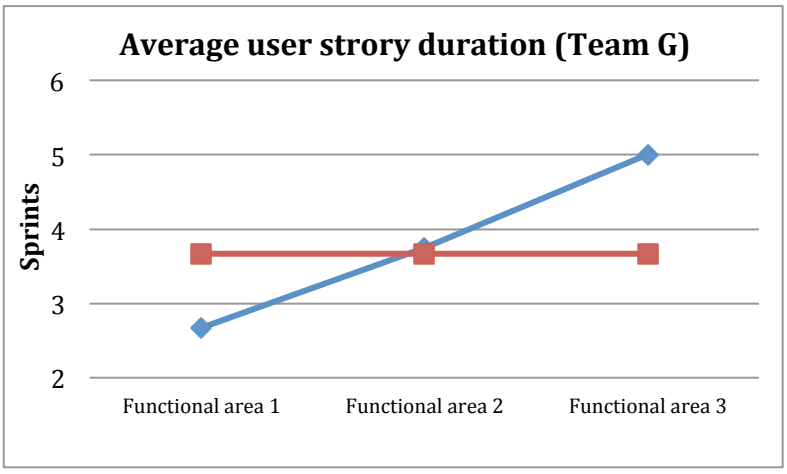


*Figure 29. Average user story duration across functional areas in Team G*

While the average user story duration of Team G is 3,7 sprints, the average duration of user stories across different functional areas varies between 3 and 5 Sprints.

It has been identified earlier that a different number of teams is assigned to each functional area. Since teams develop across multiple functional areas, the total development duration comes from multiple functional areas.

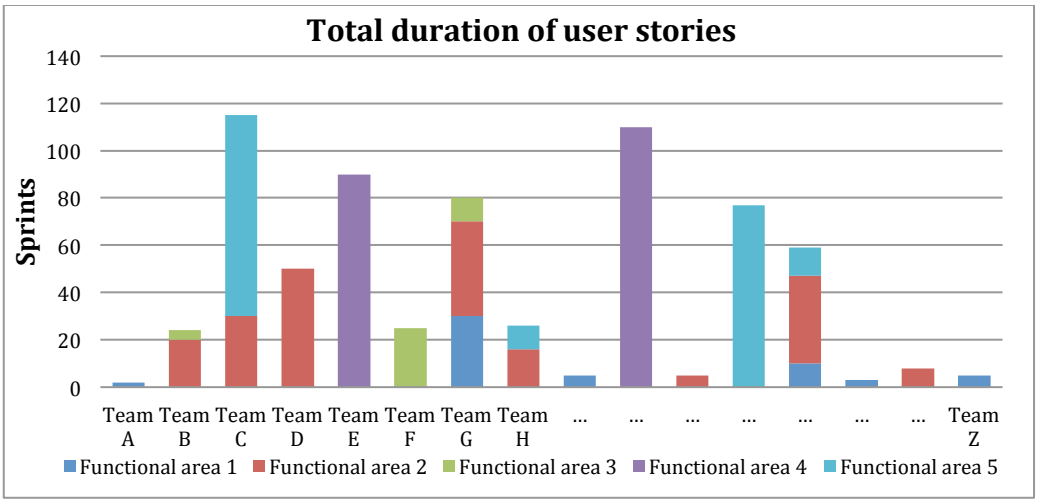


*Figure 30. Total duration of user stories across teams*

Figure 41 shows that the overall duration results from a combination of user stories across different functional areas and none of the teams have similar structure. Additionally, comparing the total duration with the number of actual sprints (see Figure 13 earlier), it can be concluded that teams develop multiple user stories simultaneously.

**To sum up, single-scale analysis of the teams, shows that the total effort completed by each team is combined of multiple functional areas. Due to varying set**

**of functional areas and expertise level across teams, teams cannot be compared. Also, since team expertise varies across functional areas and team competence is lower in supportive areas, the current allocation of user stories decreases efficiency and complicates planning.**

### (c). Two-dimensional analysis of the teams

As proved earlier, requirements vary in complexity and team expertise differs. Also, since teams develop multiple user stories concurrently within and across different functional areas, the team average velocity depends on the number of user stores developed in each functional area.

Further, two-dimensional analyses on a sprint-by-sprint level were carried out for individual teams and functional areas separately. One team is chosen to illustrate the results and findings regarding velocity and overall development process by individual teams.

Multidimensional analysis of Team B in different sprints is used as an example to illustrate and discuss the differences in the story points completed over the period of time (i.e. sprints).

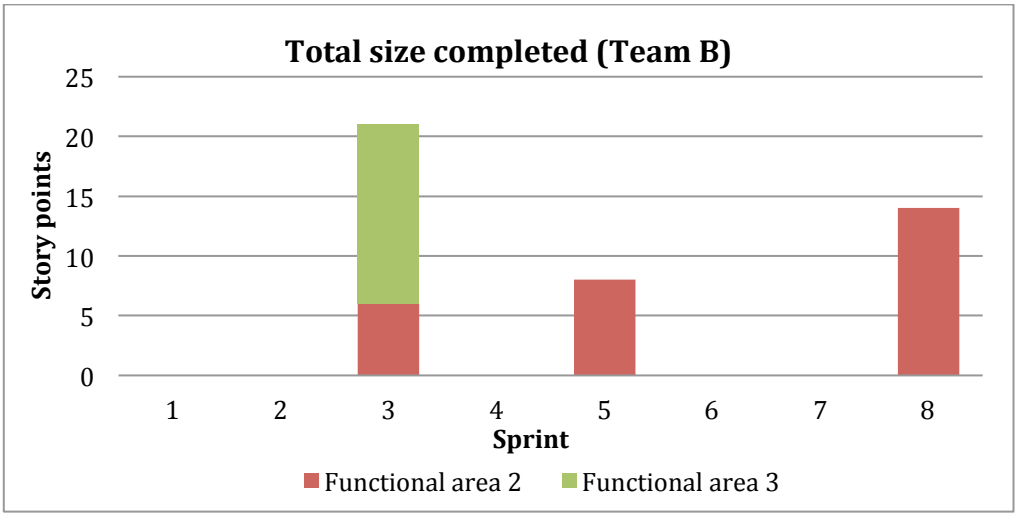


*Figure 31. Total effort completed by team Bravo (Sprints 1-8)*

As illustrated in Figure 31, user stories were completed in sprints 3, 5 and 8. Also, in Sprint 3, the total amount of completed story points appears from two different functional areas.

Next, the amount of completed user stories was calculated across functional areas. The results are illustrated in Figure 32 below.

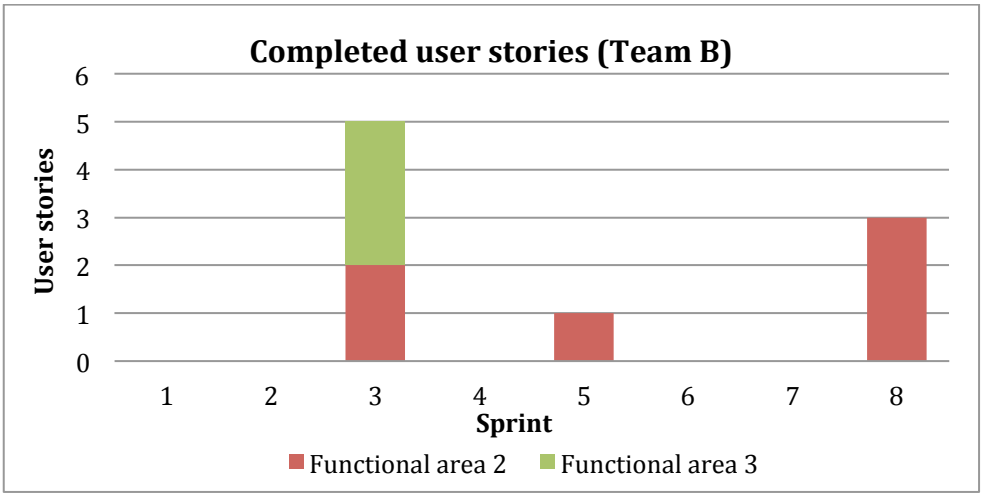


*Figure 32. User stories completed by Team B*

Team B actually completed multiple user stories in Functional areas 2 and 3. Multiple user stories were completed during sprints 3 and 8. Additionally, given that user stories vary in duration, the actual completed effort was distributed over different number of sprints. Thus, the existing methods do not provide accurate information about velocity. Instead, duration of user stories needs to be taken into account. Knowing sprint start and sprint end, the number of ongoing user stories in each sprint was calculated. The results of Team B are shown in Figure 33 below.

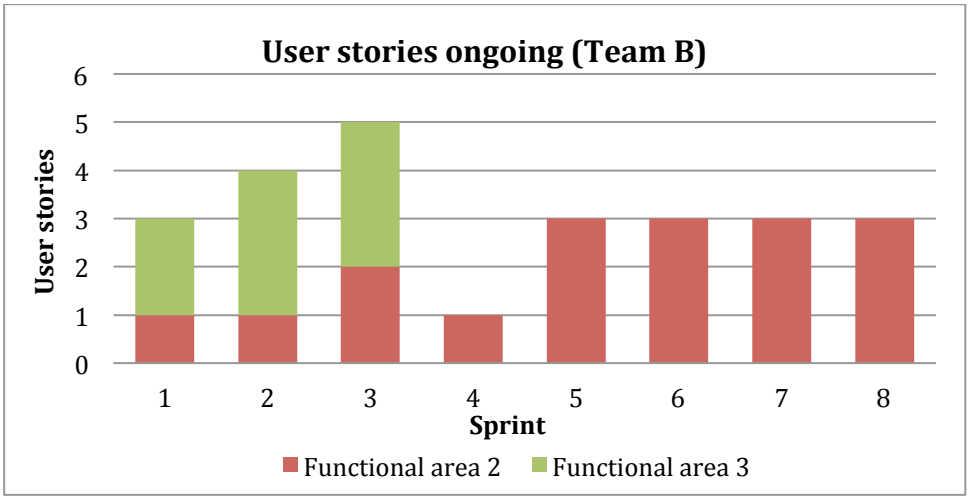


*Figure 33. Ongoing user stories each sprint by Team B (Sprints 1-8)*

We can see that for Functional area 3 the team completed user stories in Sprint 3. However, one of the user stories actually lasted 3 sprints and another has been developed

for 2 sprints. Additionally, one can observe that the team developed multiple user stories concurrently across different areas.

Given the findings, in order to calculate velocity, i.e. story points completed in one sprint, the duration of individual user stories should be included in the calculations. Also, because functional areas differ, the velocity should also be calculated separately for each functional area.

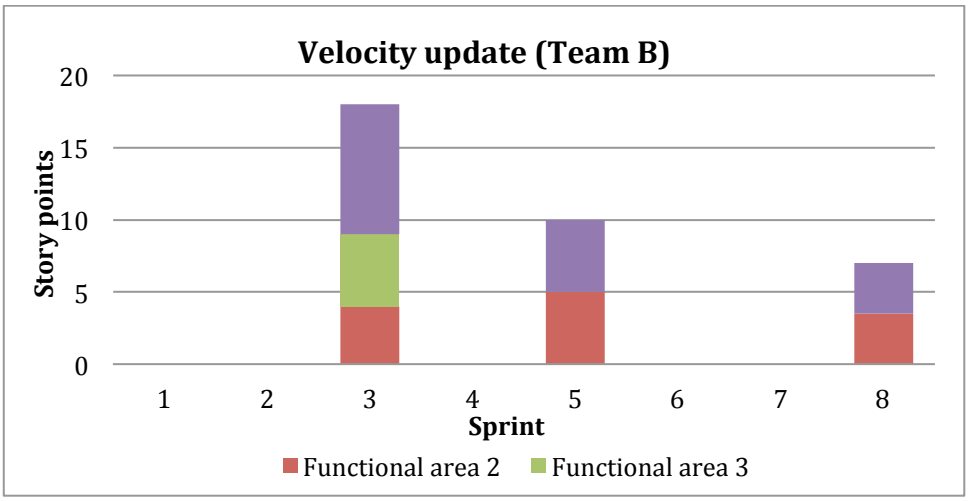


*Figure 34. Velocity update of Team B*

As it can be seen, velocity is calculated only in sprints when a user story is completed. Each completed user story is divided by the amount of sprints it lasted and in case of multiple user stories their velocities are added.

It was further noticed that team velocity in each functional area is affected by parallel activities across different areas. For example, in Sprint 3 the team completed user stories in Functional areas 2 and 3, implying that actually the team members had been working on different functional areas in parallel. Further, since only a part of the team was working on Functional area 3, average the velocity in Functional area 2 is lower. Table 4 below shows the average velocity of team B.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Average |
|---|---|---|---|---|---|---|---|---|---|
| Functional area 2 | | | 4 | | 5 | | | 3,5 | 4,2 |
| Functional area 3 | | | 5 | | | | | | 5 |
| Sprint velocity | | | 9 | | 5 | | | 3,5 | 5,8 |

*Table 4. Sprint and average velocity of Team B*

Velocities were calculated in sprints when the user stories were completed. The average velocity is calculated from sprint velocities. However, the overall team velocity is affected by parallel activities, thus the average velocity in each functional area cannot be considered separately and the sprint total velocity may be different if a different set of user stories is assigned. It was identified that some teams perform more multitasking than others, which makes it even more difficult to estimate the team average velocity in each functional area.

**To sum up, two-dimensional analysis of individual teams showed that the velocity of the teams is different in different functional areas. Also, because user story size and duration vary, completed effort is not the same as velocity. Further, since teams develop across multiple functional areas concurrently, a team's average velocity differs from velocities in each area, which complicates velocity-based planning.**

### (d). Two-dimensional analysis of feature development processes

Further, calculations for individual teams helped in identifying the existing challenges in planning and tracking development given existing allocation of user stories.

Since teams developed user stories across multiple functional areas in different sprints, completed story points resulted from varying number of teams, functional areas and amount of sprints. Because some user stories last longer, velocity will depend on the size of each completed user story and the amount of sprints it has lasted.

Also, it was identified that some functional areas are completed in high quantities over few sprints, while in other areas the content was delivered over longer period and in smaller portions. This happens due to varying complexity and overall expertise of the teams. Thus, duration will be longer for larger and more complex user stories. Consequently, total velocity in each area will depend on the amount of teams developing user stories in parallel and velocities of those teams. Additionally, given that functional areas vary in size and complexity, completed amount of story points in a sprint is not proportional with how much was developed in that sprint.

One or more teams develop the content in different areas. It was identified that velocity fluctuated because the number of teams and user story duration varied over

sprints. Table 5 below shows the figures for the total realized and average sprint velocities in each functional area, which results from the output of all teams.

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Average realized |
|---|---|---|---|---|---|---|---|---|---|
| **Functional area 1** | | | 6 | 20 | | | | | **13** |
| **Functional area 2** | 2 | 5,0 | 3,5 | | | | 15,7 | 4,0 | **6** |
| **Functional area 3** | | | | | | | | 1,7 | **1,7** |
| **Functional area 4** | | 2,0 | | | | | | 3,5 | **2,8** |
| **Functional area 5** | | 2,5 | 8,0 | | 8,0 | 17,0 | 5,2 | 15,5 | **9,4** |
| **Sprint velocity** | **2** | **9,5** | **17,5** | **20** | **8** | **17** | **20,9** | **24,7** | **16,8** |

*Table 5. Sprint and average velocity of feature development*

The sum of velocities in each sprint provides the total realized sprint velocity across all feature development teams and functional areas. The average realized velocity in each functional area is calculated from sprint velocities respectively. One can see that each area has a different velocity. Further, the total velocity is a result of different teams with varying expertise, performance and combination of tasks across multiple areas. Therefore, the functional areas and teams need to be tracked individually. Also, team velocities are incomparable due to varying combination of expertise areas.

**To sum up, the two-dimensional analysis of feature development processes shows that larger functional areas need higher velocity and smaller areas require lower velocity. Further, total velocity is based on the sum of velocities of individual teams within each area. Currently, multiple teams produce different quantities of output across multiples functional areas simultaneously. Therefore, the average velocity in each area will depend on the number of teams assigned, team velocity, as well as capacity resulting from the allocation of user stories in other functional areas. Since teams develop user stories simultaneously and in different order, velocity of each area fluctuates depending on other parallel activities of the teams.**

### (e). Overall versus release-specific analysis

Finally, release specific content was analyzed to identify the main aspects, which need to be addressed in order to facilitate higher accuracy and more structured approach to release planning and development. The analyses were carried out for one selected major release developed over the same period of time (i.e. same sprints). Comparing the release

specific figures with the overall development helped identifying the differences in the release content and team load in the identical sprints.

Firstly, even though release was developed during the same sprints, in some functional areas the total developed size differed from the release specific content, as illustrated in Figure 35 below.

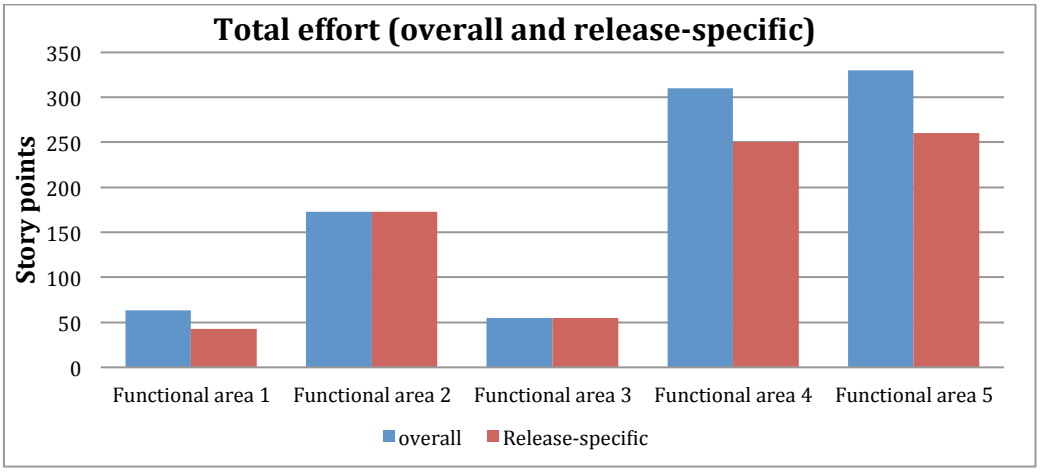


*Figure 35. Total effort completed overall and in individual release*

Observing that the release specific content in some functional areas is smaller than the overall size developed implies that part of requirements are internal or belong to another release. Further, release specific content could be developed either by fewer teams or in parallel with other requirements.

Carrying out the analysis, it was identified that while in some areas fewer teams were assigned, the same teams mostly developed release-specific requirements in parallel with the overall content.

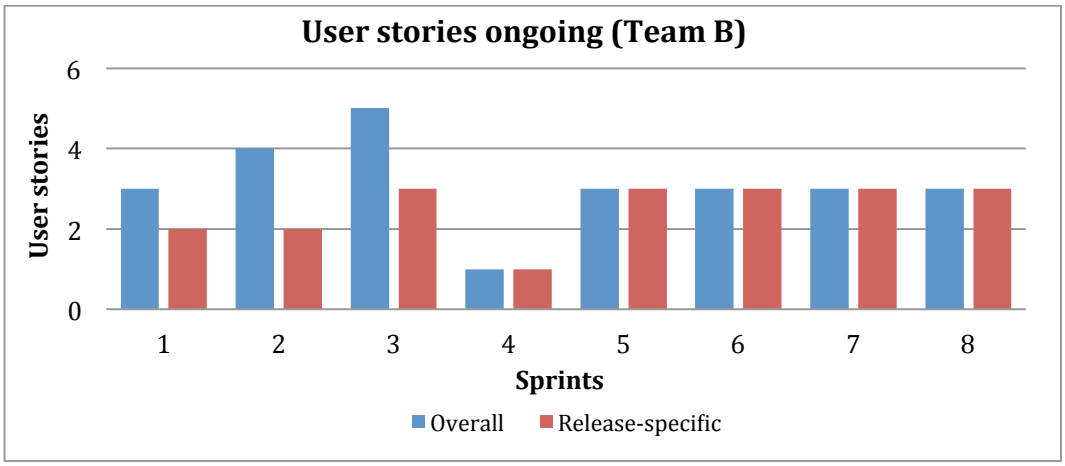


*Figure 36. User stories ongoing in Team B (Overall and Release-specific)*

Teams are assigned new requirements across areas depending on their level of expertise. Also, some requirements belong to a different release or are internal requirements. As shown in Figure 36, it was identified that during some sprints teams partially developed requirements which did not belong to the main ongoing release.

As a result, the total velocity of a release will differ from the actual velocity of all feature development teams. In Scrum, planning and tracking of development is based on velocity given full capacity. However, the velocity of the teams and consequently functional areas will be affected by multitasking. Generally, possible parallel activities should be avoided or considered in release planning. For example, if multiple releases need to be developed concurrently, assigning different teams would support more accurate planning and monitoring of each release. Alternatively, teams could work on user stories of one release at a time in the order of preference.

**To sum up, the findings imply that in addition to the number of teams, size of functional areas and user story complexity, release-specific velocity will depend on the amount of "outside release" activities carried out in parallel during the development sprints. Since these external activities affect the critical path of a release, they need to be addressed in the velocity calculations. In other words, to optimize planning and minimize the critical path, release specific content should be developed in an organized manner and with maximum capacity, or release-specific velocity needs to assume team load on tasks outside the main release.**

#### *3.3.3.2. Questionnaire results*

With Product Backlog analysis several issues complicating planning were identified. The questionnaire further validated that teams have been multifunctioning and expertise within teams varied. Also, the questionnaire results showed the challenges related to effort estimations across the development teams.

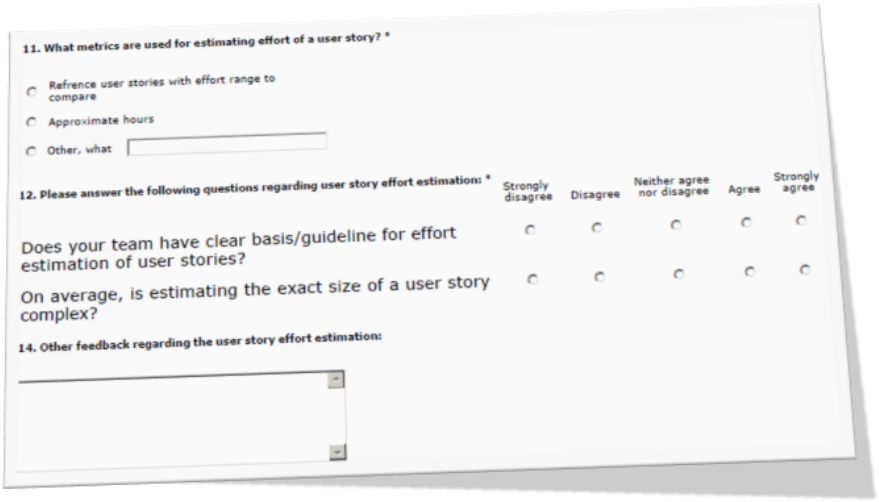


*Figure 37. Cross-team questionnaire*

According to the majority of the respondents, all team members could not develop every user story assigned to the team (22/31 strongly disagree or disagree; 8/31 agree or strongly agree that all team members can perform all tasks). Due to varying expertise and multitasking within teams, team velocity would depend on the combination of the assigned tasks (Figure 38).

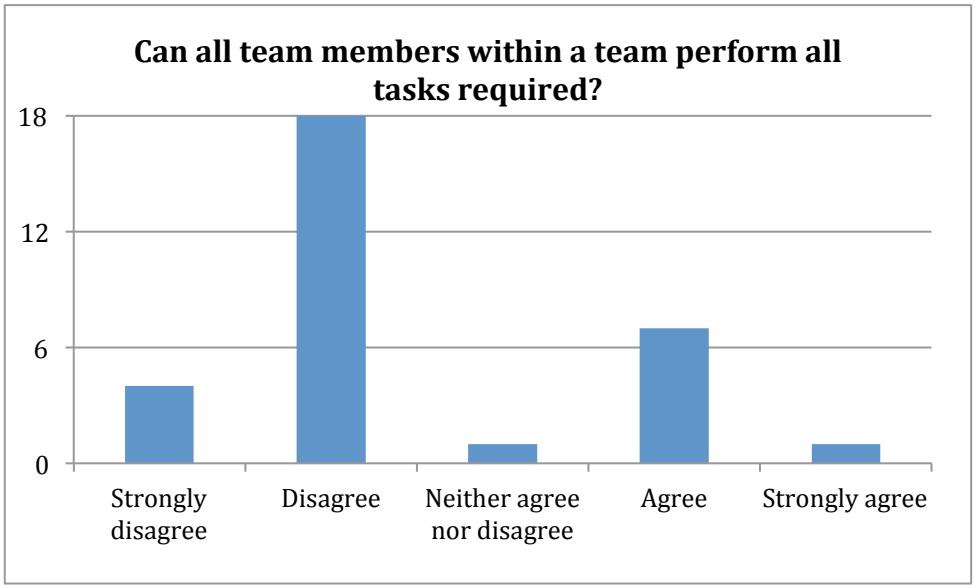


*Figure 38. Most teams are not multifunctional within*

All respondents reported that a team performed multiple tasks simultaneously either always (18/31) or sometimes (13/31), meaning that within a single team parallel activities

were carried out by different individuals. Consequently, the output of a single team resulted in development of different requirements in parallel (Figure 39).

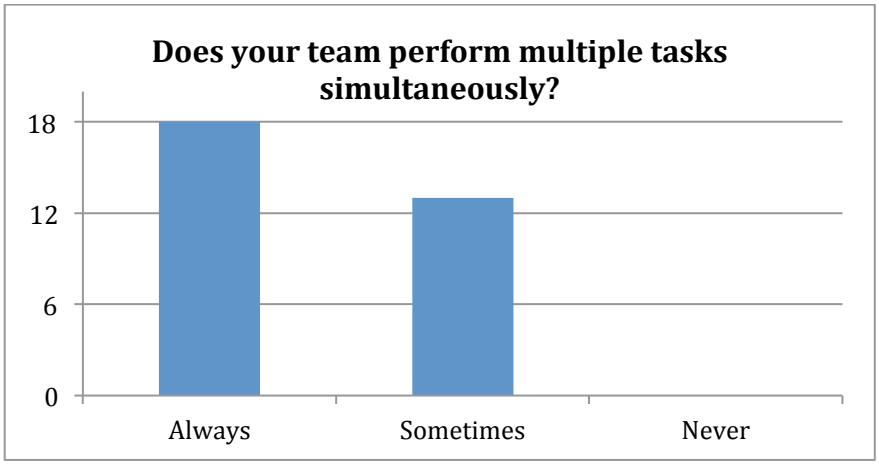


*Figure 39. Teams do multitasking*

Apart from discussions with the Flexi management and developers, the aim of the questionnaire was to provide solid information on current issues related to effort estimations. As expected, the questionnaire showed lack of common metrics and references, as well as lack of general understanding about the importance and the use of effort estimations.

As shown in Figure 40 below, from the responses it appears that teams use different metrics for estimating the user story size. In Scrum, effort estimations are done in story points to measure the relative size of different requirements. From the respondents, only half (16/31) have been using reference user stores with an effort range, 8 respondents estimate in approximate hours and 6 respondents use other metrics, such as guessing based on *"experience and gut feeling"*, *"referring to previous user stories developed by the team"* and *"initial effort estimations of expert team"*.

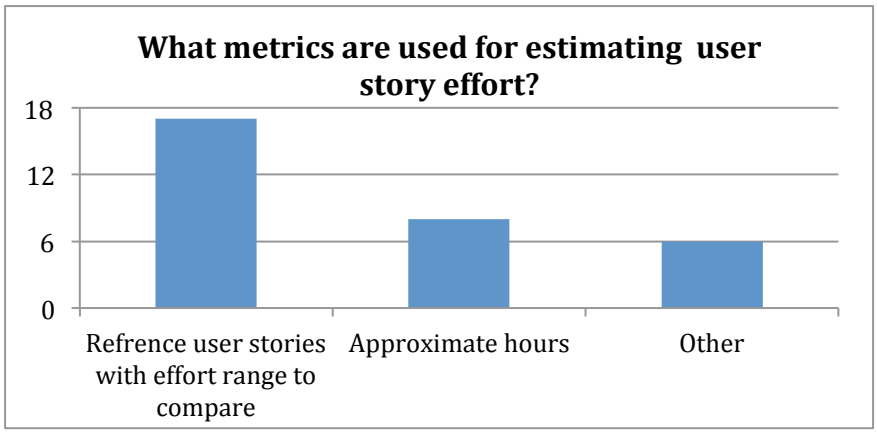


*Figure 40. Different metrics are used in effort estimations*

Further, the questionnaire showed lack of clear basis and guidelines for effort estimations. Based on discussions with key stakeholders, the guideline had been not to exceed initial estimates. Further, according to the questionnaire, 18 out of 31 respondents did not agree on clear basis of effort estimations of user stories (Figure 41). So, inaccurate and incomparable effort estimations result from unclear basis of initial effort estimations and insufficient guidelines on how effort estimations should be done

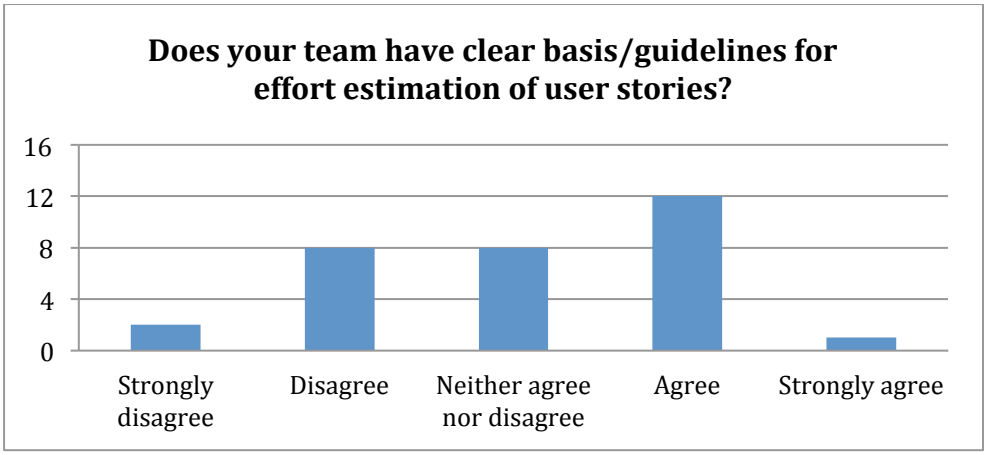


*Figure 41. Common basis of effort estimations are missing*

Additionally, open-text answers provided additional information to support the ambiguousness in effort estimations (Table 6).

| |
|---|
| *"Team doesn't see effort estimation so important, it doesn't describe anyway how long it can take to complete user story."* |
| *"Effort estimation or story point are not important to our team so we just make guess at the beginning of sprint."* |
| *"Effort estimation started with relative size (Story Points) and is now converted to time needed multiplied by average SP per Sprint."* |
| *"Received effort estimation inside feasibility studies is not clear. References are different in every team/location."* |
| *"Some reasons for failed estimates: unclear requirements/grey areas, poor team competence on the functional area under development."* |
| *"Previous experience not always works since the area of the user story might change or be new or it is about an unknown area."* |

*Table 6. Open-text answers on effort estimation (Questionnaire results)*

Open-text answers identified lack of understanding about the implications and importance of effort estimations. It appeared that initial effort estimations inside feasibility studies were not clear and references varied in every team and location. Also, some teams made estimations in hours and converted the size into story points. This method contradicts with theory since effort estimations should tell relative size of requirements, i.e. estimations of effort or size compared to other user stories in the same functional area.

As discussed, the accuracy of effort estimations is related to past experience in similar user stories. Therefore, unbalanced experience in different functional areas causes additional inaccuracies in effort estimations and velocity. User story effort estimations will vary across teams depending on velocity and expertise level of the assigned team.

While velocity is the main metric to plan and track release progress, Scrum teams are self managing and responsible for planning sprint activities. The questionnaire also revealed issues related to team velocity.

In the questionnaire, only 9 out of 31 respondents reported to know the velocity of their team. During sprint planning, teams break down user stories into tasks and effort estimations on task level are carried out. Further, teams plan activities based on estimated

effort of tasks. Without planned team velocity, teams cannot track whether the development progressed according to plans.

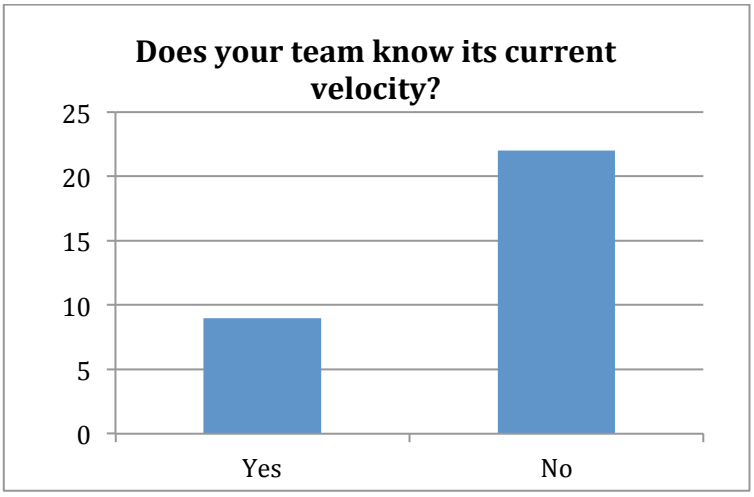


*Figure 42. Most teams don't know their velocity*

Questionnaire results revealed that currently there are many distracting activities that slow down velocity (21 respondents either agree or strongly agree; 3 disagree).

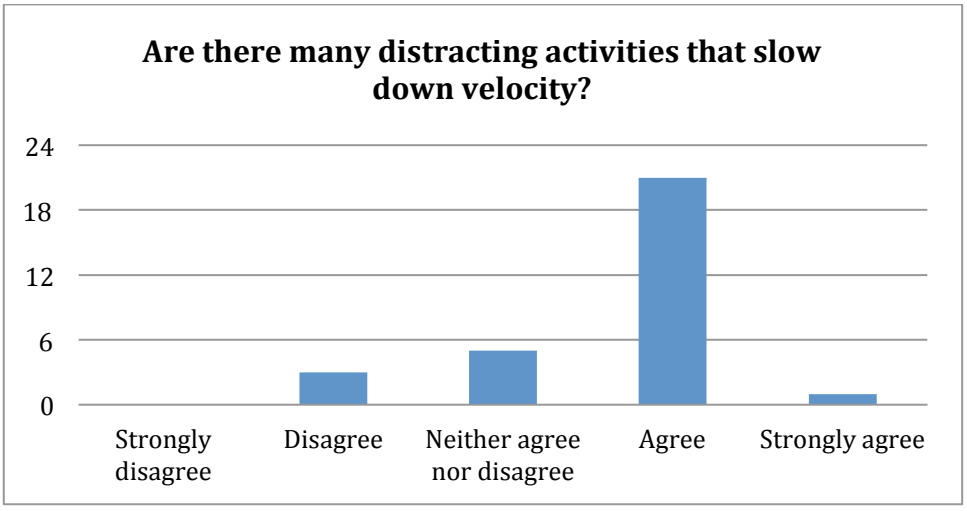


*Figure 43. Velocity is affected by parallel activities*

Based on open-text answers, in respondents' opinion the main factors that slow down velocity are newly raised requirements, maintenance tasks or other parallel activities as well as hardware unavailability. The following factors indeed affect how many story points are developed in a sprint, however the amount of these activities is more or less stable over sprints.

Additionally, respondents provided valuable information regarding velocity in open-text questions. Below are some of the responses:

| |
|---|
| *"We used to calculate velocity but after a while team decided to stop tracking it. Team took in many sprints too many story points into sprint content and it didn't bother much the team. User stories was taken into sprint content more like what was seen sensible to do at the moment regardless how big they were (and they still do like that). It was seen insane to spend huge amount of time to split user stories artificially into small enough to fit into one sprint (2 week) as we have so many other issues which are effecting do we get things done in 2 week or not."* |
| *"In order to keep/improve velocity, some sort of steady path (environment) is necessary. Besides, how do you measure velocity? For what a group might be 13 story points and includes all possible testing, for other teams might be 25 and do not care about testing it properly or checking if it breaks other parts of the code."* |
| *"Velocity is meaningful on long term average level and for stable teams."* |
| *"At a general level team knows whether a story is doable in a sprint or not."* |

*Table 7. Open-text answers on velocity (Questionnaire results)*

Open-text questions revealed some reasons why teams did not know or track their velocity. In order for velocity to be meaningful, a steady path is needed, which in some teams' opinion was not the case. Apart from distracting activities, instability in team velocity was caused down by development across different functional areas.

Another issue resulted from the complexity of requirements. While many user stories were too large to fit into one sprint, teams considered artificial splitting of user stories as a waste of time. As a result, teams took too many story points into a sprint content and lost track of sprints velocity.

**To sum up, since velocity is based on effort estimations, absence of common methods and metrics in effort estimations resulted in a lack of understanding regarding the velocity. Also, developing requirements across many functional areas caused instability in the development environment, causing further problems with accuracy of effort estimations and velocity. Resulting from current problems, most teams could not track or improve velocity.**

## 3.4. Root causes of problems based on "5 Whys"

Results from the data analysis and questionnaire support inaccuracies and challenges in current release planning. Having separated and recognized existing issues, root-causes of problems need to be identified. Further, "5-Whys" analysis technique will be carried out. "5-Whys" analysis method is a Six Sigma tool, which helps analyze the symptoms of problems by asking question "why" to the main identified problem and each of the succeeding issues until true root cause of a problem is understood.

### 3.4.1. Teams do multitasking

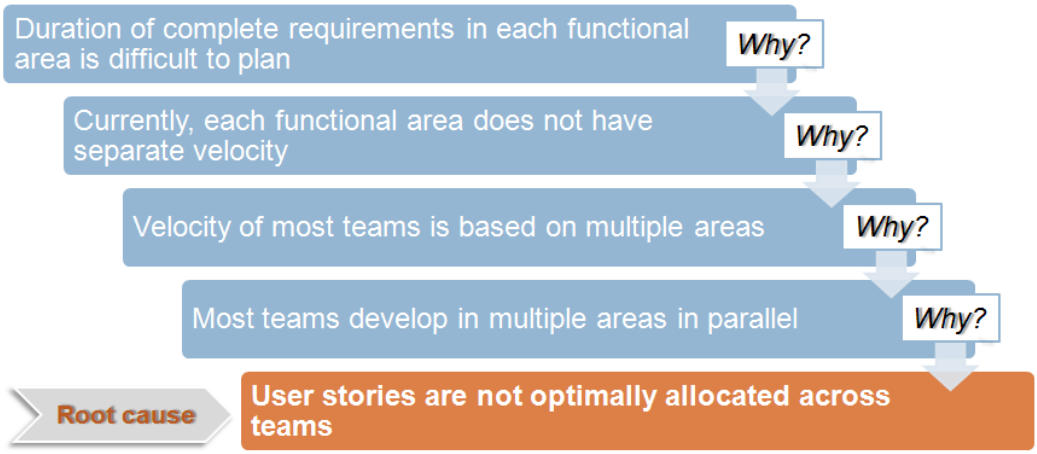


*Figure 44. Root cause of the problem with user story allocation*

Analyses identified that total effort of functional areas differs and the average velocity needed to complete the content of each area varies. The effort assigned to each team is limited compared to the overall content and expertise of teams varies across different areas. Since teams develop in multiple functional areas, planning based on velocity within individual functionality is more complex. Given that teams have higher expertise and thus higher in the core competence areas, overall velocity in each functional area is decreased by supportive teams.

Two-dimensional analysis showed that teams developed multiple user stories across different areas concurrently. This implies that individual team members were responsible for main and supportive functional areas. Therefore, carrying out development by individual members of teams in parallel areas splits expertise within those teams. Also, the velocity of such team in the core area decreases and is formed from combined multiple

velocities across different areas. Since the velocity within each area will depend on size and complexity of multiple user stories across different areas, planning is more complicated.

Additionally, current allocation of user stories makes development less efficient, because teams highly capable to develop faster in their core area of competence are assigned to supportive functional areas, whereas the requirements of the core area are also assigned to less experienced teams. As a result, the overall average velocity decreases and puts a team responsible for multiple functional areas. Further, since team expertise varies, more complex user stories may further be re-assigned during development to a more competent team. As a result, load on teams who are in the critical path may be increased even more and extend the duration.

Also, analysis proved that team velocity within a release differs from the total team velocity due to multitasking. Therefore, even if team overall velocity was higher, the content has been burning down slower. Managing multiple releases within each team complicates planning individual releases because velocity of each release will depend on parallel activities outside the release.

**Therefore, the root cause of the problem is the allocation of user stories. Teams are assigned to multiple functional areas and development takes place across multiple releases in parallel, which complicates planning and decreases overall efficiency.**

### 3.4.2. Effort estimations are not managed sufficiently

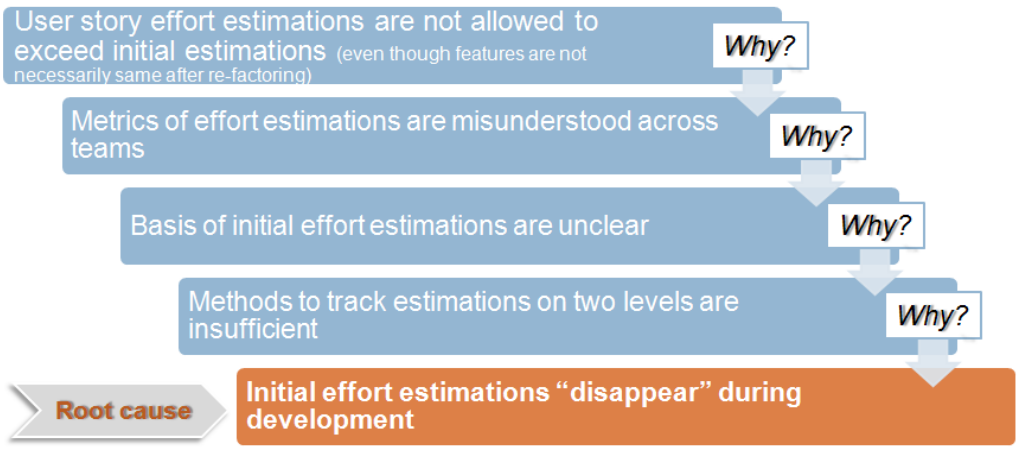


*Figure 45. Root cause of the problem with inaccurate effort estimations*

While initial planning is based on effort estimations of features, which are done prior to main development process, during development feature requirements (breakable content) and development plans (velocity and content re-assigned between teams) may change. Since such changes affect development progress and velocity of individual teams, re-planning should be carried out as new and more accurate information is available after every sprint.

Lack of common understanding about effort estimations resulted in teams giving different estimations about the same content. While in Flexi the effort estimations are done on two levels, problems in managing the accuracy of effort estimations across levels resulted in effort estimations of development teams being fixed to initial estimations. From the analysis, it was identified that metrics were misunderstood across teams.

While feasibility studies of features contained a list of requirements and effort estimations of features, existing references to user stories did not provide clear basis for initial effort estimations of requirements to establish common understanding across teams. Therefore, if requirements changed, it was challenging to track back changes in effort estimations.

Further, it is more complex to estimate the size of new and complex features. Such requirements become more clear and accurate during development as user stories are studied in more detail. Therefore, content of some features may change during development, which will facilitate a need to correct initial estimations during the main development period. However, since these changes would take place on user story level, whereas initial estimations and requirements would be defined on feature level, existing methods to link two levels were insufficient.

**Thus, the root cause of inaccurate effort estimations is that initial effort estimations actually disappear after re-factoring, which makes it impossible to track possible changes in content after re-factoring.**

### 3.4.3. Velocity calculations are incorrect

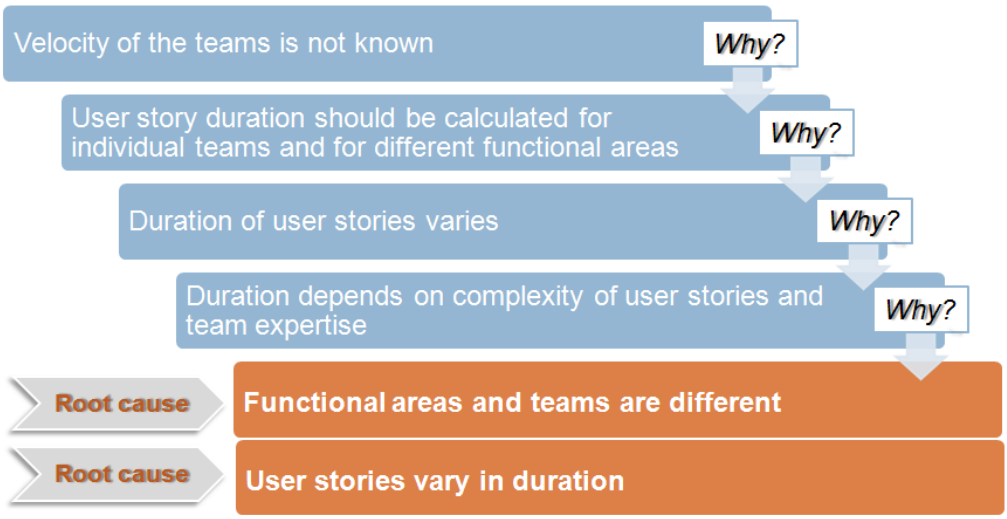


*Figure 46. Root cause of the problem with inaccurate velocity*

Velocity is the main metric used in release planning and it is measured in the amount of story points that a team develops during each sprint. However, analysis showed that velocity of the teams is not known neither by management nor by the teams themselves. Since functional areas vary, velocities cannot be compared across functional areas. Also, because teams vary in expertise and develop across multiple areas, velocities of the teams within same functional area cannot be compared either.

Further, duration of user stories varies depending on user story complexity and currently most user stories currently don't fit into one sprint. Further, teams develop multiple user stories simultaneously, and while the multiple stories may be completed on the same sprint, the duration of each story was actually different. In order to estimate average sprint velocity, duration of each completed user story needs to be included in calculations.

**Thus, the two main root causes of incorrect velocity are missing velocity calculations for individual teams and functional areas, as well as neglected user story duration in the calculations.**

### 3.4.4. Management tools lack visibility of development progress

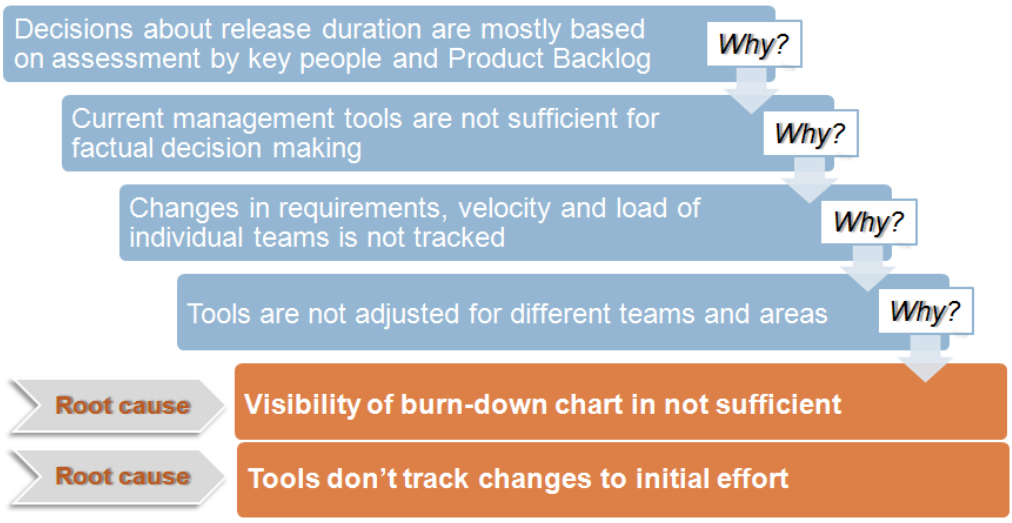


*Figure 47. Root cause of the problem with lack of visibility over the development*

Given the size and complexity of the Flexi program, release planning requires more advanced tools to support decisions by more accurate metrics and better visibility of development processes.

Since teams are multifunctional, velocity will depend on complexity of features within different functional areas and expertise of individual teams. Therefore, while functional areas and teams vary, existing tools did not individualize the differences between different areas and metrics. Failure to distinguish the differences also affected velocity calculations and allocation of user stories. Consequently, key stakeholders made decisions regarding release progress based on professional judgment and assessment of Product Backlog. While changes in requirements and re-allocation of users stories across teams should be normal activities in Scrum software projects, current tools lacked visibility to support adaptation to these changes during release development.

**Hence, root causes for insufficient management tools are insufficient tracking of initial effort estimations, as well as poor visibility on development progress across individual teams and functional areas.**

## 3.5. Improvement suggestions

In this section, improvement actions will be proposed to optimize release planning in Flexi feature development. Facing the challenge of the program design, Epics should be

utilized for mapping teams with their skills set and experience in core areas for more accurate and organized release planning and development.

Effort estimations across teams can be improved with references between initial and development estimations for individual user stories. This will provide clear guidelines about the estimated size through references of requirements.

Velocity should to be calculated for individual teams, because teams have different areas of expertise and load. Additionally, user story duration needs to be included to calculate velocity in a single sprint more accurately.

As an outcome, organizing development processes based on functional areas would facilitate more accurate scheduling and provide better visibility to track progress. Additionally, this would provide more efficient methods to re-plan development when requirements change in order to anticipate possible bottlenecks. Next, I will discuss the proposed improvement actions in more detail.

### 3.5.1. Allocation of user stories

To optimize and improve release planning, velocity needs to be stabilized. Stable velocity implies that development takes place within one functional area at a time. Then it will be possible to track and predict team velocity within each area more accurately. If functional areas are too small, teams can develop across multiple areas, however it is vital that user stories are developed in one area at a time. This way, a team would have multiple separate velocities, which will not depend on overlapping activities across different areas. An example of how user stories should be allocated and developed by some teams is illustrated in Figure 48 below.

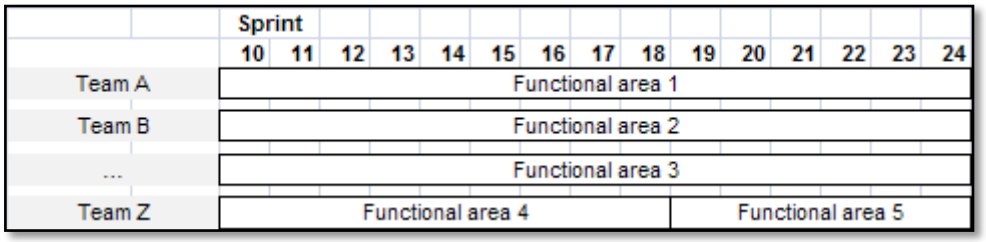


*Figure 48. Proposed allocation of user stories*

Since expertise level affects velocity, team should develop in team's core (strongest) functional area to maximize velocity and effort input in development. Work on the core

area with maximum velocity should provide faster development time for that area. Size of user stories (in story points) needs to be equally allocated across teams who share same areas of expertise. Balanced allocation of user stories across teams will provide stable and targeted growth of expertise in teams and functional areas.

The resources and expertise across different functional areas should be balanced. Further, the right amount of teams should be assigned to each functional area depending on team velocities, total size and overall complexity of requirements. In this way, burn-down of user stories in each area will be based on multiple teams.

| Functional area 1 | Sprint | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | Velocity | Completed |
| Team A | 3 | 4 | 5 | 5 | 4 | 5 | 5 | 5 | 5 | 5 | 4 | 4 | 3 | 3 | | 4,3 | 60 |
| Team D | 5 | 5 | 8 | 8 | 8 | 6 | 5 | 6 | 7 | 8 | 6 | 6 | 6 | 8 | | 6,6 | 92 |
| Team H | 6 | 4 | 4 | 5 | 5 | 4 | 4 | 4 | 5 | 5 | 4 | 4 | 4 | 5 | | 4,5 | 63 |
| Team P | 2 | 3 | 3 | 3 | 2 | 2 | 2 | 3 | 2 | 3 | 2 | 3 | 3 | 2 | | 2,5 | 35 |
| | | | | | | | | | | | | | | | | 17 | 236 |

*Figure 49. Proposed allocation of user stories across multiple teams within a functional area*

As demonstrated in Figure 49, if the total estimated content of Functional area 1 would be 236 story points, total content of Functional area 1 can be allocated and monitored for individual teams based on individual team velocities.

### 3.5.2. Team structure

Since in current release development teams vary in expertise release planning and development efficiency could potentially be further improved by team re-structuring.

Due to complexity of program requirements, the ability to gain sufficient level of expertise and efficiency in each functional area requires time. Existing teams have been focused on multiple areas, thus expertise within different teams varies. If restructuring is possible, skills in different areas could be gathered from across teams and combined as illustrated in Figure 50 below.


*Figure 50. Expertise levels within an ideal team*

Re-organizing teams to combine different levels of expertise within same areas could create more efficient teams. The complexity of tasks is not always known beforehand and having such team structure would facilitate learning within teams and make each team capable to develop tasks of varying complexity.

Secondly, multi-skilled teams with different areas of expertise cause inaccuracies in estimation of velocity. Team velocity will depend on the combination of tasks assigned. Focusing teams on an individual area, on the other hand, creates a more stable development environment within teams, therefore improving the accuracy of velocity calculations and reduce complexities in release planning based on functional areas.

Due to technical complexity of the product, improving organizational processes is a complex process and needs to be taken gradually. Current organizational issues, in my opinion, cause major issues with the accuracy of planning and efficiency under Scrum.

### 3.5.3. Effort estimations

In order to improve accuracy of effort estimations, common methods need to be established between expert team and development teams. Also, changes to estimations during development need to be visible during development. In order to clarify and agree upon metrics, expert team needs to have clear and well-defined common baseline for estimations. Further, initial estimations should be allocated to re-factored user-stories and references, and initial size estimation needs to be referenced. Since planning is based on initial effort estimations on feature level, *planned estimations* should be tracked separately from *development estimations* on user story level, in order to monitor the actual changes in release size.

| Link to Reference | Planned estimation | Development estimation | Size per sprint (Planned) | Size per sprint (Developed) |
|---|---|---|---|---|
| Feature 1 | | | | |
| User story 1_1 | 21 | 21 | 4,2 | 4,2 |
| User story 1_2 | 45 | 45 | | |

*Figure 51. Initial effort estimations are documented on user story level*

A reference link to initial estimations for each user story would be a good solution for keeping a clear baseline. Provided references of each user story, teams would understand how the size of the requirements was estimated for individual user stories. If teams disagree regarding the size of initial estimations, they need to update estimations and update requirements list on the user story page to explain the basis for changes.

### 3.5.4. Velocity

Velocity calculations should be done for each team and functional area individually because teams work on different functional areas and each area varies. An example is illustrated in Figure 52 below:

| Team A velocity | Sprints | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 10 | 11 | 12 | 13 | 14 | 15 | ... | Average |
| Functional area 1 | | | | | | | | |
| Functional area 2 | | | | | | | | |
| Average/Total | | | | | | | | |

*Figure 52. Tracking velocity for each team in each functional area*

Team velocity is different in different functional areas, and will depend on task complexity. In order to have more accurate velocity estimations in each area, teams should avoid development different user stories in parallel. Since velocity will be based on the duration in sprints the complexity of tasks, to have more accurate velocity teams need to focus on one user story at a time. While it reality it might not be the case, the practical recommendation would be to avoid developing very different user stories in parallel.

Therefore, sprint velocity in each functional area should be calculated based on realized velocities of completed user stories and average overall velocity in each functional area equals to the average of realized sprint velocities.

### 3.5.5. Management tools

Especially in complex agile software development environments, visibility of development progress is important to support better decision-making and re-planning throughout the development period. Visibility depends on the accuracy and compatibility of tools and charts, which utilize data in the Product Backlog. In Flexi case study, it was identified that Product Backlog is missing essential data (effort estimations on two levels)

to track development based on initial plans. Also, it was concluded that current tools could not provide visibility on the sufficient level.

To improve visibility and to support decision-making based on facts and figures, progress should be tracked relative to the initial plans. Thus, release planning tools should track progress based on *planned* and *realized* progress, as well as to make *forecast* based on new data available after every sprint.

The Flexi development structure requires tracking development on multiple levels, where the release is on the top level. Also, additional visibility is needed over each functional area and on team level. Team estimates will exceed initial estimates if changes to initial feasibility studies are needed.

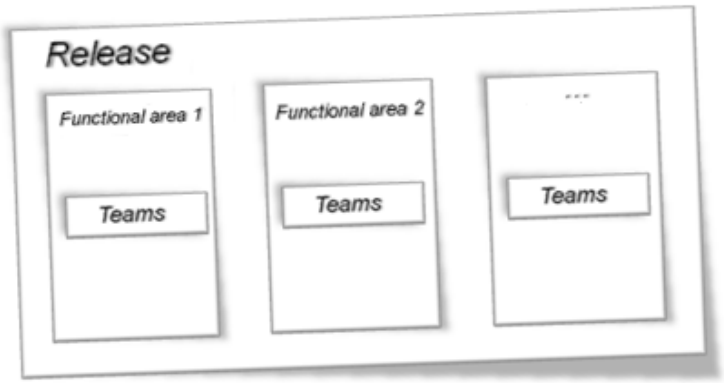


*Figure 53. The proposed development structure*

While the initial plans area based on the expected velocity and initial size estimations of features, the actual realized velocity and size of each feature is currently not known, and mistakes are not corrected in planning of future releases. When effort estimations are done on two levels, tracking should also be done on both levels to give visibility over changes in content and release plan.

| | Initial estimate | Team estimate |
|---|---|---|
| User story 1 | 6 | 6 |
| User story 2 | 4 | 10 |
| User story 3 | 4 | 5 |
| **TOTAL** | **14** | **21** |

*Figure 54. Tracking estimations on multiple levels*

By keeping track of changes in estimations, new information can be aligned and compared to initial estimations after every sprint throughout the release. Additionally,

additional visibility of the amount of “completed”, “ongoing” and “not started” story points in each functional area would provide more visibility over team and release status. This will allow management to make factual decisions and control progress of the release after every sprint.

In the next part, I will discuss and illustrate the proposed management tool in more detail. The aim of the new tool would provide more visibility on individual functional areas and teams, as well as to allow tracking release progress based on multi-level effort estimations.

### 3.6. Other recommendations

Flexi software complexity causes natural inaccuracies in initial estimations, and the success of release is based on adaptability of the development environment to occurring changes.

In Scrum, release planning highly relies on historical data. While effort estimations and velocity can be estimated more accurately for familiar functional areas and features., requirements of new complex features may be unclear and have risk to be under-estimated. Further, in completely new functional areas, velocity required and complexity are initially not known. Given complex and new content, it is recommended to include certain flexibility either to content or deadline until requirements and velocity are sufficiently clear. Alternatively, such features can be addressed through feature prioritization. To eliminate uncertainties from the beginning, new complex features may be developed first. Further, to grow expertise across teams, especially in areas where existing expertise level is low, pair-programming may be facilitated within sites. Teams working on same areas should preferably be located in the same location to facilitate communication and sharing of knowledge.

Finally, since the Flexi product is relatively new, the amount of maintenance work is yet quite unpredictable. As confirmed by the questionnaire results, maintenance work and other activities affect velocity. Therefore, it is recommended to distribute maintenance work reasonably across teams within same functional areas and keeping track of average amount of maintenance work and its effect on velocity over time. Alternatively, dedicated sprints can be planned for the teams to carry out the maintenance work.

# Practical part

## 4. New Product Backlog and Management Tool

In agile practices various instruments are used to facilitate the accuracy of planning and tracking release development, including metrics and visuals. Product Backlog is the most important artifact for gathering data and it should the fit specificities of the development environment.

In the previous sections, I have discussed the challenges and complexities, which currently exist in Flexi, as well as have proposed improvement actions, which need to be undertaken in order to optimize planning. Considering the proposed release planning methods, more structured data collection and tracking instruments would be needed.

To support the required changes, part of my task was to improve the existing Product Backlog and to build new managerial tools in Excel. General tools used in agile project management were discussed in the literature review section. In this section I will further discuss the proposed content of the new product backlog and managerial tools, including the benefits of such tools and methods to build these in Excel.

### 4.1. Building release management tool

As was stated earlier, implementation of more accurate management tools is required in order to increase visibility of development progress on different levels and to track changes to initial plans. Also, new tools are needed in order to improve velocity calculations.

While currently there are plenty of companies offering software solutions to support agile project management, such software is expensive and requires a lot of customization to fit the specificities of complex development environments, such as Flexi. Thus, to support the required changes and improvements, new Product Backlog- and Data Mining Spreadsheets were designed and developed in Excel.

In this chapter, I will briefly discuss the proposed new Product Backlog structure and tool implementation.

### 4.1.1. Product Backlog

Firstly, additional changes to the Product Backlog were proposed. While existing product backlog was comprehensive, the format lacked important information and data consistency. The major changes were related to effort estimations, but the Product Backlog was also further improved to support better functionality of spreadsheets and more efficient use of data.

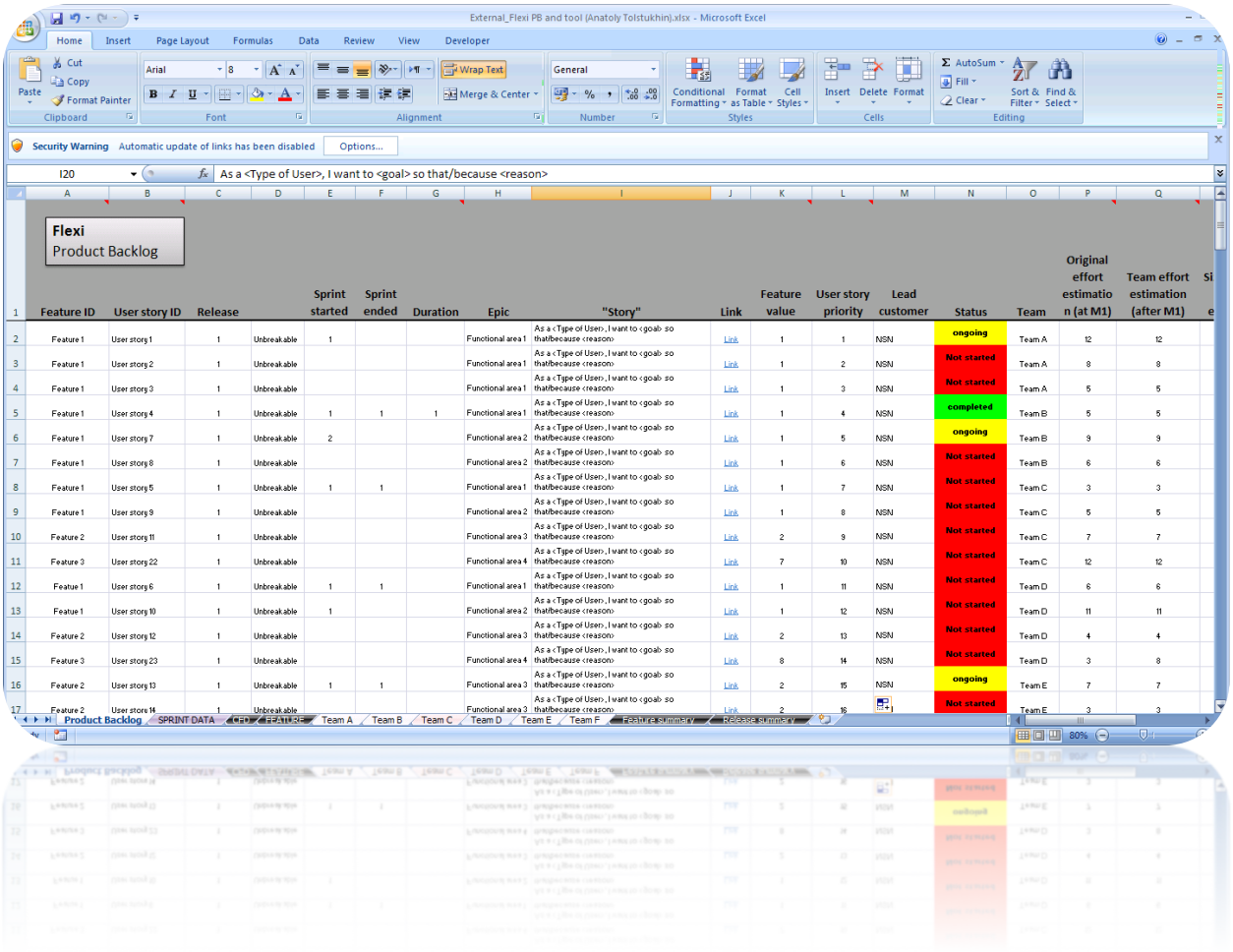


*Figure 55. Screenshot of the new product backlog*

The content of new Product Backlog data is proposed in *Table 8* below and further discussed in more detail. It can be compared to the old Product Backlog structure as illustrated in Table 8 earlier in my thesis.

| Column | Content Definition |
|---|---|
| **Feature ID** | Feature ID |
| **User story ID** | User Story ID |

| | |
|---|---|
| **Target Release** | Identifies release |
| **Compulsory** | Is the US compulsory or optional content of a release? |
| **Sprint Start** | In which Sprint a story started |
| **Sprint End** | In which Sprint a story will be/was completed |
| **Duration** | How many sprints completed user story lasted |
| **Epic** | Functional area a story belongs to |
| **"Story"** | User story itself<br>As a <Type of User>, I want to <goal> so that/because <reason> |
| **Feature value** | Feature value in release |
| **User story priority** | User story priority in development |
| **Lead Customer** | Key customer for whom the user story is intended |
| **Target date** | When should user story be ready |
| **Status** | User Story status (Completed, Ongoing, Not started, Cancelled, Re-factored) |
| **Team** | Allocated Team |
| **Link to Wiki** | Link to additional information on the story |
| **Original effort estimation at M1** | User story size in story points before M1 milestone (initial content estimation done by the expert team) |
| **Team effort estimation after M1** | User story size in story points after M1 milestone (development teams) |
| **Size per sprint (Team estimations)** | Average size per sprint of completed user stories |
| **Comments** | Space for comments |
| **User story added** | Date when user story was added |
| **Release Identifier** | Release identifier to link user story with burn-down |
| **Team category** | Which team category does the user story belong to? |

*Table 8. Proposed content of the new Product Backlog*

The highlighted rows in Table 8 represent new or modified columns as compared to the old product backlog. Next, I will explain the application and value of each data column in the new product backlog to Flexi agile release planning and tracking.

**Feature ID**

Currently Excel is used to keep a single product backlog and track release progress; also multiple user stories belong to different features. In order to track progress of features, a separate column is required which identifies specific feature to which a user story belongs. Collecting this data will help calculate and track user stories on feature level.

**Sprint Start, Sprint End and Duration**

Currently, user stories last more than one sprint. Sprint Start column is used to estimate the beginning sprint. Further, user stories may begin earlier or later, and it is important to

update Sprint Start column when the user story development actually begins. Once the user story is the completed, relevant Sprint End needs to be identified.

Additionally, since most user stories last longer than one sprint, the duration should be included into product backlog as a separate column to provide better visibility. This column would also simplify velocity calculations and formulas in the management spreadsheets.

**Feature Value and User Story Priority**

Even though the following columns existed in the old product backlog, these columns were mistreated, e.g. user story priority was tracked in feature value column. The following columns are especially important for teams to manage the user stories and to prioritize development order. Briefly, it would be recommended to keep track of feature value as the first decimal for priority and user story priority within each feature as the second decimal.

**Link**

Since clear references are needed to manage the content of user stories on multiple levels, each user story needs clear references to effort estimations. It was recommended to build pages for each user story. These pages could be viewed and modified by the expert team, as well as the assigned development teams. This will facilitate understanding about requirements and effort estimations. These pages need to be easily accessible and traceable through a Web-links in the product backlog.

**Original effort estimation at M1**

The original effort estimations of user stories and requirements are done prior to the main development period. Therefore, these estimations need to be tracked throughout the project. These estimates can be modified before commitment to deadline, and if a user story is added after M1, the respective cell should be empty (equal 0). This information allows tracking changes in the release content before- and after commitment to deadline.

**Team effort estimation after M1**

The expert team does primary effort estimations and provides the references in a shared web page. Further, when the development teams complete brainstorming the user story requirements, story points need to be updated if some requirements are missing or

have changed compared to the initially referenced estimations. Changes to estimations should be further referenced on the same user story web page to update requirements and estimations.

**Size Per Sprint (Team estimations)**

Since user stories often last more than one sprint, the realized user story size per sprint of completed stories needs to be documented separately. The size per sprint figure is used in velocity calculations to compute the amount of story points teams are actually capable to complete in a single sprint.

**User Story Added (Date)**

Different stakeholders continuously add user stories to the product backlog. Therefore, the information on the date when a user story was added is essential. The following information is useful to improve information sharing across many people in the organization. This column was missing from the existing product backlog.

**Team Category**

In order to track feature content across different team categories, each user story should be marked with a relevant team category, i.e. Feature Development, Performance testing, Network Verification or Common team. This information about the features will support tracking ongoing releases and planning future releases across different categories.

As discussed, the proposed product backlog contains new valuable information tailored to the specific needs of the development environment in the organization. This guide may serve to illustrate what kind of data is important in Scrum. In the next chapter, I will discuss the content of new data spreadsheets and their link to product backlog.

### 4.1.2. Management tools

Apart from new product backlog, new data mining spreadsheets were created to provide more accuracy in velocity calculations, as well as better visibility through burn-down charts on all stages including planning, tracking and forecasting.

Since teams and functional areas differ, the new tools provide visibility of progress of individual teams in separate areas. Further, while keeping the initial plans, the advanced burn-down charts offer projected forecast based on changes after every sprint. The

empirical data behind charts is calculated from separate teams and functional areas, thus making the critical path identifiable and progress of individual teams accessible. Further, the content and functions of spreadsheets of the new tool will be discussed.

#### 4.1.2.1. Team spreadsheets

The most important component of the new tool is a set of individual spreadsheets with data calculated for each team separately. In each separate spreadsheet the content is divided into two parts: *Team Data* and *Release Burn-down*.

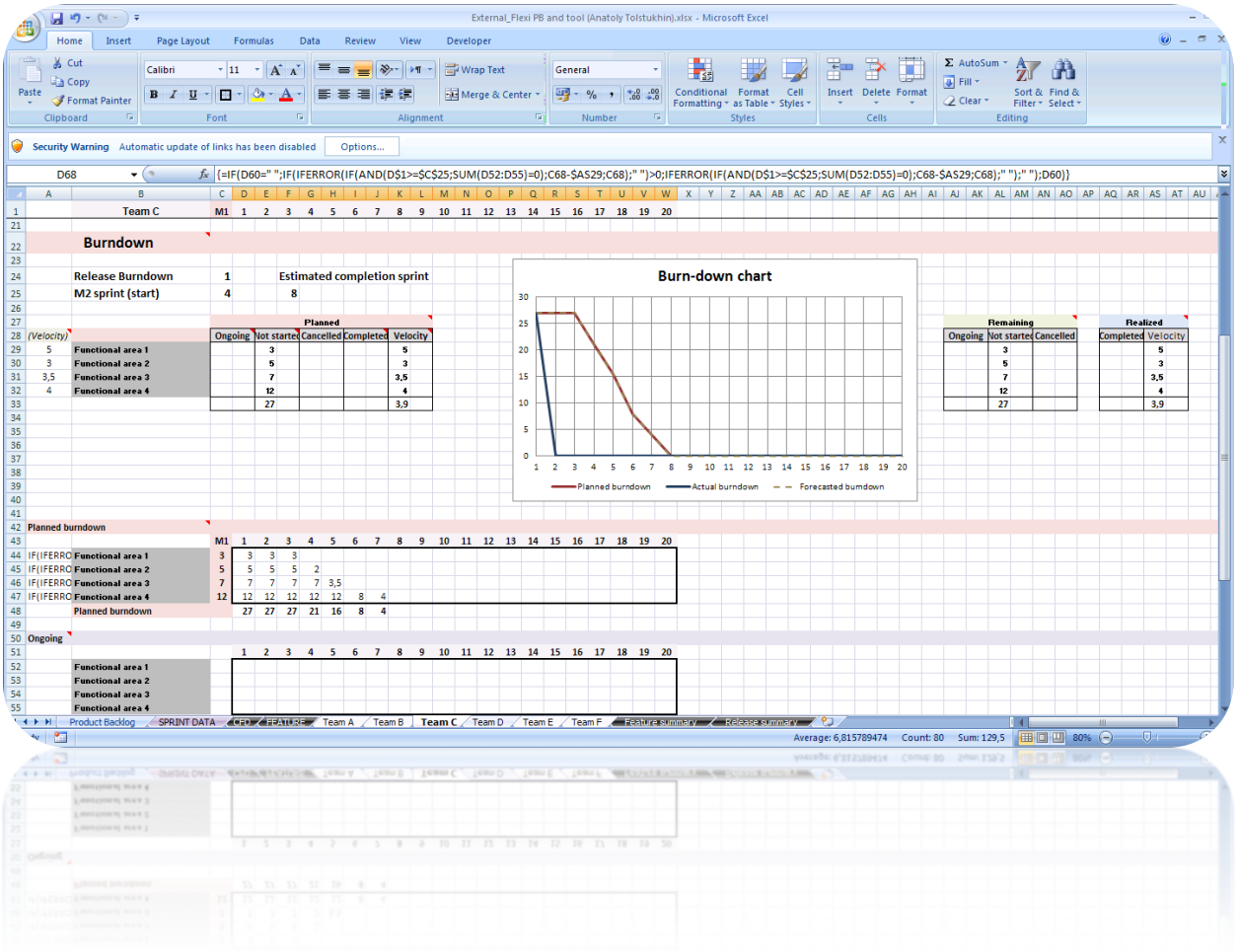


*Figure 56. Screenshot of the Team Data spreadsheet (E.g. Team A)*

Team Data includes two tables calculating the amount of completed story points over different sprints. This table provides visibility about accomplishments of each team across different functional areas. The second table calculates realized velocity by distributing story points over the sprints when related user stories were developed. For example, if a

story of size 15 story points lasted 5 sprints, then the average 3 story points per sprint would be distributed in this table. One should keep in mind that the accuracy of the velocity depends on parallel activities across multiple areas and maintenance work. It is recommended to keep track and possibly update the data manually if required, to have the most accurate estimator of average velocity for each team.

The second part of the spreadsheet i.e. Burn-down, includes relevant team data necessary to manage release planning, which includes planned, realized, remaining and cancelled story points for the ongoing release.

Team burn-down chart is constructed based on the release burn-down identifier specified in the product backlog. For example, if the main ongoing release is Release 7, all user stories, which are expected to be ready during the respective development period, should have the same release identifier. This also includes internal user stories and requirements from other releases. This will allow to track actual team load during each release and expected completion sprint. Further, actual release content can be separated, which will be discussed later.

The tool predicts burn-down for initially planned content, which is locked at M1. Further, the tool tracks actual burn-down of user stories and makes forecasted burn-down based on updated story points and velocity after every sprint. As a result, these inputs create separate lines on the burn-down chart: planned burn-down (fixed at M1), actual burn-down (updated every sprint) and forecasted burn-down (forecasts expected completion date based on updated velocity and total remaining effort). Additionally, the tool includes the expected completion sprint of the ongoing effort. In summary, the following factors are considered in calculations for burn-down of story points:

1. Burn-down begins after M1 sprint (when initial planning is done)
2. If a team is assigned to multiple functional areas, then burn-down will be forecasted for functional area 1 based on team velocity in that area. When the team completes functional area 1, the burn-down of functional area 2 will start from that sprint.
3. If there are ongoing user stories, the tool uses average velocity to estimate when the ongoing story points would be completed and then predicts burn-down of the remaining content from that sprint.

The calculations in these sheets are updated automatically, when the last completed sprint is updated in the spreadsheet "Sprint Status" which will be discussed later. In order to track changes in release content, initial estimations and forecasted burn-down need to be locked at M1 (e.g. copy-pasted as values, or macro applied), so that the data in Planned Effort Tables does not change after that.

The tool calculations are based on product backlog data and functions were applied on single- and two-dimensional scales (i.e. overall story point status at the moment, and historical progress on sprint timeline). Examples of logical functions for these spreadsheets can be found in Table 9 below.

Example: Team C, Sprint 2, Functional area 3, Planned M1 at Sprint 4, Release 1

***Completed effort of Team C (Sprint 2, Functional area 3)***
*= Sum of team effort estimations of user stories,* at Sprint 3
*IF Team Name = Team 3;*
*IF Sprint End = 2;*
*IF Status ="completed";*
*IF Functional area = Functional area 3; Otherwise cell empty.*

***Velocity of Team C (Sprint 2, Functional area 3)***
*= Sum of average effort per sprint of completed user stories,* at Sprint
*IF Team Name = Team C;*
*IF Sprint End <= 2;*
*IF Sprint Start >= 2;*
*IF Status ="completed";*
*IF Functional area = Functional area 3; Otherwise cell empty.*

***Estimated burn-down of ongoing effort for Team C (Sprint 2, Functional area 3)***
*= Sum of (Team C effort estimations – (Velocity of Team C in Functional area 3 * (Current sprint - Sprint Start))),*
*IF current sprint is* Sprint 2
*IF Team Name = Team C,*
*IF Release Burn-down = 1;*
*IF Status ="ongoing";*
*IF Functional area = Functional area 3; Otherwise cell empty.*

***Planned burn-down for Team C (Sprint 2, Functional area 3, M2 at Sprint 4)***
*= ("Not started" effort of Team C in Functional area 3 – velocity of Team C in Functional area 3)*
*IF Current sprint >= 4;*
*IF Sum of effort in previous functional areas is 0;*
*IF Sum of estimated ongoing effort burn-down at sprint >= 4 is 0; Otherwise cell empty*

***Actual burn-down of story points for Team C (Sprint 2, Functional area 3)***
*= Sum of Team C effort estimations*
*IF Team Name = Team C,*
*IF Release Burn-down = 1;*
*IF Status ="completed";*
*IF Functional area = Functional area 3; Otherwise cell empty.*

| |
|---|
| ***Forecasted burn-down of story points for Team C (After Sprint 2, Functional area 3, M2 at Sprint 4)***<br>*= Sum of (Team C remaining effort estimations in Functional area 3 in Sprint 2 – (Velocity of Team C in Functional area 3)*<br>*IF no ongoing user stories;*<br>*IF Sprint > than last completed sprint;*<br>*IF Sprint >= M2 Sprint; Otherwise cell empty.* |

*Table 9. Functions in the Teams Spreadsheet*

#### 4.1.2.2. Feature/PET/NEVE spreadsheets

The data of individual teams' spreadsheets needs to be combined for each team category, for example Feature teams are Teams A, B, E and F; Team C is a PET and Team D is a NEVE team.

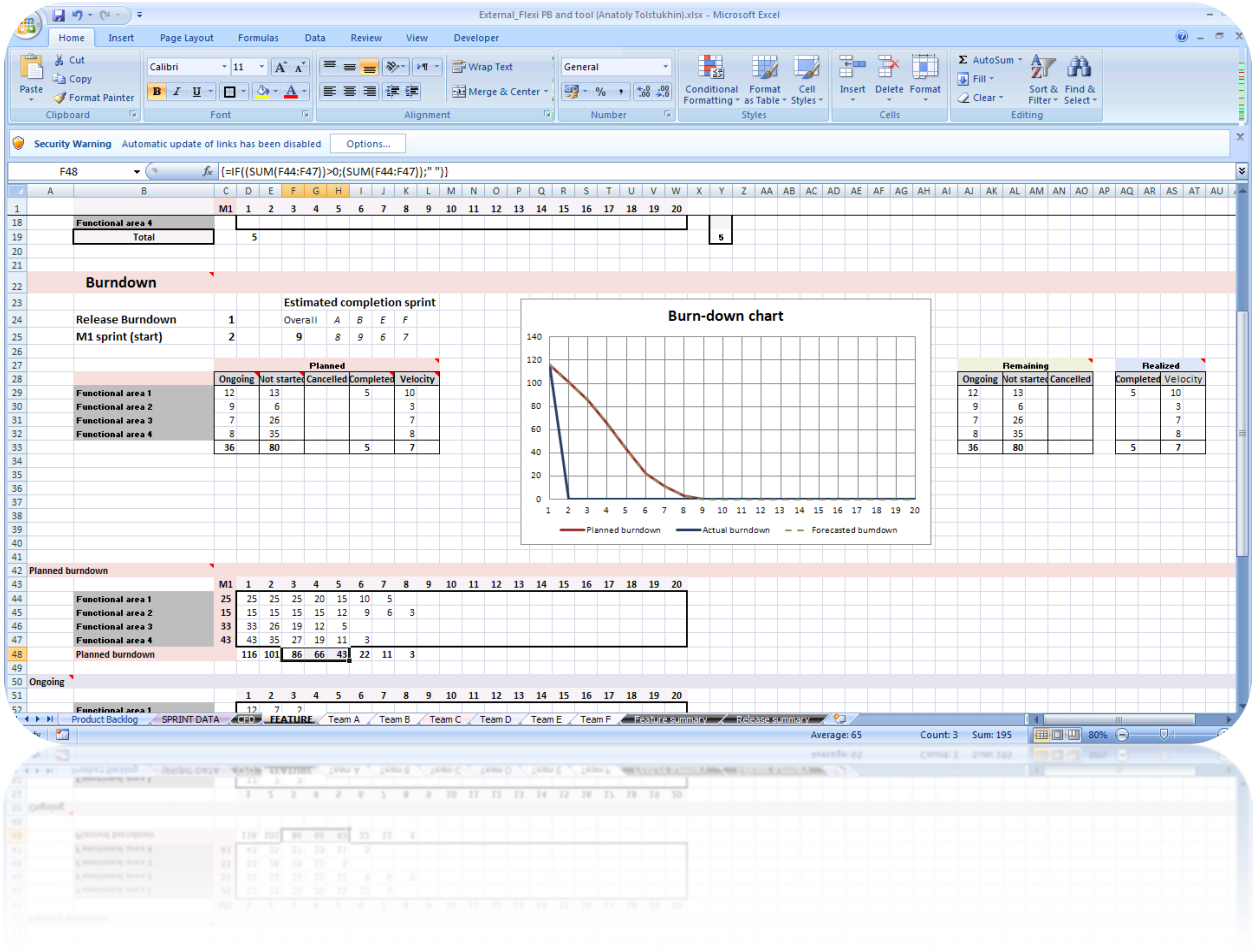


*Figure 57. Screenshot of a Team Category Data spreadsheet (E.g. Feature Development)*

While planning would be based on the individual team level, the overall burn-down allows viewing the overall release readiness and estimated burn-down figures of all feature teams combined. This chart allows observing the total burn-down of the release, as well as

tracking the expected completion sprint of each team. Estimated completion sprint for each team is visible next to overall burn-down chart. This way, management can easily identify bottleneck and make decisions to keep overall release progress on track.

| Example: Feature teams, cell D7 |
|---|
| ***Cell value in Feature Spreadsheet*** |
| *=SUM(Team A!D7;Team B!D7;Team E!D7;Team F!D7)* |

*Table 10. Function in Feature teams Spreadsheet (e.g. Cell D7, all cells expect average velocity)*

#### *4.1.2.3. Sprint Data*

The visibility of overall status of Team- and Functional area Readiness helps tracking the development progress of releases.

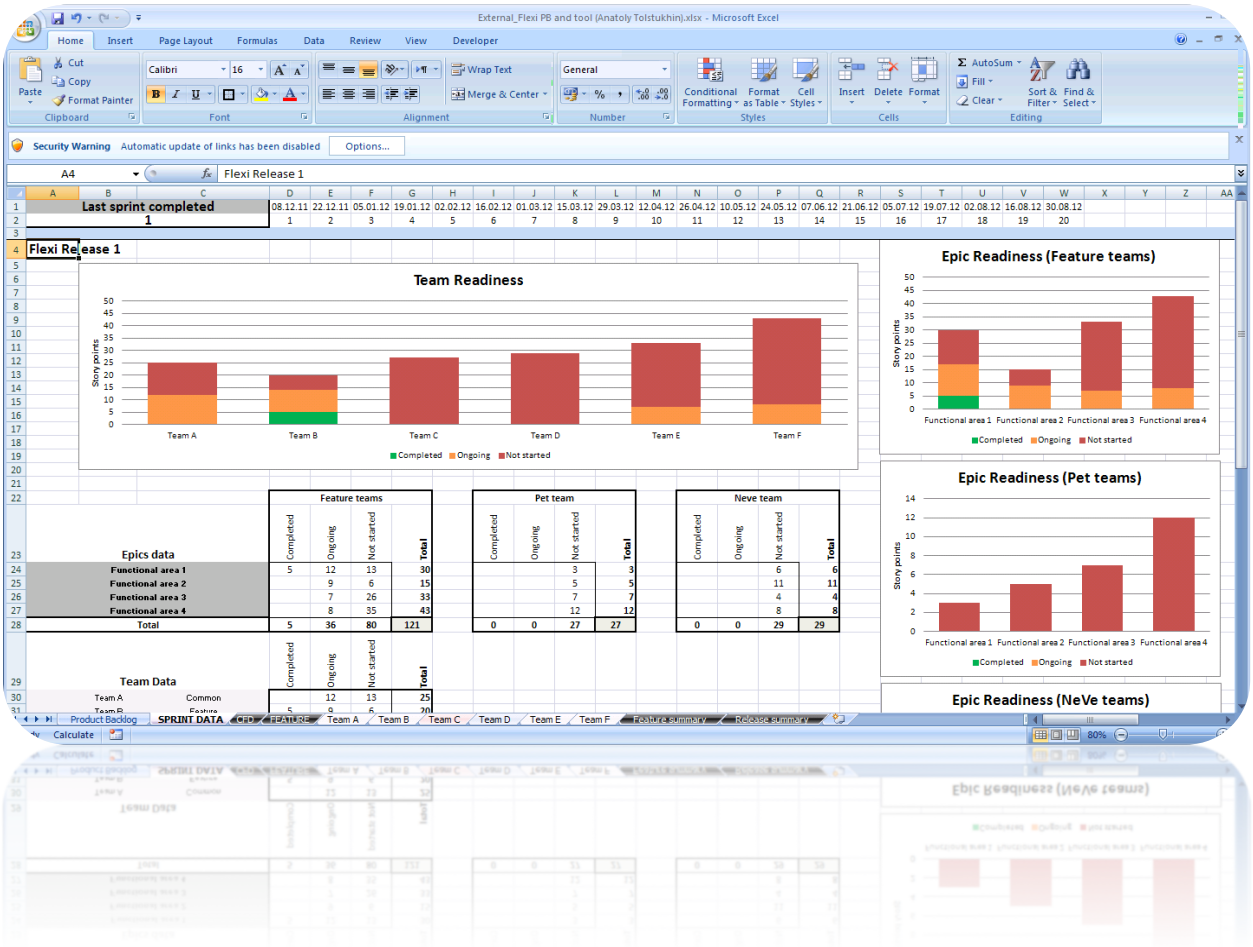


*Figure 58. Screenshot of a Sprint Data spreadsheet*

A spreadsheet was designed to visualize the amount of work across teams and functional areas, and includes the amount of completed, ongoing and not started user

stories separately for each team and functional area. Such information provides better allocation of user stories and higher visibility on overall progress by each team and in each functional area in total.

Furthermore, the cell in the top left corner of this spreadsheet indicates last completed sprint. This cell needs to be updated together with the product backlog data when sprint begins. Updating and saving this cell should be done when the information in the product backlog is updated and a new sprint officially begins. This following action generates new burn-down charts based on updated information from the last completed sprint.

#### *4.1.2.4. CFD (Cumulative Flow Diagram)*

Cumulative flow diagram chart provides an overall visual representation of progress over sprints on the amount of completed, ongoing, not started and cancelled story points.

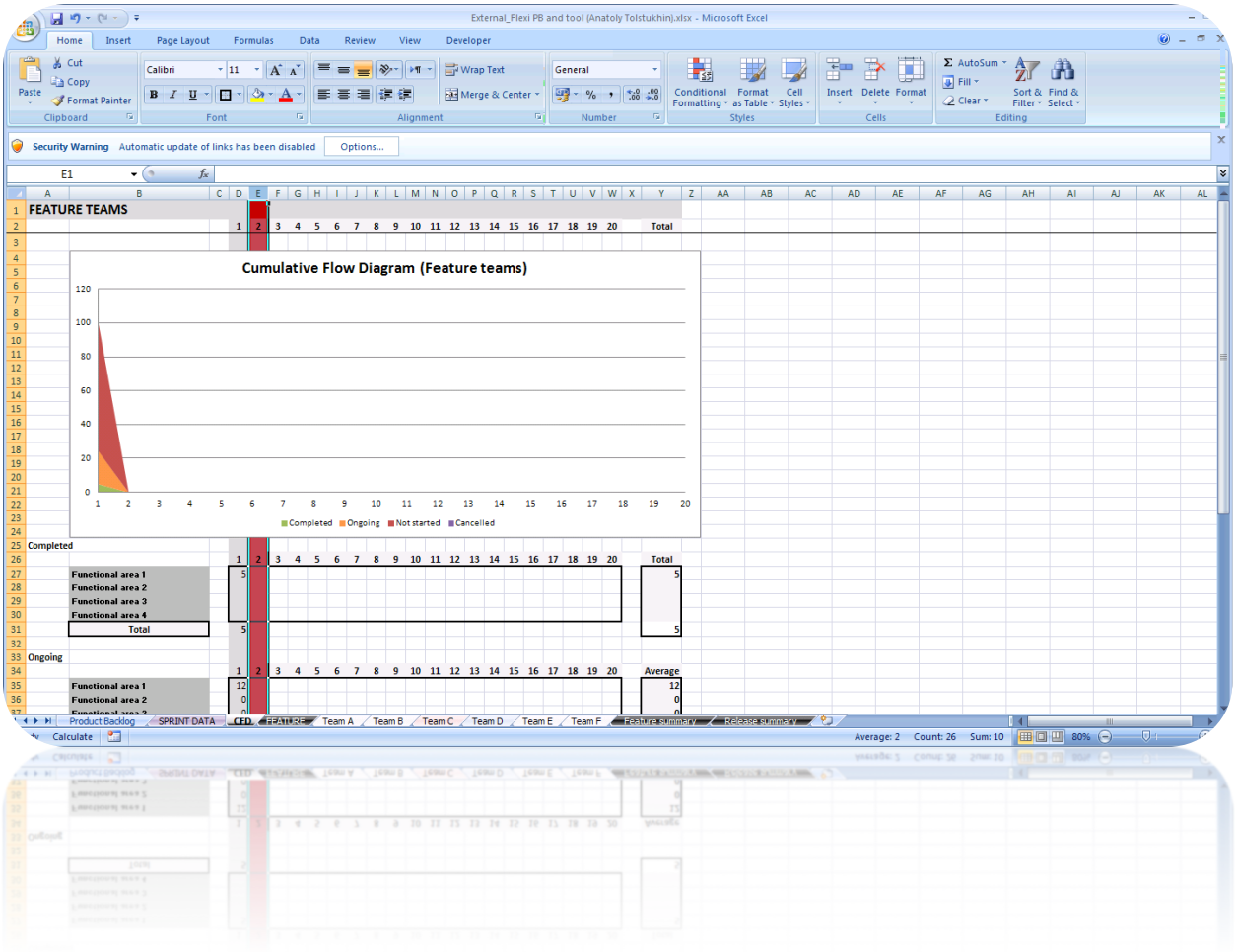


*Figure 59. Screenshot of the CFD spreadsheet*

Collecting the historical information on the total story point status in each functional area helps in decision-making regarding the specific functional areas over the long-term period, such as the lifetime and changes in the workload and expertise level per area.

Furthermore, the data can be used to observe the progress of the individual functional areas over a period of time. Specifically, tracking this information on the functional area level allows identifying e.g. whether a functional area has grown or decreased over time or how the efficiency and methods changed over time. This information helps management decide, for example, whether a functional area needs more or less teams in the future.

Automatically generated figures in the highlighted column show the information about the story points related to the last completed sprint. These figures need to be captured in the beginning of the following sprint. For example, when Sprint 2 is completed, figures in the highlighted column of CDF spreadsheet will be updated. The red columns need to be copy-pasted as value or alternatively, a macro needs to be applied to save the data. These actions are required to allow tracking actual changes in status of user stories. The formulas of this spreadsheet are linked to the data of the Feature team category spreadsheet.

#### *4.1.2.5. Feature Summary*

Feature summary spreadsheet gathers valuable information about individual features in each release. It provides visibility about the total size (story points) of each feature, which was originally planned and actually realized during every release and by different team categories (i.e. Feature, PET, and NEVE). This data is very valuable when planning future releases because it allows more accurate estimation of similar features, as well as shows how much each feature has changed in relation to initial plan. Also, it allows comparing performance of different teams in similar features. This table contains the feature list and different functional areas and teams.

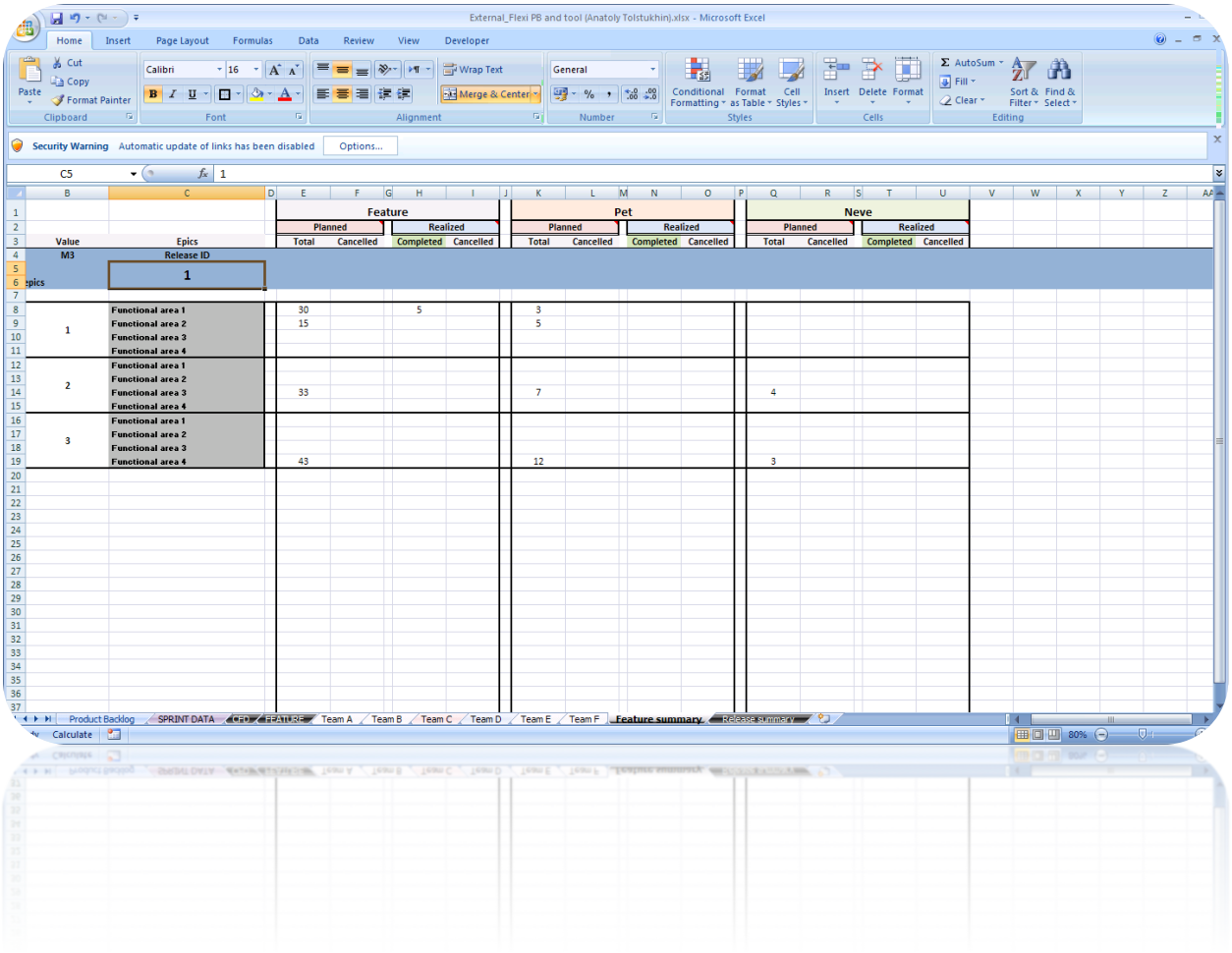


*Figure 60. Screenshot of the Feature Summary spreadsheet*

When the commitment to initially planned content is done, planned effort estimations need to be locked, whereas the realized size should be locked when the release is completed. The features list needs to be updated by the stakeholders adding user stories. Functions in this spreadsheet are based on the information from the product backlog regarding Feature ID, Release ID and the Functional area.

Example: Release 1, Feature 2, Functional area 3, Team category PET

***Planned total***

*= Sum of initial effort estimations*
*IF Feature ID = 2*
*IF Status ="completed" OR "ongoing" OR "not started";*
*IF Release Burn-down = 1;*
*IF Functional area = Functional area 3;*
*IF Team Category = PET;*
*Otherwise cell empty.*

***Realized total / Completed***

| |
|---|
| *= Sum of team effort estimations*<br>*IF Feature ID = 2*<br>*IF Status ="completed"";*<br>*IF Release Burn-down = 1;*<br>*IF Functional area = Functional area 3;*<br>*IF Team Category = PET;*<br>*Otherwise cell empty.*<br>***Cancelled***<br>*= Sum of initial effort estimations*<br>*IF Feature ID = 2*<br>*IF Status ="cancelled"";*<br>*IF Release Burn-down = 1;*<br>*IF Functional area = Functional area 3;*<br>*IF Team Category = PET;*<br>*Otherwise cell empty.* |

*Table 11. Functions in the Feature Summary spreadsheet*

#### *4.1.2.6. Release Summary*

The Release summary spreadsheet contains the list of releases, including the release identifier, respective milestones as well as separate functional areas and team categories. Moreover, it contains information about the overall planned and realized size of each release. Information in the planned columns needs to be locked at M1 and in the realized columns at M4 milestones. These calculations will provide information about how much the realized release content has changed compared to the initial plans in total and separately in each functional area. Also, this spreadsheet provides information regarding sprints for each milestone.

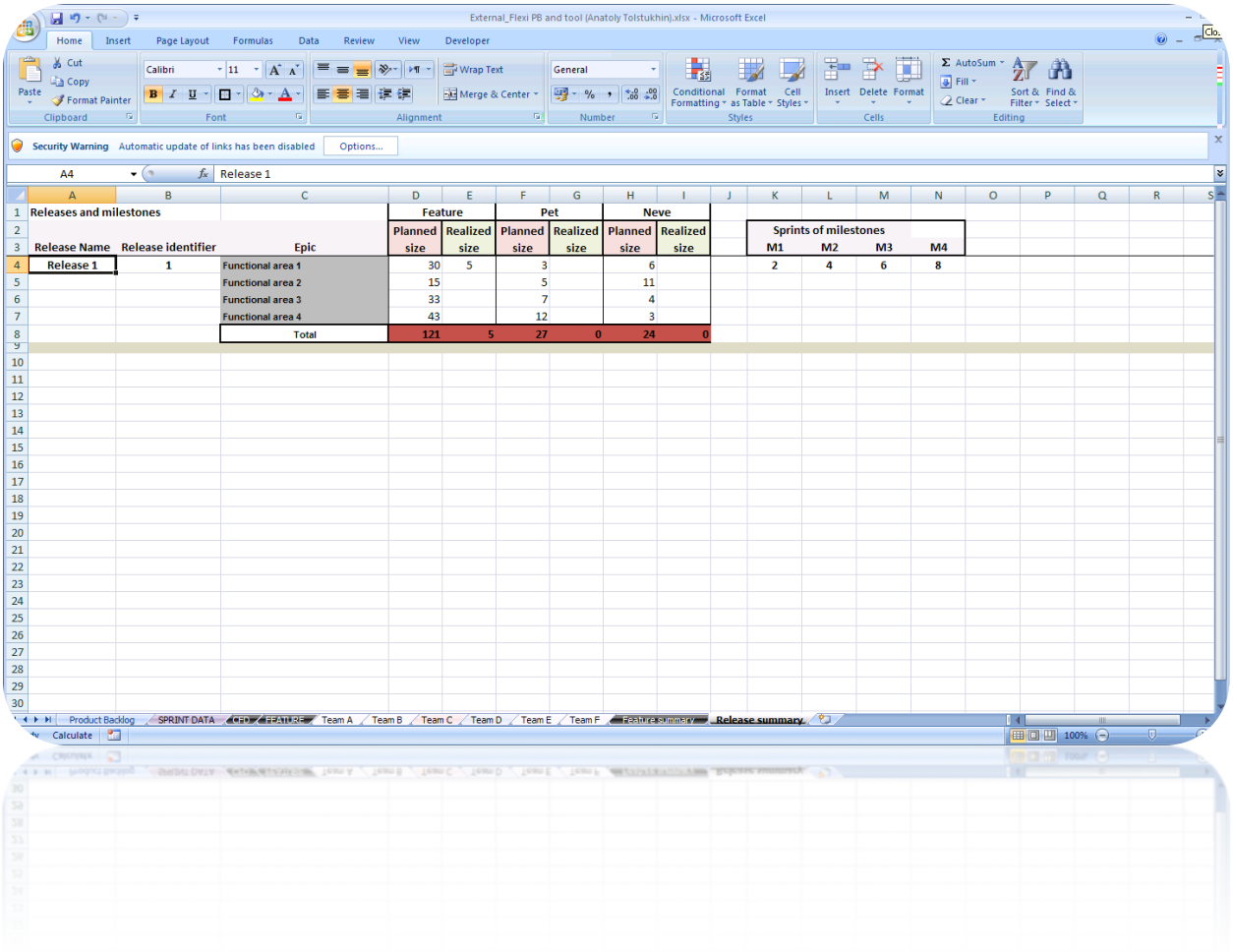


*Figure 61. Screenshot of the Release Summary spreadsheet*

If needed, also the planned and realized average velocities can be added to this spreadsheet based on existing calculations in the release burn-down spreadsheet of team categories (e.g. the Feature spreadsheet). However, since the velocity data is available in the burn-down spreadsheet, and because velocity may depend on other factors than plain team performance, it was suggested to monitor effort estimations and plan releases based on average velocities of individual teams independently from individual releases.

The Table 12 below shows the functions applied in "release summary" spreadsheet of the tool.

| Example: Release 1, Functional area 2; Feature team category |
|---|
| ***Planned total*** |
| *= Sum of initial effort estimations* |
| *IF Release ID =1;* |
| *IF Status ="completed" AND "ongoing" AND "not started";* |

| |
|---|
| *IF Functional area = Functional area 2;*<br>*IF Team Category = Feature;*<br>*Otherwise cell empty.*<br>***Realized total / Completed***<br>*= Sum of team effort estimations*<br>*IF Release ID =1;*<br>*IF Status ="completed";*<br>*IF Functional area = Functional area 2;*<br>*IF Team Category = Feature;*<br>*Otherwise cell empty.* |

*Table 12. Functions in the Release Summary spreadsheet (e.g. Release ID 1, team category "Feature", Functional area 2)*

## 4.2. Evaluation and deployment

The new product backlog and managerial tools provide more information about the release development processes and with higher accuracy than the previous tools. The tool was presented to the management and it was put in the deployment stage during the time the thesis work was completed.

Based on the empirical findings, the new tool reflects the necessary changes related to the planning and tracking methods existing in the Flexi development environment. Furthermore, the tool provides more accurate velocity calculations and burn-down charts, as well as collects more valuable information about release content and size to support planning and tracking.

As the tools deliver better visibility over the development progress, the results highlight the importance of a more structured and organized approach to the allocation and development of user stories across individual teams, standardized effort estimations and right methods to calculate velocity, particularly in a complex distributed development environment.

The deployment of the new product backlog and management tools is simple, since it only requires replacing the old product backlog file with the new version to collect additional information. The instruments were created based on the empirical findings and results regarding organizational and managerial processes under the Scrum framework, as well as supported by in-depth studies of existing development processes in the case company. Moreover, since the new tools were tailored specifically to meet the needs of the

Flexi development environment, the additional data input was identified to be essential for planning and managing releases. Therefore, enriching the product backlog and data mining sheets was not seen as a drawback, but rather an important improvement.

Additionally, new tools were applied and tested in planning a new release. It proved to be offering more flexibility and higher visibility in release planning and tracking activities on multiple levels. Apart from tracking changes to requirements and velocity on the team level, the tool also makes re-planning and forecasting after every sprint possible based on right application of new information. This allows the management to make decisions on time to maintain the planned schedule.

The new instruments should support factual decision-making and provide improved methods to plan and track release progress under the Scrum principles, thus leading the case company towards overall optimization of release planning and development processes under Scrum.

Overall, this section illustrated the importance of high visibility and appropriate tools to support accurate planning and monitoring development under Scrum methods. Furthermore, this section also pinpoints that in complex development organizations Excel tools can be of high value. Additionally, this section may serve some as a useful guide to designing their own tailored tools.

## 5. Summary and conclusions

Relevant recent literature concerning agile project management was reviewed. Further, this thesis acted as a practical case study of Scrum applied to develop large-scale and complex software in Nokia Siemens Networks.

Keeping the theoretical framework in mind, the following research showed that while Scrum methods are also applicable in large software development projects, planning and tracking methods are more complex and require different approach than in simple environments. The original goal was to study velocity and effort estimations in an attempt to improve the accuracy and optimize release planning. This thesis has identified and addressed the research questions defined in the introduction, including more accurate methods to calculate velocity, to standardize effort estimations across the distributed

teams, as well as to improve the accuracy of planning and tracking methods in the case company. Moreover, the new product backlog and tools was developed to support the Flexi management with planning and tracking of the future releases. By continuously addressing the specificities of the Flexi software product and development environment throughout the case study, the empirical findings related to agile planning and tracking methods were proposed, which have not been previously discussed in the literature on the Scrum methodologies. The findings discussed in this thesis are particularly valuable because the study was carried out in a real business situation.

One of the main findings from the case study at NSN was that agile project management in large-scale setting requires a specific structure and a more detailed approach Since larger and more complex products, such as Flexi, are combined of technically different areas, a different approach to planning may be needed. On the other hand, this thesis offers another approach to manage large agile projects, which was not earlier discussed in the literature. The results of this thesis suggest that teams may not be fully multifunctional.

While the Scrum emphasizes cross-functional teams, the case study identified that in complex software projects team knowledge is limited to specific areas of the overall product. This thesis provided practical methods to manage the distributed knowledge along with the theoretical framework of Scrum methodologies. Due to the existence of different functional areas of the software, which have been developed by different teams, it was concluded that complex Scrum projects consist of multiple parallel projects. Thus, to manage and address such complexity in a more structured way, different areas within a product can be structured as separate smaller projects. Since team expertise in different functional areas varies, velocity also needs to be calculated for different functional areas separately.

Moreover, empirical recommendations were provided on how to manage the resources including the allocation and development of user stories, planning and tracking development progress and calculating velocity. Findings suggest that lack of visibility in the development progress decrease accuracy and complicates planning and tracking throughout product development life cycle. Among other authors, Cohn (2006), Heikkilä et al. (2010) and Vähäniitty et al. (2010) have discussed content management in a multi-team scenario. This thesis contributes through a practical perspective about possible challenges

and methods how the resources can be optimally managed in a complex environment, including planning and tracking development on multiple levels.

The research showed that in large and complex agile projects maintaining accuracy of effort estimations is especially challenging. In Flexi, initial planning is necessary prior to the actual development in order to estimate the scope of the project and the release date before investing resources in development and assigning requirements to development teams. In the initial planning, effort estimations serve as the roadmap for the actual development, whereas actual release content and velocity are known when the requirements are actually developed by the teams.

This thesis also identified that in a complex setting effort estimations need to be properly managed during development to provide visibility about actual progress based on new lessons learnt after every sprint. While Cohn (2006) discussed the importance of common baseline for effort estimations, this thesis showed that from practical perspective maintaining common estimations might be challenging. Further, methods on how to maintain the same baseline for effort estimations across different teams and on different levels of requirement hierarchy were identified and provided. Further, Product Backlog and tools were created, which allow to control effort estimations throughout release development and accuracy of initial estimations to be adjusted during development.

Supported by the case study, the thesis offers empirical methods to manage velocity and output of the Scrum teams provided varying knowledge areas and user story duration. In existing literature about Scrum, a lot of emphasis was put on fitting user stories in one sprint. While it is a sold theory, in practice this may be either impractical or unfeasible. It was identified that while the overall development follows agile principles, individual teams may work on requirements, which may take more than one sprint to develop. Further, duration of user stories is not only effected by actual size, but also by complexity and team expertise. With user stories varying in duration and developed in parallel, it was identified that duration of individual user stories needs to be included in velocity calculations.

Additionally, keeping in mind Scrum methodologies, this thesis served as the basis for making new product backlog and agile project management tools. The tools were created in Excel to assist planning and tracking development under Scrum in the case company.

Due to the specificities of the development environment, from its value to NSN the tools may compete with agile management software currently existing in the market.

## 5.1. Practical implications

From the applied perspective, the results proved to be valuable and practical for the case company. Therefore, this thesis may be useful for other large organizations with similar product specificities, which current face difficulties in maintaining accuracy of planning and tracking under Scrum. Additionally, new empirical management tool designed specifically for the case company offers instruments to make better decisions related to planning and tracking. It also provides higher visibility about development progress to support re-planning after every sprint. Since this thesis addresses managing major challenges in release planning, the results of this thesis may be beneficial to larger organizations adopting- or considering adoption of Scrum methodologies in a complex software development.

## 5.2. Academic implications

From the perspective of the academic research, this thesis provides new empirical findings about specificities of agile project management in large and complex software development projects. Particularly, it contributes to the exiting studies on agile release planning and project management carried out by Cohn (2006), Vähäniitty et al. (2010), Heikkilä et al. (2010) and Machnic (2011). This thesis offers a multi-project approach to managing release planning in complex projects with varying areas of expertise and new methods to calculate velocity when user stories last multiple sprints. Additionally, methods to adjust and standardize effort estimations across multiple teams and at different levels were identified and presented from the practical angle. Finally, the results may be useful to a reader questioning the challenges caused by agile methods in large organizations.

## 5.3. Further Studies

The main focus of this study was identifying problems with release planning methods and providing improvement recommendations on optimizing methods and metrics to improve accuracy of release planning and monitoring, and included applying results in a new management tool. This tool served as an additional contribution to the organization

and was built to support the main findings and to provide better tools than existing prior to completed work. While the new management tool proved to be well integrated with current development environment in Flexi and results created convincing support for applying the methods proposed in this study, given the overall scope of this thesis and limited time frame, further studies could be carried out to further improve proposed management tools and to fully automate tracking activities, e.g. applying macros. Also, studies could be carried out to access the value and suitability of agile project management software currently existing on the market, i.e. to identify whether offered solutions could be tailored sufficiently to fit the company needs based on the main findings in this thesis.

Further, since the main focus of the thesis was on optimizing methods and metrics of regular Scrum development teams, the findings regarding velocity calculations and planning methods should also apply to other team categories (e.g. PET, NEVE) given that their practices also follow agile principles. During my work, it was noticed that activities of some teams (e.g. Common teams) have been quite unique over sprints and thus could be planned based velocity. Further studies could be carried out to analyze inter-dependencies across different team categories and to improve performance of nontraditional agile teams to cut overall time to market.

Overall, this thesis managed to integrate theoretical knowledge about the Scrum methodologies with a practical case study to provide empirical findings related to release planning optimization and agile management in large and complex projects.

While this thesis generally provides empirical results and findings about the agile project management in large distributed projects, thus adding value to the existing literature, alternative studies may be carried out to identify other specificities and methods needed in differing agile development environments and other industries.

## References

Alleman, G. (2002), “Agile Project Management Methods for IT Projects”, The Story of Managing Projects: A Global, Cross–Disciplinary Collection of Perspectives, Greenwood Press, Berkeley 2002.

Augustine, S., Payne, B., Sencindiver, F., and Woodcock, S. (2005), “Agile project management: steering from the edges”, Communications of the ACM, 48(12): 85-89.

Beck, K. et al. (2001), “Manifesto for Agile Software Development”, www.agilemanifesto.org, accessed 28.7.2011.

Boehm, B. (2002), “Get Ready for Agile Methods, with Care”, IEEE Computer, 35(1): 64-69.

Buglione, L., Abran, A. (2007), ”Improving estimations in Agile projects: Issues and avenues”, Proceedings of Software Measurement European Forum (SMEF), 265-274.

Ceschi, M., Sillitti, A., Succi, G., Panfilis, S., (2005), “Project Management in Plan-Based and Agile Companies”, IEEE Software, 22(3): 21-27.

Chow, T., Cao D. B. (2008), “A survey study of critical success factors in agile software projects”, Journal of Systems and Software, 81(6): 961–971.

Cohn, M. (2006), “Agile Estimating and Planning”, Boulder: Prentice Hall, 2006.

Dyba, T., Dingsøyr, T. (2008), “Empirical studies of agile software development: A systematic review”, Information & Software Technology, 50(9-10): 833-859.

Eduardo, M., Pierre, B. (2010), “Agile monitoring using the line of balance”, Journal of Systems and Software 7(83): 1205-1215.

Grimstad, S., Jørgensen, M., Moløkken-Østvold K. (2005), "Software effort estimation terminology: The tower of Babel", Journal of Information and Software Technology, 48(4): 302-310.

Shahir, H. Y., Daneshpajouh, S., Ramsin, R. (2008), "Improvement Strategies for Agile Processes: A SWOT Analysis Approach", Sixth International Conference on Software Engineering Research, Management and Applications, Czech Republic, Prague, 221-228.

Hass, K. (2007), "The Blending of Traditional and Agile Project Management", Project Management World Today, 9(5), http://www.pmforum.org/library/tips/2007/PDFs/Hass-5-07.pdf, accessed 28.7.2011.

Heikkilä, V., Rautiainen, K., Jansen, S. (2010), "A Revelatory Case Study on Scaling Agile Release Planning", Proceedings of the 36th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2010), 289-296.

Kittlaus, H. B., Clough, P. (2009), "Software Product Management and Pricing: Key Success Factors for Software Organizations", Germany, Berlin 2009.

Larman, C. (2006), "Agile and Iterative Development: A Manager's Guide", Agile Software Development Series, Addison- Wisley, USA, Boston 2006.

Logue, K., McDaid K. (2008), "Agile Release Planning: Dealing with Uncertainty in Development Time and Business Value", 15th International Conference on Engineering of Computer-Based Systems (ECBS), 437-442.

Mahnic V., "A Case Study on Agile Estimating and Planning using Scrum", Electronics and Electrical Engineering, Kaunas Technologija, 5(111): 123-128.

Moløkken-Østvold, K. M., Haugen N. C., Benestad, H. C. (2008), "Using Planning Poker for Combining Expert Estimates in Software Projects" Journal of Systems and Software, 81(12): 2106-2117.

Momoh, J., Ruhe, G. (2006), "Release Planning Process, Improvement – An Industrial Case Study", Software Process and Improvement Practice, 11(3): 223-249.

Ruhe, G. (2005), "Software release planning", Handbook in Software Engineering and Knowledge Engineering, World Scientific 2005, 3(1): 1-21.

Saliu, O., Ruhe G. (2005), "Supporting Software Release Planning Decisions for Evolving Systems", Proceedings of the Software Engineering Workshop, 25th Annual IEEE/NASA, Washington 2005.

Stober, T., Hansmann, U. (2009), "Agile Software Development – Best Practices for Large Software Development Projects", Springer Publishing, New York 2009.

Tolstukhin, A. (2009), "Values and Success Factors of Agile Project Management - Assessing agile methodologies and building a tool for agile project management software evaluation", Bachelor's Thesis, Aalto University School of Economics, Helsinki 2009.

Vähäniitty, J., Rautiainen, K., Heikkilä, V. & Vlaanderen, K. (2010), Towards Agile Product and Portfolio Management, Aalto University School of Science and Technology, Software Business and Engineering Laboratory (SoberIT), Helsinki 2010.

# Appendices

## Appendix 1. SWOT Analysis

| ***Strengths*** | ***Weaknesses*** |
|---|---|
| S1. Flexible and adaptive to changing requirements<br>S2. Higher stakeholder and user satisfaction<br>S3. Higher value and quality of the product<br>S4. Requirements prioritization<br>S5. Effective planning<br>S6. Project progress visibility<br>S7. Process and design simplicity | W1. Lack of documentation<br>W2. Threat of inefficient communication methods<br>W3. Limitations in globally distributed development teams<br>W4. Heavy reliance on the development team<br>W5. Low level of planning and design<br>W6. Limitations in large and complex projects<br>W7. Reliability of testing |
| ***Opportunities*** | ***Threats*** |
| O1. Facilitating inter-team and intra-team communication<br>O2. Utilizing technology and tools in distributed development teams<br>O3. Looking for new opportunities<br>O4. Improving planning and forecasting accuracy<br>O5. Expert advice<br>Q6. Standardizing testing methods | T1. Lack of interest in agile methodologies<br>T2. Lack of will for improvement strategies |

## Appendix 2: Questionnaire

*The data was modified due to confidentiality reasons**


### Questionnaire

*The purpose of the following questionnaire is to investigate teams' practices related to effort estimation, velocity and other processes. The results will be collected anonymously.*

***Please answer the questionnaire once and complete all questions.***

*This questionnaire is part of the Master's Thesis on the topic of release planning optimization. The results will be extremely important for understanding the challenges faced by a multi-team agile development environment, as well as identifying possible areas of improvement and setting common basis across teams. Additionally, the results will help establish the parameters for the optimized release planning tool.*

*The following questionnaire is divided into 5 sections: Introduction, General practices, Effort estimation (User story and task levels), Cross-team collaboration and Velocity questions. Effort estimation section is split into two parts: user story effort estimation and sprint planning effort estimation. Questions from the introduction will serve to group the results for analysis.*

***Each section includes fill-in** questions, multiple choice questions, Likert opinion scale questions as well as comments questions. Likert opinion scale questions have a 5-**level rating where two ends of the scale are opposite positions of opinion,** "5 = Strongly Agree" **and** "1= Strongly Disagree". In comments question you can write any information you consider useful for the study of the respective section.*

*Introduction*

**1. Where is your team located? ***

**2. Which category does your team belong to? ***

○ Feature ○ Network Verification ○ Performance Testing

**3. How many team members are there in your team? ***

○ below 6 ○ 6 – 9 ○ more than nine

**4. To what extent is your team experienced with agile methods? ***

○ under one year ○ 1-2 years ○ more than 2 years

*General processes*

**5. Which methods are used for managing tasks on sprint level? ***

☐ Post-it notes ☐ White board ☐ Spreadsheets ☐ Agile project management software ☐ Other

**6. Are all user stories which are not completed within one sprint continued in the next sprint? ***

○ Always ○ Sometimes ○ Never

**7. Does your team do multiple tasks simultaneously? ***

○ Always ○ Sometimes ○ Never

| **8. Please, answer the following questions about some general pratices. *** | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| Are all requirements always defined clearly and extensively? | ○ | ○ | ○ | ○ | ○ |
| Are all user stories prioritized sufficiently according to their value in a given release? | ○ | ○ | ○ | ○ | ○ |
| Are user story interdependencies always properly taken into account? | ○ | ○ | ○ | ○ | ○ |
| Are new user story characteristics very much similar to the user stories completed earlier? | ○ | ○ | ○ | ○ | ○ |
| Are all team members within your team fully multifunctional? (i.e. all team members within your team are able to perform all tasks required by the team) | ○ | ○ | ○ | ○ | ○ |
| Is the team progress fully visible during the sprint? | ○ | ○ | ○ | ○ | ○ |
| Is communication fully sufficient within the team? | ○ | ○ | ○ | ○ | ○ |
| Is there a lot of bug fixing or other unexpected processes during the sprints? | ○ | ○ | ○ | ○ | ○ |

*Effort estimation*

**10. How is the size/effort of user stories done in your group? ***

☐ Individually ☐ Team discussion ☐ Planning poker ☐ Other

**11. What metrics are used for estimating effort of a user story? ***

○ Refrence user stories with effort range to compare ○ Approximate hours ○ Other, what

| **12. Please answer the following questions regarding user story effort estimation: *** | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| Does your team have clear a basis/guideline for effort estimation of user stories? | ○ | ○ | ○ | ○ | ○ |
| On average, is estimating the exact size of a user story complex? | ○ | ○ | ○ | ○ | ○ |
| Is there always a mutual agreement among all team members members about the size of a user story? | ○ | ○ | ○ | ○ | ○ |
| Is effort estimation based on the estimates from the past user stories? | ○ | ○ | ○ | ○ | ○ |
| Is effort estimation done based on general experience of the team and user story characteristics analysis? | ○ | ○ | ○ | ○ | ○ |
| Does your team spend enough time on estimating the size of user stories? | ○ | ○ | ○ | ○ | ○ |
| Does your team estimate user story sizes successfully? | ○ | ○ | ○ | ○ | ○ |

**14. Other feedback regarding the user story effort estimation:**

*Cross-team collaboration*

| **20. Please answer the following questions regarding the activities related to the other teams: *** | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| Is there a well-defined basis/guideline for effort estimation with other teams? | ○ | ○ | ○ | ○ | ○ |
| Does your team regularly discuss the basis/guideline for effort estimation with other teams? | ○ | ○ | ○ | ○ | ○ |
| Are tasks assigned to your team highly interdependent with other teams? | ○ | ○ | ○ | ○ | ○ |
| Does your team always carry out the whole user story on its own? | ○ | ○ | ○ | ○ | ○ |
| Does development process flows smoothly? (you don't have to wait for other teams' work to start on a task) | ○ | ○ | ○ | ○ | ○ |

*Velocity*

**22. Does your team always know its current velocity? ***

○ Yes ○ No

**24. Does your team have any ongoing activities or plans to improve velocity? ***

○ Yes ○ No

| **23. Please answer the following opinion questions regarding velocity: *** | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| Does your team continuously track its progress during the sprint using its velocity? | ○ | ○ | ○ | ○ | ○ |
| Is velocity increasing continuously over sprints? | ○ | ○ | ○ | ○ | ○ |
| Is velocity consistent or fluctuating? | ○ | ○ | ○ | ○ | ○ |

**25. Other comments related to the team velocity:**

**Thank you!** Submit