From agility to productivity

An analysis of agile method fragments

Master thesis

Author
Erik ten Brinke
Master Business Informatics
Utrecht University

Supervisors
Dr. Slinger Jansen (Utrecht University)
Drs. Kevin Vlaanderen (Utrecht University)
Rob van Vliet (Centric B.V.)
Hans Bredewoud (Centric B.V.)
Herwin Kwekkeboom (Centric B.V.)

Version: 1.0
Date: August 07, 2013

Universiteit Utrecht

CENTRIC

Thesis title	From agility to productivity: an analysis of agile method fragments
Thesis number	
Author	Erik ten Brinke Master Business Informatics Universiteit Utrecht e.c.tenbrinke@students.uu.nl / tenbrinke.e@gmail.com
Student ID	3786234
First supervisor	Dr. Slinger Jansen Department of Information and Computing Sciences Universiteit Utrecht
Second supervisor	Drs. Kevin Vlaanderen Department of Information and Computing Sciences Universiteit Utrecht
External supervisors	Rob van Vliet, Hans Bredewoud, and Herwin Kwekkeboom Centric B.V.

Abstract

Organisations want to influence the productivity of their development teams. There are a variety of methods available for teams to use for its development purposes, this study focuses on the agile development methodology. Currently there is no knowledge to which extent elements of agile development influence productivity. This research project identified influences on productivity by means of analysing process changes within development teams (i.e. addition, change, or deletion of a method element).

Productivity is the main subject for this research project and is operationalizes and scoped by a triangular model describing three types of metrics: technical (LOC), functionality (user stories, tasks, bug fixes), and quality (post release defects). The aforementioned three metrics were measured at each method fragment which was analysed.

Twenty five method fragments were identified and found eligible for productivity analyses. Ten development teams of the case company were interviewed in order to identify process changes during its agile usage. Twenty eight process changes were identified at these teams. A selection of teams was necessary since the productivity analysis phase and quantitative backlog of the teams was substantial in terms of size. This selection was established by means of five criteria assessments.

A detailed quantitative analysis method was used, of which for instance the complexity of the functionality measurements was a challenge. Thirteen method fragments in total were analysed on its influence on productivity. From these thirteen method fragment, eight were found to have an effect on productivity. Five method fragments had an effect specifically on productivity namely the scrum board, the product backlog, the burn down chart, the demo, and sprints. Three method fragments had an effect specifically on quality, being pair programming, the sprint backlog, and self-contained teams. The effects on productivity were of varying strengths. An interpretation of the results is added to support organisations into understanding exactly why certain method fragments did have an effect while other method fragments did not.

Teams experienced process changes in varying ways. Most teams indicated that introduced method fragments needed time and refinement to be integrated in an efficient manner within the development process. The results and corresponding effects are validated by means of expert sessions. This research made a contribution to the field of productivity, method engineering, and agile development by identifying which method fragments of agile development methods influenced productivity and to which degree it had an effect. It is the first research project which measured productivity on a method fragment level.

Keywords: *Productivity, Development process changes, Method fragments, Agile development, Scrum*

Acknowledgements

This research project and corresponding thesis would not have been possible without the support and supervision of several people. Therefore I would like to thank certain people who contributed to this research and thesis.

First of all I would like to thank my company supervisors Hans Bredewoud and Herwin Kwekkeboom for their support, feedback and total time spent on both the research project and the thesis. An additional note to be made is that this research project would not have been possible without its initiator Rob van Vliet. Slight adjustments were made prior to starting the actual research, the aforementioned people were flexible in adjusting the goal of the research.

Next I would like to thank all the teams that were interviewed for undertaking the effort and allocating time for the interviews that were conducted. They were not only supportive in conducting an interview but they were also available and happy to answer questions when these arose after the interviews. I would also like to thank the organisation of Centric in general for their support in terms of resources.

Next to the people of Centric, I would like to thank my supervisors from Utrecht University namely Dr. Slinger Jansen and Drs. Kevin Vlaanderen for their support and feedback on various research deliverables as well as the eventual thesis.

Table of content

List of tables

List of figures

1. Introduction

1.1 Problem statement

The business environment nowadays requires a lot from organisations in order to have a chance to be successful. Increasing globalization, changing customers' demands, top quality, fast delivery of software products and updates are aspects which organisations experience and must conform to in order to align with the requirements of its customers.

Productivity of developers and software product managers is of great value to organisations and is a factor in achieving an organisation' goals and complying with the requirements of its customers. Improved productivity can enable organisations to shorten the release cycle, increase the amount of requirements per release, and improve overall product quality. Organisations typically opt for a situational development method since they tend to tune a method to their own organisation rather than implementing a complete method that is based on theory. Organisations could for instance implement an agile method like Scrum or XP. But how will an organisation obtain the knowledge to decide which specific parts of an agile development method are beneficial for its software development processes? And more specific for this research: if an organisation would like to increase productivity, which specific parts of an agile development method should it implement?

Process changes are a major aspect when it comes to introducing a development method like agile or certain parts of it. These changes in the overall process affect variables of a software development environment, like for instance productivity. Process changes can also be considered as an insertion, change or deletion of so called method fragments because this research focuses on certain fragments of agile software development methods. Method fragments are defined as coherent pieces of information system development methods (Brinkkemper, 1996).

By performing a retrospective study as part of the overall thesis research, it will describe which changes in agile development processes influence productivity. Other factors also have influence on whether a software development process change is beneficial for an organisation or not. These factors relate to for instance organisational fit (will a certain change actually work in the organisation?) and the acceptance level of the software development department.

In relation to process changes, productivity refers to practical matters like the amount of bugs solved, shortened sprints, increased number of releases per year, increased number of requirements per release, and increased overall performance of the product. Additionally, implementing method fragments of agile development causes concern in terms of culture and organisational structure due to its nature being different compared to other development methods like the waterfall method. This research will also give insight into how these method fragments were integrated within the organisation, whether this was successful or if certain pitfalls can be identified.

The formal problem statement for this thesis project is formulated as:

"Which method fragments of agile software development have an influence on productivity"

1.2 Objective

The objective for this thesis project is to establish which agile development method fragments influence productivity. The second main objective is to give organisations insight into how these fragments were integrated into the organisation and (existing) software development method(s).

By achieving the first objective, organisations are given insight into method fragments that influence the overall productivity of agile development methods. By following the provided integration description one could successfully integrate the proposed method fragments into the development environment.

It can be interpreted as best practices and lessons learned aspect of this thesis project. This will enable organisations to use the proposed productivity improving method fragments efficiently and deliver quality software to its customers.

The formal objective for this thesis project is stated as:

> *"Provide insight into method fragments that influence productivity"*

1.3 Research questions

A thesis project requires a central research question, which in this case was set up with the support of Centric and both supervisors of the Utrecht University. The research question of this thesis project is:

> *"To which extent do process changes in agile software product development influence productivity?"*

Next to the central research question, there are six additional sub questions to support answering the central research question:

1) How is productivity defined and how can it be measured?
2) What method fragments can be identified in agile software development methods?
3) Which method fragments are eligible in terms of influencing productivity?
4) What effect do these method fragments have in terms of quantitative data?
5) What is the experience of the software development department regarding these changes in the development process?
6) In what manner are method fragments that positively influence productivity integrated and optimized within an organisation?

1.4 Scope

Process changes and process analysis are very broad terms and therefore needed to be scoped properly prior to starting this research in order to get a clear goal and overall reach of the thesis project. The scope of this research project consists of all agile software development method fragments, which could be of various agile software development methods like Scrum, XP, DSDM, and FDD. However, this research could not support and describe an unlimited amount of method fragments. For this research we aimed to include 10 to 15 method fragments which would be extensively analysed, more details concerning this analysis can be found in the next chapter. This amount is based on a combination in-depth analysis of every method fragment and the overall thesis length of six to eight months. Further scope decisions, such as productivity metrics, the selection of teams and the length of quantitative analysis per method fragment, will be depicted at its dedicated chapters.

1.5 Scientific relevance

Agile development is a popular and successful software development methodology. It has grown substantially in terms of usage over the last decade. Agile, however, is difficult to implement because its nature is so much different compared to traditional development. The social aspect and culture are aspects that are considered difficult to implement from an organisational perspective while the sprints, continuous integration and extensive testing are challenges from a managerial and technical perspective.

Implementing agile processes can be a large hurdle for organisations, especially when they are used to working with a traditional development methodology like the waterfall method. This is why it is important to indicate to organisations which changes in the development process are beneficial compared to their existing software development processes. The aforementioned example relates to agile and waterfall development but it could also include other development methods like RUP. The beneficial changes with regard to an organisations development process is for this specific research project concerned with productivity improvement.

Research has been done regarding the productivity of developers (see the theoretical background section for more details) but not much has been done focusing on software development process changes influencing the overall productivity. This research will give insight into just that. Additionally, there has been no research performed on method fragment level of agile development methods concerning its influence on productivity apart from the pair programming method. Additionally, insight is given into how certain method fragments that positively influence productivity are integrated into the development process.

1.6 Main deliverables

The thesis project resulted in various deliverables along its lifeline. These deliverables brought structure and gave clarity into what aspects needed to be completed in order to let the thesis project have a chance of success. Additionally, it provided input for setting up a planning of the overall thesis period.

Deliverable type	Description
Long proposal	The long proposal is the document which proposes a thesis project. It was judged by the supervisors of Utrecht University. When the proposal was accepted, the actual thesis project could start.
Planning	Every project needs a proper planning in order to stay within time limits and to show when certain aspects of a project will be finished. For this thesis project it was crucial to determine a sufficient planning to stay within the thesis period length of 6 to 8 months and to give an indication of how much time was needed to finish the various deliverables.
Productivity definition	A concrete, well stated, and scoped definition of the term productivity is described in chapter 3.1.
Identification of agile method fragments	An overview of available method fragments concerning agile software development methods. This overview is given in chapter 4 and lists the identified method fragments of agile methods like Scrum and XP.
Selection of eligible productivity influencing agile method fragments	A selection of identified method fragments that were found to be eligible for influencing productivity. Method fragments are eligible when certain parameters, which are described in chapter 5, are met.
Selected method fragment analysis	Analysis of selected method fragments. Detailed specification of analysis content is depicted in chapter 2.5.
Interview plan	An interview plan was set up to gather information about the overall thesis subject and also specifically about sub questions and the central research question. By talking to experts, the quality of the research was improved. The interview plan is described in chapter 6.1.2.
Quantitative results of selected method fragments	Quantitative analysis and results of the selected method fragments. Quantitative metrics that will be gathered are LOC, the amount of developed functionality and the amount of post release defects. Further details can be found in chapter 6.2.
Software development department experience	Description of the software development department' experience concerning changes in the software development process is described in chapter 7. This experience can be either positive or negative. The overall experience of changes in the software development process is of importance since method fragments that have a positive influence on productivity does not necessarily have to lead to a positive work experience. It also relates to the overall acceptance level of a software development department.
Method fragment integration description	Description of in what manner the selected method fragments were integrated within the organisation is depicted in chapter 8. Integration could for instance be done at once or in phases. Others factors include training of personnel.
Validation	Research does not mean a lot when it is not validated in practice. Validation of this thesis project will be in the form of expert opinions (interviews). The data analysis from the various selected teams was substantial in terms of size, therefore a separate pilot validation was not included.

	Validation concerning the quantitative data is in the form of analysing the same method fragments of different teams. A full description concerning the validity of this research can be found in chapter 9.
Answer central research question	The eventual goal of the thesis project was to answer the central research question. By providing the several sub questions with answers, input was generated for eventually answering the question to which extent process changes in software product development influence productivity. Answering the central research question can also be considered the 'end-deliverable' of this research project. The central research question is answered in chapter 10.
Thesis document	The end-deliverable of the overall thesis period and specifically for the Utrecht University. It will describe the thesis project and its results in detail. The document will be graded by the two supervisors.

Table 1: Thesis' main deliverables

1.7 Case company

The research project has been conducted at the case company Centric. Centric is a Dutch IT organisation which provides software and infrastructure for various markets such as the public sector, finance, healthcare, education, construction, retail and logistics. Every department has several development teams working on software products for these markets. Centric acquired many companies over the years and has therefore grown substantially in terms of size, 5.300 employees at the time of writing. The organisation of Centric is apart from The Netherlands also active in ten other EU countries such as Belgium, France, Germany, Switzerland, United Kingdom, Romania, Norway and Sweden.

2. Research approach

Every research needs a proper and sound approach to be able to perform a successful research project. In this section of the thesis, the research methods which were used during the thesis project will be discussed in detail.

2.1 Literature study

The first research method that was used is the literature study. A clear search and selection strategy is one of the most important aspects of a literature study (Carnwell and Daly, 2001). The actual study was performed on the topics of general development methodologies, software development process changes, productivity, and agile method fragments. The theory on these topics was gathered from journals, papers, and conference proceedings.

The literature study will have six main goals:

- Gather knowledge on the thesis subject;
  As a start to the thesis project, knowledge about the topic of general software development methodologies and corresponding process changes was gathered. This was beneficial for myself as I gathered basic knowledge about the subject as well as an introduction to the actual research project.

- Describes the context of the thesis project;
  The literature study also gave the context of the thesis project, an introduction and description of the domain where software development processes are active.

- What work has already been performed in this research field;
  One of the core aspects of a literature study is to be able to show what work has already been done on a particular research topic. For this thesis project it gave an insight into what research has been performed into software development methods and in what manner it can change the development process.

- Identifying gaps and opposing views;
  Another functionality of the literature study is to identify gaps within the available literature and knowledge. This can for instance be used to be innovative with a research project or to bridge a gap between conducted research. Identification of opposing views from various research projects is also interesting since this could indicate a contradiction in findings and results on the same subject. An example of opposing views in literature are the various ways in which productivity is measured, this will described in detail in the theoretical background chapter.

- It was the base for answering sub question one, two, and three;
  Sub question one and two were answered based on findings from the literature study. Concerning sub question one; productivity is a term which was hard to define in theory. Defining the term in detail of what it entails was crucial to the rest of the research, also to establish a scope for the term productivity. Since sub questions one and two are theory based, answering them using a literature study was appropriate. Sub question three partly used literature in order to identify which method were eligible in terms of influencing productivity.

Additionally to the points mentioned above, it was also a base for contact information concerning expert interview and (expert) validation of the research results. The actual method that was used for the literature study is the traditional literature study. Cronin et al. (2008) describe that the main purpose of the traditional literature method is to provide the reader with a comprehensive background in order to understand the past and current state of a (research) domain and also to indicate the significance of new research. Additionally, a sound research question is crucial prior to starting the actual traditional literature study (Beecroft et al., 2006). From an analysis of existing literature study methods, and from the sources stated above, one can conclude that a traditional literature study method was the most appropriate for this thesis project. No systematic literature review was performed because a systematic literature review is mostly a research on itself (since it is so detailed and complex) and therefore aimed towards only indicating and explaining in fine detail what kind of research has been done in a particular domain instead of also performing new and innovative research. Since this thesis project was also aiming to be an addition to the scientific domain, the traditional literature study method was selected. The snowballing method was used to support the traditional literature study, this method is about using references of a paper to find new papers on the same subject.

2.2 Interviews

Interviews were a part of this thesis project and were designed according to an interview protocol which is described in chapter 6.1.1 and 6.1.2. Interviews were conducted with employees/managers of Centric to analyse the various used development methods and in what manner these methods were used. This was used to assess the current situation of development because departments of Centric use different kinds of development methods. Based on this overview of used development methods one was able to see which departments and teams used an agile methodology. Interviews were used for sub question 2, 3, 5, and 6. For sub question 2 and 3 questions were asked like what kind of process changes have occurred over a particular time period. Interviews were also used for sub question five to indicate how developers experienced the software process changes over time, how they feel about introducing new elements to their work, whether they feel like the productivity has improved from their point of view, and if they like their new way/style of working. Conducting interviews was also beneficial for sub question six because it led to (detailed) descriptions in what manner new elements were integrated within the software development process.

The data from the various interviews resulted in an identification of what kind of process changes have occurred over a particular time period, how a developer's working day has changed over time, and how these process changes were integrated. Secondly, the data from the interviews were used to answer the designated sub questions and ultimately supported answering the central research question. Additionally interviews were held with (industry) experts in the form of scrum coaches. Their knowledge about the domain, agile development methods and trends was of great value to the thesis project and especially regarding validation aspects. The conducted interviews were semi-structured and resulted in qualitative data for thesis project purposes.

2.3 Quantitative analysis

The process change analysis was also performed by means of metrics. This was to actually measure the process change, its variability in numbers and to indicate if it has improved the overall software development process or not. Determining and measuring the quantitative productivity factors is difficult due to for instance software development methods being very different from each other (agile and waterfall for example) and productivity within these development methods is hard to put in numbers. A detailed quantitative analysis method is described in chapter 6.1.6. Especially sub question three, four, and the central research question was based on (and answered by) quantitative analyses of the process changes and its corresponding method fragments.

2.4 Document study

Studying documents of Centric was done in order to obtain knowledge of the various software development processes within the organisation. This also gave insight into the changes that occurred over a particular period of time. It can be considered a backlog of everything relating to the software development processes within the organisation. Specific examples of documents which were used are the release notes and retrospective documentation. These documents were especially beneficial for sub question 2, 3 and 4.

2.5 Software development process analysis

Based on the research approach one can set up a process analysis model.

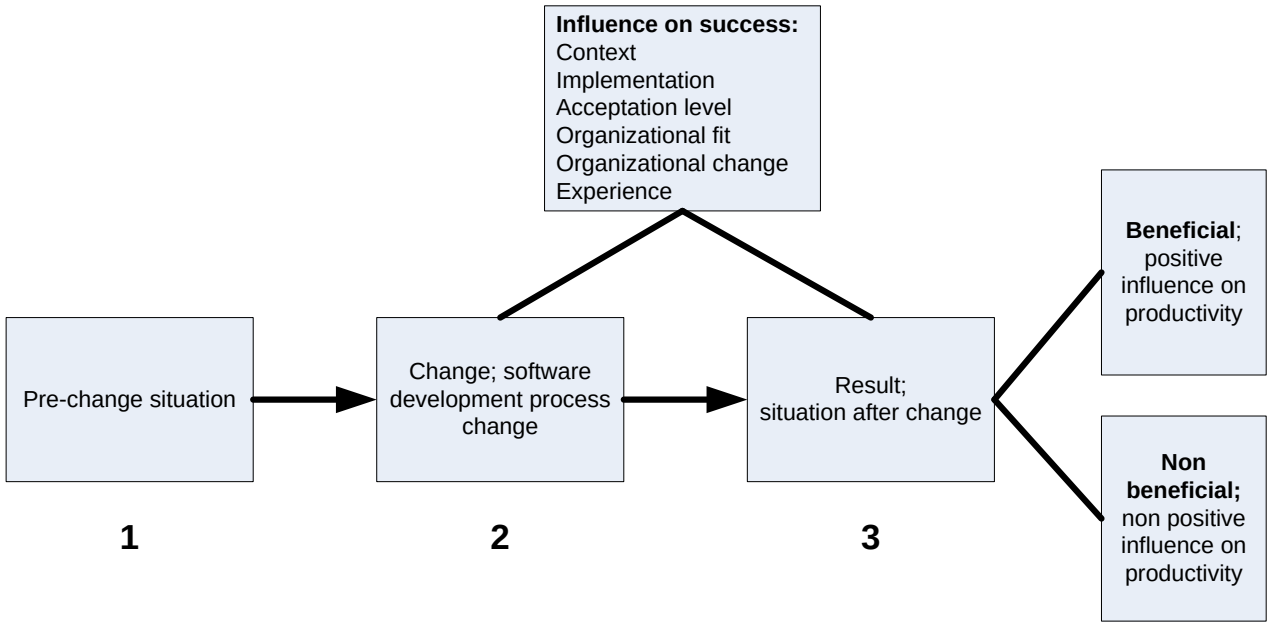

Figure 1: Three phase process analysis model

Figure 1 emphasizes three phases: a pre-change (as-is) situation, a change, and the result of that change. This change will result in either a positive or a non-positive influence on the overall productivity. The model can also serve thesis project validation purposes.

Phase 1 is about describing how the development process was prior to a certain process change. This can also be seen as a so called ‘base-line’ to compare the process changes with. Phase 2 describes the process change in detail, what kind of method fragment has been introduced, changed, or removed from the overall process.

Aspects of the process changes that will be described in detail are for instance:

- The introduced, changed or removed method fragment;
- The context of the process (overall software development process or a particular department?);
- Has the method fragment been adjusted prior to implementing?;
- The amount of time it took to implement;
- In what manner has it been implemented (i.e. at once or in phases);
- Has the method fragment changed over time?;
- Has it lead to organisational change? (i.e. training of personnel).

A timeline for instance can help indicate whether a method fragment was adjusted over a period of time, the amount of time it took to implement, and in what manner it has been implemented.

The method fragments and the software development process were modelled using the PDD technique, which stands for Process Deliverable Diagram. This was used for both analyses as well as visualization purposes of the aforementioned method fragments and overall software development process. The PDD technique was developed by van de Weerd et al. (2008). The left side of a PDD is the process side, which is modelled using UML activity diagrams. The right side of a PDD is the deliverable side, which is modelled using UML class diagrams. PDD's have proven to be effective for the meta modelling of methods, especially for analysis and design stages. Phase 3 is about the situation after the software development process has been changed. Has changing the process been beneficial in terms of productivity or not? There are factors that influence the change of success like the context, way of implementing, acceptation level, organisational fit, organisational change, and overall experience of the software development department. These influence factors on success are also depicted in the conceptual model of figure 1. One can state whether a change in the software development process has had a positive or non-positive influence on productivity based on the definition and scope of productivity mentioned in sub question one. Phase 1 and 2 used interviews and document study as a basis, phase 3 additionally used quantitative data.

The research methods which were mentioned and addressed in this chapter supported answering the various sub questions of this research. Figure 2 depicts an overview of which sub question used which research method.

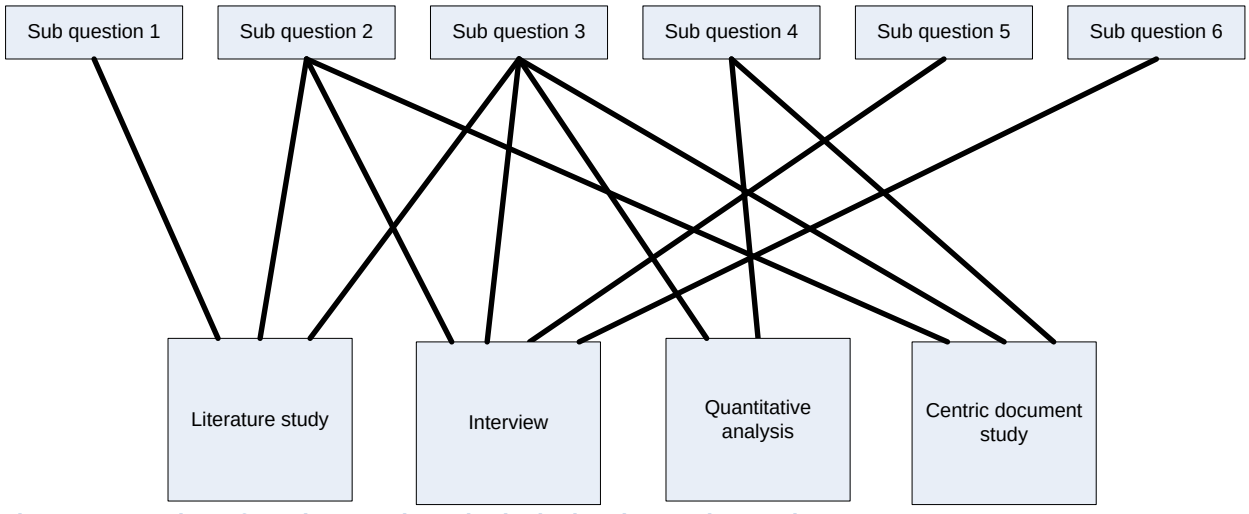

Figure 2: Overview of used research methods depicted per sub question

3. Theoretical background

This chapter will describe the available literature background on the topics of interest concerning this research project.

3.1 Agile development

Agile development allows an organisation to release software frequent and handle changing customer requirements. This development method follows specific principles which are set up in the Agile Manifesto (2001). The principles are:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation;
- Responding to change over following a plan.

There are several methods within the agile methodology such as scrum, XP (extreme programming), FDD (feature driven development), and DSDM (dynamic system development method). A survey of 500 participants in 70 countries concluded that 40% used scrum as its agile method, compared to 7% using XP, 4% using FDD and 1% using DSDM (Scrum alliance, 2013). One can conclude that scrum is used significantly more than other agile methods.

A study by the Standish group (2012) showed that agile projects have a success rate of 42% compared to a 14% success rate of the waterfall method. This shows that agile projects are approximately three times more successful compared to projects which use the waterfall method. The results are based on software development projects ranging from 2002 to 2010. The study regarded a project successful when the project was delivered on time, on budget, and completed with all planned features.

One of the aspects which makes agile development stand out is the iterative nature which are called sprints. Sprints are development cycles ranging from two to four weeks, depending of a team's preference. After every sprint a team is able to releases working software to its customers. Other (unique) aspects of agile development are described in detail in chapter 4. A general overview of the scrum process is depicted in figure 3.

3.2 Software process improvement

Studies have been conducted on the topic of software process improvement and corresponding process changes. Process changes in literature relate to for instance inserting certain method fragments or combining development methods. A process change in software development could also be that work is divided over several parallel working teams. A study by Perry et al. (2001) states that there is a significant correlation between the amount of parallel working being done on a particular software component and the number of defects and problems in terms of software quality. This indicates that dividing work over several departments can influence overall software quality.

The Capability Maturity Matrix (CMM) is another method to analyse software process maturity. Paulk et al. (1993) developed the initial model, it describes five levels ranging from basic to advanced: initial, repeatable, defined, managed, and optimizing. Every level holds various activities which have to be integrated within the software development process to achieve a particular level. An example of an activity on the repeatable level is requirements management, on the defined level a training program is an example.

Results from a study of Jiang et al. (2004) showed that software process management maturity is positively associated with project performance. Additionally, they concluded that the 'process engineering and organisational support activities' was the most important level of the CMM in a way that it is positively associated with control, interaction quality, and software flexibility. This means that the engineering and management of the software development process is highly important in relation to the eventual end-product.

SPI is a term which is used in many literature resources. It stands for Software Process Improvement and is therefore applicable to this research. Shih and Huang (2010) did research into SPI on three aspects: overall support for SPI within the organisation, impact on the quality of the software product, and the degree to which SPI is used. Other studies also focus on the organisational aspect, like studies of Stelzer and Mellis (1998) and Iversen and Ngwenyama (2006). This research is focusing on another aspect, namely that of SPI in relation to productivity. Hansen et al. (2004) performed a literature study on 322 contributions to the SPI domain. Their findings included that the field is dominated by one approach, the CMM. Other findings were that most studies are biased towards a prescriptive way rather than descriptive or reflective. By prescriptive is meant that it indicates to SPI professionals what to do. Hansen et al. conclude that these two trends are not necessarily beneficial to the field. One of their suggestions is that more statistical analysis should be done, which is part of this thesis project.

The theory of agile development prescribes that productivity should be enhanced. A study of Ilieva et al. (2004) compared the productivity of two similar projects, one of which used traditional methods while the other used XP. They measured the productivity for three iterations of each project. Overall, the results showed a 42% increase in productivity for the agile team. The increase in productivity was the largest in the first iteration, while there was hardly any difference in productivity concerning the third iteration.

The case study by Layman et al. (2004) compared an old release developed with traditional methods with a new release developed with agile methods. The results showed a 46% increase in productivity for the new agile release compared with the traditional release. However, the agile team had notably greater domain, programming expertise, and project manager experience. This was due to the fact that three of the team members on the new release had previously worked on the old release. Other studies did not find such results, like the study of Wellington et al. (2005) found a 44% decrease in productivity for an XP team compared with a traditional team. Furthermore, Svensson and Höst (2005) only found evidence that when the agile process was introduced, the team improved their productivity during the first iterations.

Some studies have focused on measuring software development productivity in general. Maxwell et al. (1996) used the lines of code (SLOC) as their main metric and concluded that the use of tools and modern programming practices were major factors in productivity improvement. Mahmood et al. (1996) focused on the several stages of software development such as design, code changing, and quality checks and used the hours spend on these stages as metrics. Banker et al. (1991) used the same metric for their productivity approach. This is an indication that no standard has been set on measuring productivity.

Since method fragments belong to the topic of method engineering, the literature section also focuses on this broader topic. With regard to the conceptual model in figure 1, Nejmeh et al. (2006) state that the context of a company is the largest influence on software process change cycles. Research by van de Weerd et al. (2010) on process improvement and method engineering showed that a shared infrastructure is critical for rollout. They found that company-wide availability of tools with regard to supporting a method is a critical success factor for proper adoption of new method fragments. Another interesting finding was that especially small method fragments facilitate gradual process improvement, because most organisations struggle to implement large method fragments. Rossi et al. (2004) did a study on method rationale. Method rationale is also applicable to this research since method rationale relates to the history, changes, and decisions being made to a certain method. Rossi et al. found that method rationale is a powerful mechanism in maintaining knowledge on systems development. Benefits include that it can support future decision making on the method, familiarize (future users) with a certain method, and give insight into the limitations of the current method.

3.3 Defining productivity

As mentioned in the main part of the theoretical background, literature describes several methods to measure productivity in a software environment. Cocomo is such a method and is a software cost estimation model developed by Boehm (2000). It computes the amount of people that are required for a particular development project. The calculation is based on the effort expressed in an estimated amount of lines of code and the estimated development time that is needed for that particular project. Cocomo is a well-known method but not very well suited for this research project since it focuses on the costs of a software project rather than what elements influence productivity. It is also too high level for this research since we want to zoom in on particular method fragments and not the overall development method.

Measuring function points is another way of calculating productivity. Function points are functionalities of a given software product and the method was developed by Albrecht (1979). It captures functionality aspects like search forms and unique reports. This could be an efficient method if this research was focused on overall development methods and not specific method fragments. Identifying function points of a software product costs considerable amount of time and can be too high level compared to specific method fragments. Additionally, Albrecht and Gaffney (1983) concluded that function points are highly correlated with lines of code. However, a function point analysis is thought to be more useful than LOC (lines of code) when considering work effort estimation. This is due to function points being relatively easy to estimate from basic requirements in terms of work effort prior to starting the actual development. Since this research does not focus on estimating work effort, this advantage is neglected.

Productivity concerning software development is also described in scientific papers but it is defined in varying ways. Yet, commodities across these studies can be spotted. A study by Ilieva et al. (2004) measures productivity in KLOC (lines of code in thousands) and manhours per month. This study also describes three possible levels of detail, namely team, pair, and individual. Wood and Kleb (2003) and Moser et al. (2008) use the same metrics for their study. Layman et al. (2004) uses the same kind of metrics but also addresses the amount of user stories per user per month. Unfortunately during their study this metric was unavailable since the team that was under study did not utilize user stories during the comparison release. Layman stresses the fact that this metric would help in gauging the amount of actual functionality, rather than only focusing on the amount of code the developers produce. Williams et al. (2004) uses the KLOC as well as the user stories metric because they state that neither metric is perfect on its own. They mention that LOC is a precise metric, but customers pay for features, not for LOC. Moreover, LOC could also be reduced due to code reuse and refactoring. Williams describes a benefit of the user stories metric as that it creates no extra work for the team. A combination of both metrics is beneficial. A third metric they used for their study is the Putnam productivity parameter, this is a parameter value which results from an equation. This equation is based on production data from large software projects and is based on KLOC, effort (in years of work done), time (elapsed years of the project), and a skill factor (Putnam, 1978). Williams sees this parameter as an extra supporting factor concerning increased or decreased productivity. A downside of this parameter is that the theory is rather old, namely 1978. A study which aligns with that of Layman et al. (2004) and Williams et al. (2004) is that of Abrahamsson (2003). This study also uses LOC and developed user stories, but measures these against the post release defects (bugs). Next to the overall productivity, it also gives an indication whether the productivity has led to improved software quality. This is especially interesting because if improved productivity leads to a decrease in software quality, this improved productivity is not beneficial for an organisation.
Maurer and Martel (2002) mention that productivity is defined in textbooks as the size divided by the effort. Their study includes LOC, but also uses the number of methods and classes which in their case is specific for java. They also use the amount of solved bugs and features added in a particular release, thus it aligns with Abrahamsson (2003) in terms of quality. Nierstrasz (2004) also acknowledges this and states that productivity is measured by functionality and quality compared to the overall effort. According to Nierstrasz, functionality and quality can also be seen as software requirements.

One can see commodities between various literature resources in the form of LOC as a technical metric and user stories as a functional metric. Additionally, if these two types of metrics were to be compared against the amount of defects (bugs) it can give a proper indication of the productivity as well as an

indication whether this degree of productivity has had an influence on the software product quality. The LOC and user stories will be measured per release, this calculation is also used in the studies of Ilieva et al. (2004), Wood and Kleb (2003), Moser et al. (2008), Layman (2004), Williams (2004), Abrahamsson (2003), Maurer and Martel (2002), and Nierstrasz (2004). Should sprint length differ over time, a solution was set up prior to the actual analysis to measure the metrics per month instead of per release. In terms of user stories, the developed functionality should be of the same size and complexity to be able to make comparisons to previous releases.

The type of defects will be post-release defects because it can state something about the software quality delivered to the customer. Pre-release defects would merely state something about the software quality delivered to for instance the test department. Studies that use bugs in their calculation, like the studies of Abrahamsson (2003), Maurer and Martel (2002), and Nierstrasz (2004) use the amount of bugs rather than repeating bugs or bug solving time. This is due to the fact that the amount of post release defects can be measured against LOC and the functionality quantity. Metrics like repeating bugs, the delay in development due to bugs or bug solving time are not applicable nor feasible for this research project since the outcome of these parameters would not fit in terms of comparison to the other used metric types. It is important to scope other definitions as well, similar to defining productivity previously in this paragraph. Another term which needs to be defined is a (software) bug. A bug can be interpreted in many ways, which is why the following definition is used: A bug is an error in coding that leads to malfunctioning of a software application or it produces incorrect/unexpected results. By stating it this way customer requests and wishes (RFC) are excluded from this research project.

Figure 3 displays the triangular model which was used for productivity measuring purposes. A technical element in the form of LOC, and a functional element in the form of developed user stories, tasks, and bug fixes. The third element is how these two metrics relate to quality in the form of post-release defects. The quality metric is depicted at the top of the model since it is a balancing factor in relation to the technical and functional metric which are expressing an effect in a quantitative manner. All analysed method fragments were assessed on these three metrics.

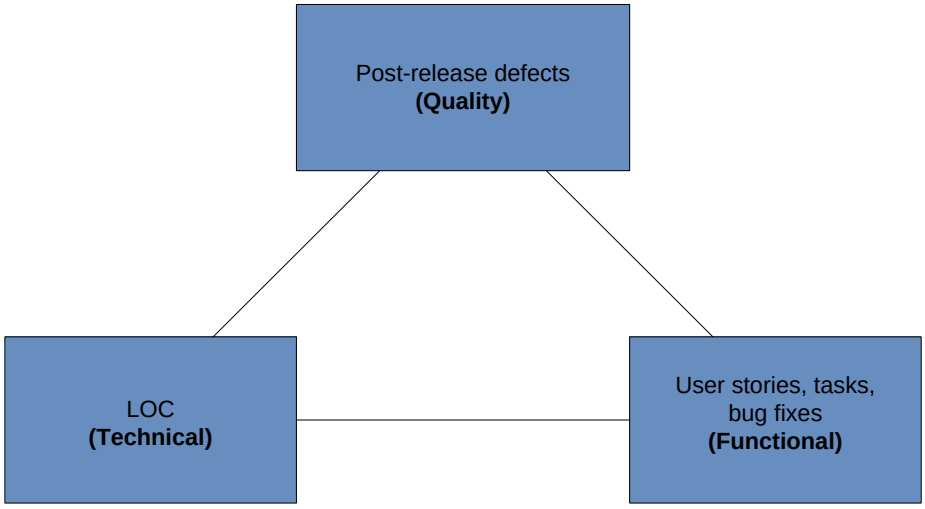

Figure 3: Triangular model for measuring productivity

3.3.1 Validity of triangular model of productivity

The triangular model for measuring productivity was validated with the support of several authors in the field of software productivity and agile in general. The productivity model including its content description was send to authors mentioned in the previous paragraph which are specialised in software productivity and have experience with conducting a large amount of studies in this research field. This was done to assess their opinion on the way productivity is measured during this research project, by means of an expert opinion the model could be validated.

Eleven authors were approached for validation purposes of which six replied with a detailed expert review. Among the respondents are highly regarded names like Jeff Sutherland (founder of Scrum) and Pekka Abrahamsson (numerous studies on software productivity). Jeff Sutherland found the LOC metric of less importance than the other metrics since *"applications of agile development will have less code due to refactoring, it is a conservative indicator"*. The LOC metric is indeed of less importance compared to the functionality metric, as will be explained in the next paragraph. Pekka Abrahamsson found that *"in principal, the research is proposing in the lines of measuring productivity. I see value in the work and the way you plan to capture the metrics but you are still dealing with something that it is hard to measure"*.

Other authors in the field of software productivity did also express their opinion. Laurie Williams is a professor in the computer science department at North Carolina state university and replied she *"totally agreed with the model and the perspective that quality needs to factor in due to the tradeoff between speed and quality"*. William Wood is a researcher at Nasa Langley research center and expressed *"in my opinion your set of metrics is a good collection. I do like the inclusion of a quality metric, and do agree that post-release defects are critical. I think the functionality metric is well described. LOC is a good metric for use within a particular team. In general, fewer lines of code to implement the same functionality is better. So if there are two teams who implement the same functionality, the team that used fewer lines of code was probably the more skilful. But within the same team, from iteration to iteration their skill level will stay about the same, so more lines of code in an iteration would indicate more productivity"*. Frank Maurer is professor and head of the agile software engineering group at the University of Calgary. He indicated that *"defining productivity is problematic. User stories are not cleanly defined, i.e. one user story in my team might be of different size than one user story in your team"*. The differentiation in user stories was indeed a problem which needed to be solved prior to the actual quantitative analysis. A solution was found in creating complexity scales for every team concerning its user stories based on the number of story points and/or allocated hours. This is described in detail in chapter 6.1.7. Daniel Graziotin is a Ph.D. student in computer science at the University of Bolzano and has published several papers concerning software productivity. He found it *"a nice research angle, I think you did a wise decision in including a dimension of quality in your model. People talk about quality in software productivity assessment since (at least) 1978; yet many studies are still using a ratio of size/effort, e.g., LOC/time. I would ask any researcher to also take the "human" part of developers (happiness, unique skills) into account when starting a new measurement study of software development. In my opinion, there is a huge need to innovate in software productivity research and it seems that you are taking this direction"*. The human/skill factor was not added in the productivity measurements due to complexity and time that would be spend on analysing skills of each team. This is described in more detail in the limitations section.

An overview of the reviewers and why they are considered experts is depicted in table 2.

Reviewer	Expert because of…
Jeff Sutherland	Founder of scrum
Pekka Abrahamsson	Numerous studies and publications on software productivity, additional studies specifically on pair programming
Laurie Williams	Numerous studies and publications on agile development and software productivity
William Wood	(Productivity) studies regarding XP
Frank Maurer	Numerous studies and publications on software productivity, head of an agile software engineering group.
Daniel Graziotin	Several papers on software productivity and protégé of Pekka Abrahamsson

Table 2: Overview of reviewers and the reason they are considered experts

3.3.2 Productivity scenarios

When concluding something about the degree of productivity, one would like all three parameters (LOC, functionality, post release defects) to be either positive or negative. However, there could be a situation where one or two parameters are for instance positive and the other(s) are negative. In this situation it is difficult to draw conclusions since not all parameters are indicating the same effect. When such a situation occurs, it was important to establish a proper policy on forehand how to handle this and what can be concluded from which result. Table 3 presents an overview of the possible results and corresponding conclusions.

	LOC (technical)	User stories (functional)	Post-release defects (quality)	Conclusion
Situation 1	Positive	Positive	Positive	Definite symptoms of improved productivity
Situation 2	Positive	Negative	Positive	Symptoms of improved productivity
Situation 3	Positive	Negative	Negative	Slight symptoms of improved productivity, quality needs to be improved
Situation 4	Positive	Positive	Negative	Symptoms of improved productivity, quality needs to be improved
Situation 5	Negative	Negative	Positive	No symptoms of improved productivity, quality has improved
Situation 6	Negative	Positive	Negative	Symptoms of improved productivity, quality needs to be improved
Situation 7	Negative	Positive	Positive	Symptoms of improved productivity
Situation 8	Negative	Negative	Negative	No symptoms of improved productivity

Table 3: Results and corresponding conclusions

The term ‘negative’ in this table means that the parameter value stays neutral (does not increase or decrease) or decreases. A special note needs to be made on situation 6 and 7. In the end, the amount of developed functionality (user stories) is of more worth to an organisation compared to the amount of LOC and therefor the functionality metric has a larger influence on the result than the technical metric. As stated in the literature part, customers pay for functionality and not for LOC.

4. Agile method fragments

Agile software development has many unique aspects compared to traditional development methods. The agile methodology can be split up into several methods, namely methods like scrum, XP, FDD, and DSDM. These agile methods can be broken down into method fragments, which can also be seen as the building blocks of a method. This section of the thesis will describe identified method fragments of agile development in detail. Additionally, two Process Deliverable Diagrams (PDD) were created to visualise where method fragments are situated in the overall process.

Iterative development (sprints)

One of the main propositions of agile development is that development is performed in iterative stages, also called ‘sprints’. These sprints are the period of time in which software development takes place. The length of a sprint mostly ranges from two to four weeks, depending on which agile method is used. Feature Driven Development (FDD) for instance uses default sprints of two weeks while Scrum uses four weeks by default (Cohen et al., 2003). However, every organisation can freely choose the sprint length according to its own preference. This method fragment is depicted as number one in the Scrum PDD (figure 7).

Product backlog
The product backlog contains all the specified user stories, ranked by priority, that belong to a particular software product. From the product backlog the sprint backlogs are created since sprint backlogs are a selection of development work from the product backlog. The product backlog can also be considered a list of functional and non-functional requirements that is turned into functionality (Schwaber and Sutherland, 2010). This method fragment is depicted as number three in the Scrum PDD (figure 7).

Sprint backlog
The sprint backlog contains all the user stories/requirements that are selected for a particular sprint. This selection is to be developed during one sprint (Schwaber and Sutherland, 2010). Sprint backlogs result from the product backlog. This method fragment is depicted as number four in the Scrum PDD (figure 7).

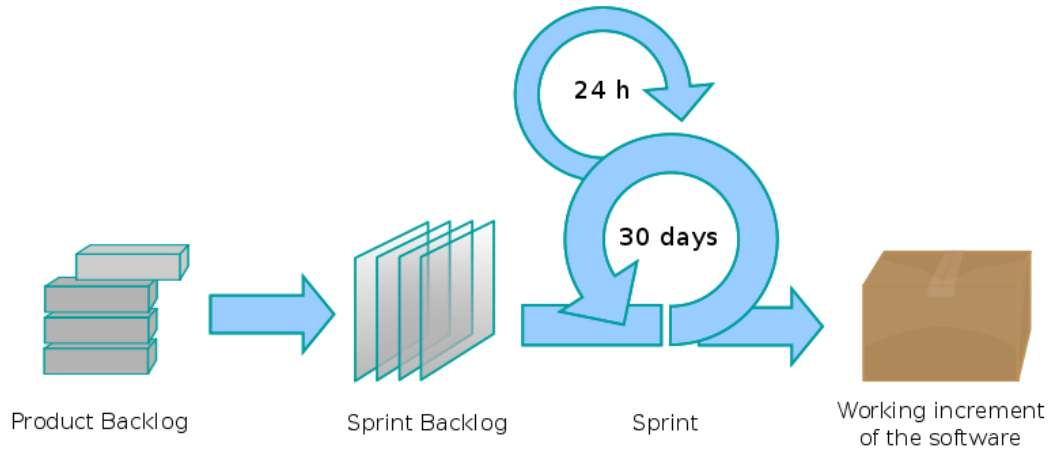

Figure 4: Basic scrum process visualized

Figure 4 visualizes the first three method fragments: the product backlog, the sprint backlog and sprints. One can see the order of the basic scrum process with sprints resulting from a sprint backlog, and a sprint backlog resulting from the product backlog.

User stories
A user story is a wish for a feature or a request for a solution to a problem, written from the customers' perspective and is part of the sprint backlog. Cohn (2004) defines a user story as a short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. User stories can also be considered requirements, since it specifies which functionality is to be developed. A large and complex user story is considered a so called epic, a theme is a collection or group of user stories. This method fragment is depicted as number two in the Scrum PDD (figure 7).

User story prioritization
This is a session held at the beginning of each sprint in which the user stories are selected for the coming sprint based on their priority. Additionally, the amount of development time is estimated for selected user stories (Cohen et al., 2003). Agile methods have various names for this method fragment. Scrum for instance uses the Scrum sprint planning and XP uses the planning game. This method fragment is depicted as number ten in the XP PDD (figure 8).

Daily standup meetings
Daily standups are meetings that are performed at the start of each working day and are attended by the development team. These meetings consist of a summary of what work has been done yesterday, what problems they encountered and what work is scheduled for the coming day. A daily standup meeting

should last only 15 minutes (Beedle et al., 1999). This method fragment is depicted as number nine in the XP PDD (figure 8).

Story board
The story board is a physical (or digital) board on the wall in the office, which holds all the user stories selected for a particular sprint. The board is divided into several status columns; namely in the order of: new, in progress, ready for testing, and done. The daily meeting can also be performed at the story board to directly relate progress or problems to a particular user story situated on the story board.

Customer involvement
Involving the customer in the development process is one of the unique aspects of agile development. A customer representative is on-site to for instance answer questions and perform acceptance tests, the result is that the customer is actively involved in the development process (Cohen et al., 2003).

Pair programming
Pair programming is about two programmers developing together. One is the so called 'driver', the other is the 'navigator'. The driver writes the actual code while the navigator reviews the code on-the-fly and gives suggestions for improvement (Williams et al., 2002). Research has proven that this principle leads to improved code quality (Williams et al., 2000). The pair programming principle belongs specifically to XP.

Unit test
Unit tests are automated tests which are run on developed functionality, the developed functionality should pass all unit tests before it moves to the next stage such as acceptance tests. Unit tests are not written by testers but by the developers themselves (Lyndsay, 2007). This method fragment is depicted as number eleven in the XP PDD (figure 8).

Acceptance test
These are tests which are defined by the customer and are used to validate the completion of a user story. Additionally, it can also give an indication if the system reacts in the expected way (Paetsch et al., 2003). This method fragment is depicted as number five in the XP PDD (figure 8).

Refactoring
Refactoring is the process of structuring a systems' internal structure. It entails structuring and improving the design of the code after it has been written (Fowler and Beck, 1999). This method fragment is depicted as number twelve in the XP PDD (figure 8).

Continuous integration
This is a practice where the development team integrates its work frequently. By doing this at least daily, it leads to reduced integration problems and results in cohesive software (Fowler and Foemmel, 2006) This method fragment is depicted as number thirteen in the XP PDD (figure 8).

Collective ownership
The code is owned by all developers and responsibility is shared across the team. Developers may make changes anywhere in the code when needed (Cohen et al., 2003).

Self-contained teams
The team is multidisciplinary. One team supports the roles of testing, development, and functional design. These roles are not spread across separate teams (Cockburn and Highsmith, 2001).

Test driven development
This development strategy is about developing automated tests (unit tests) prior to writing functional code (Janzen and Saiedian, 2005). One of the benefits of TDD is that it decreases the amount of defects and thus improves software quality (Maximilien and Williams, 2003). This method fragment is depicted as number fourteen in the XP PDD (figure 8).

Retrospective meeting (sprint review)
An evaluation of a completed sprint which aims to make the next sprint more effective (Schwaber and Sutherland, 2010). Problems, struggles, suggestions, and improvements are among things that are discussed in this session. This method fragment is depicted as number six in the Scrum PDD (figure 7).

Demo meeting
Meeting held the end of every sprint to present the software that has been developed during that particular sprint. The demo is also used to gather customer feedback (Zaki and Moawad, 2010). The main difference between the retrospective and the demo is that the retrospective is more aimed towards internal usage (only with the team itself) and the demo is additionally aimed towards external sources like customers and other departments to show and update these external sources on its progress and achievements.

Product owner
The product owner is responsible for representing the interests of stakeholders of the project and its product. Additionally, a product owner creates the initial requirements and release plans (Schwaber and Sutherland, 2010). This method fragment is depicted as number seven in the Scrum PDD (figure 7).

Scrum master
The scrum master is responsible for the overall scrum process, teaches the scrum principles to the team, and makes sure everyone complies with the practices of this agile method (Schwaber and Sutherland, 2010). This method fragment is depicted as number eight in the Scrum PDD (figure 7).

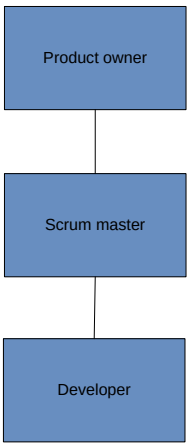

Figure 5: Hierarchy of scrum roles

Figure 5 depicts the overall hierarchy of the different scrum roles. The product owner is at the highest position since this role is responsible for the overall software product, the scrum master is beneath this role and above the team level since this role leads the teams in its scrum usage. The lowest role in this model concerns the developer (i.e. team level).

Simple design
Teams build software to a simple design. A team keeps the systems' design exactly suited for the functionality that the system has at that particular moment. There is no time wasted on a complex upfront design and this way the system is always ready for what is next in terms of development and functionality. Design is considered not an upfront thing but an all-time process (Lindstrom and Jeffries, 2004).

Open work environment
Agile development has a large social aspect. Developers work in a common workspace to improve overall communication (Cohen et al., 2003). By opting for an open work environment developers can hear conversations of each other, stay up to date with the latest progress of a project, and help each other faster due to short (verbal) communication lines.

Burn down chart
A burn down chart can help a team visualize how many features are still to be implemented during a sprint. This method fragment is not specifically for agile development, when used in an agile environment the features resemble the burn down of user stories (Cabri and Griffiths, 2006). An example of the burn down chart can be seen in figure 6.

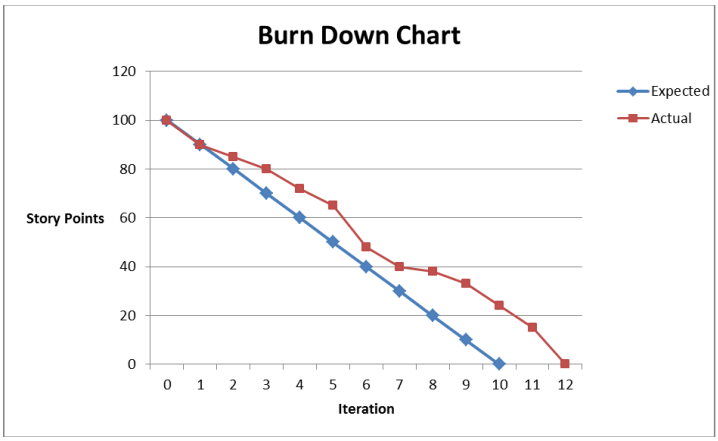

Figure 6: Sprint burn down chart

Class owner
Agile methods have a practice called collective ownership which is described previously in this chapter. FDD (Feature Driven Development) takes a different approach in that it assigns certain classes to individual developers. Thus, a developer 'owns' a class. If a feature requires changes to several classes then the owners of those classes must work together as a feature team to implement it (Ambler, 2012).

40 hour week
A principle of XP is that every week should not exceed a 40 hour limit, which means no overtime. XP allocates this because programmers should not tire themselves out by working overtime. Developers experienced that during busy periods, working overtime resulted in poor artefacts (Shukla and Williams, 2002).

Definition of done
Every user story and release has a definition of done. It defines which demands the software should conform to in order to be considered done (Kniberg, 2007).

4.1 PDD visualisation
By making use of Process Deliverable Diagrams (PDD) it is possible to visualize the various method fragments in the overall agile process. PDD's contain two aspects of a method, namely the process side and the concept side. The process side consists of all the activities from the starting point of the method to the end of the method. The concept side consists of all the deliverables that the activities produce. A PDD is written in the UML language using activity diagrams for the activity side of the PDD and class diagrams for the concept side of the PDD (van de Weerd and Brinkkemper, 2008). The overall picture of a PDD shows a meta-model of a method. The method fragments described in this chapter are also plotted on the Scrum PDD and XP PDD and are indicated with red numbers. Not all method fragments can be plotted on the PDD since some method fragments such as 'simple design', 'open work environment', and '40 hour week' are tools rather than actual activities or deliverables.

4.1.1 Scrum PDD
The PDD of the Scrum method is based on a paper by Schwaber (1995), one of the authors of scrum.
It includes a more detailed approach concerning Scrum processes and was therefore applicable to use for modelling a PDD specifically of scrum. The paper may be dated but one would like to use the original (scrum) source for it is authentic and also because of validation purposes.

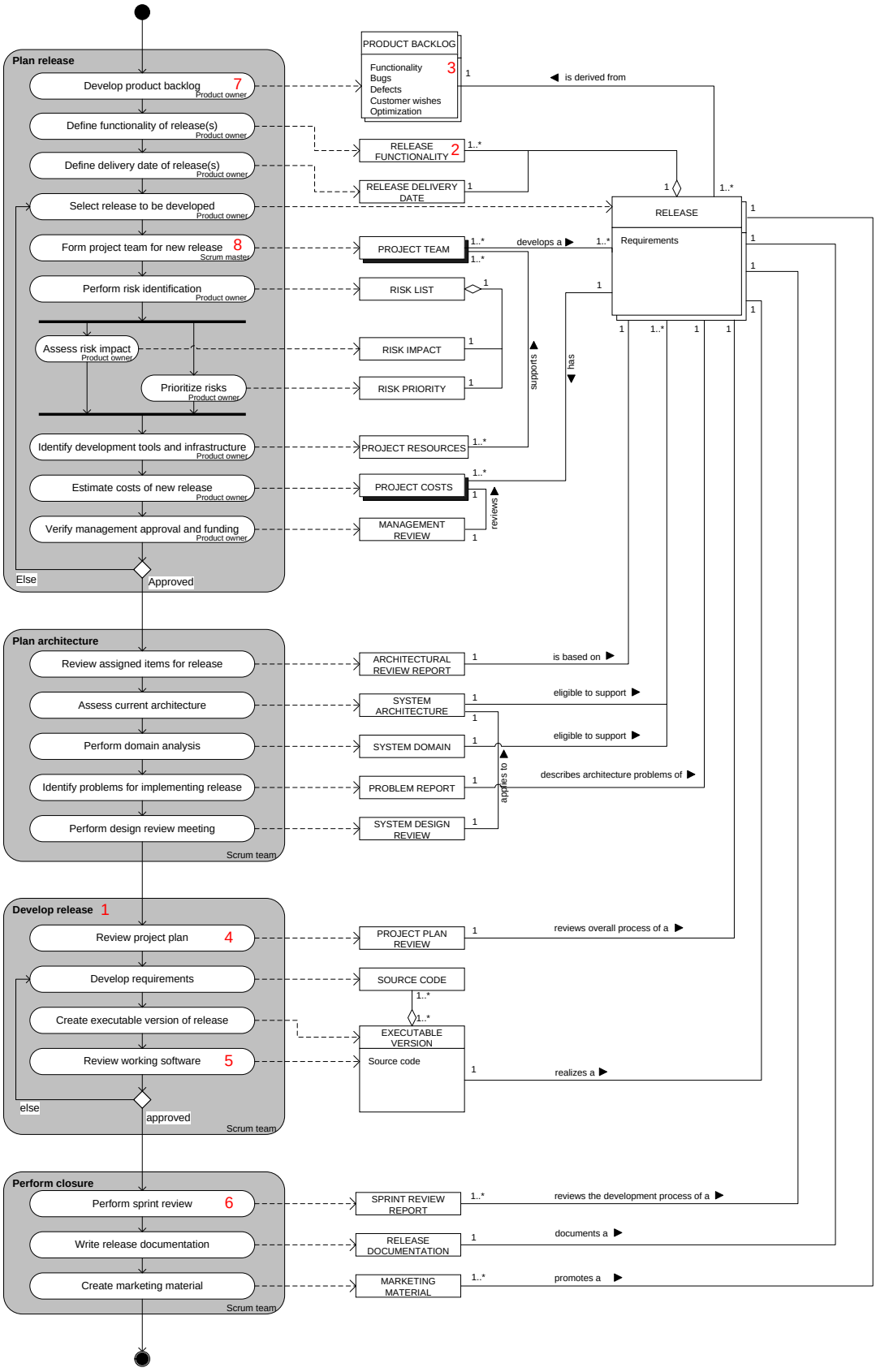

Figure 7: Process Deliverable Diagram of Scrum

Next to the meta-model, a PDD is also described textually in an activity table and a concept table. The activity table (table 4) corresponds with the process side of the PDD while the concept table (table 5) corresponds with the deliverable side of the PDD (van de Weerd and Brinkkemper, 2008).

4.1.1.1 Activity table

Activity	Sub activity	Description
Plan release	Develop product backlog	A PRODUCT BACKLOG is created by the product owner. It holds all the items that are to be developed over multiple releases. Items in the PRODUCT BACKLOG can be functionality, bugs, defects, customer wishes and optimization aspects.
	Define functionality of release(s)	The product owner defines the functionality of one or multiple releases, which results in an overview of RELEASE FUNCTIONALITY.
	Define delivery date of release(s)	Defining the date of which the customers can expect the new release results in a RELEASE DELIVERY DATE.
	Select release to be developed	The product owner is responsible for selecting which RELEASE is developed first or next.
	Form project team for new release	A PROJECT TEAM is required for developing the next release. This PROJECT TEAM is set up by the scrum master.
	Perform risk identification	The product owner identifies risks for the coming release. The identified risks form a RISK LIST.
	Assess risk impact	The RISK IMPACT is based on the disturbance it will generate when this risk occurs.
	Prioritize risks	Every risk has a RISK PRIORITY.
	Identify development tools and infrastructure	By identifying the development tools and infrastructure, a team indicates which PROJECT RESOURCES are required for the coming release.
	Estimate costs of new release	Every business project has costs and therefore the PROJECT COSTS need identification.
	Verify management approval and funding	The management needs to approve aspects like the coming RELEASE and the PROJECT COSTS before the first sprint can be initiated. This results in a MANAGEMENT REVIEW.
Plan architecture	Review assigned items for release	With regard to the systems' architecture, the selected items for the coming release are reviewed in an ARCHITECTURAL REVIEW REPORT.
	Assess current architecture	The current SYSTEM ARCHITECTURE is analysed to judge whether it can support the contents of the new release.
	Perform domain analysis	The current SYSTEM DOMAIN is analysed to indicate if it supports the contents of the new release.
	Identify problems for implementing release	A PROBLEM REPORT is generated to list potential problems concerned with implementation of the release' content with regard to the SYSTEM ARCHITECTURE.
	Perform design review meeting	A meeting is performed to review the overall architectural design of the new release. This results in a SYSTEM DESIGN REVIEW.
Develop release	Review project plan	The overall project plan is reviewed, which results in a PROJECT PLAN REVIEW.
	Develop requirements	The requirements of the release are developed in this phase. Developing the various requirements generates SOURCE CODE which is the basis of an EXECUTABLE VERSION.
	Create executable version of release	The end result of developed requirements is a working software RELEASE in the form of an EXECUTABLE VERSION which is later installed at the customer.

	Review working software	The EXECUTABLE VERSION is reviewed to judge whether it functions like described prior to starting the particular sprint.
Perform closure	Perform sprint review	The overall development process of a release is reviewed via a SPRINT REVIEW REPORT.
	Write release documentation	The content of the release needs to be explained to the customers who are going to work with this release. The contents of a release are described in a RELEASE DOCUMENTATION.
	Create marketing material	The release is promoted to potential and existing customers via MARKETING MATERIAL.

Table 4: Activity table of Scrum PDD

4.1.1.2 Concept table

Concept	Description
PRODUCT BACKLOG	A product backlog lists all the gathered requirements that are aimed towards improving a particular software product. These requirements can be functionality, bug fixes, defect fixes, customer wishes and optimization (Schwaber, 1995).
RELEASE FUNCTIONALITY	The functionality that a release delivers to its end customer (Schwaber, 1995).
RELEASE DELIVERY DATE	The release delivery date is a specified date of which the customer can expect that the release will be implemented within its organisation (Schwaber, 1995).
RELEASE	A release is a formalized sellable version of a product (van de Weerd et al., 2006) In agile development a release is a deployable working software package aimed at improving a particular software product. It is the result of several iterations (sprints) of work (Schwaber, 1995).
PROJECT TEAM	A project team is a team consisting of various roles (i.e. developers, testers, scrum master) who work together in the process of creating a new release (Schwaber, 1995).
RISK LIST	The probable frequency and probable magnitude of future loss (Jones, 2006). In Scrum the risk list contains those risks that have a probability of occurring, organized hierarchically based on the risk impact and risk priority.
RISK IMPACT	The impact of a risk is the relative negative degree of influence on the overall (development) process (Schwaber, 1995).
RISK PRIORITY	The risk priority is an indicated threat level of each risk towards the overall (development) process (Schwaber, 1995).
PROJECT RESOURCES	A validation or reselection of development tools and infrastructure used to support the development process (Schwaber, 1995).
PROJECT COSTS	An estimation of the project costs including development, collateral material, marketing, training, and rollout (Schwaber, 1995).
MANAGEMENT REVIEW	The management review indicates if the management gives its approval on the overall release project and corresponding budget (Schwaber, 1995).
ARCHITECTURAL REVIEW REPORT	The architectural review report describes the impact of the assigned requirements of the coming release on the systems architecture (Schwaber, 1995).
SYSTEM ARCHITECTURE	A systems' architecture is a model which shows how a software system is structured and how its elements work together (Bass et al., 2003).
SYSTEM DOMAIN	A system domain is a representation of relationships between elements in within a particular domain. A domain is defined by, and consists of common software applications (Tracz, 1994).
PROBLEM REPORT	A description of problems and issues concerning development and implementation with regard to the systems architecture (Schwaber, 1995).
SYSTEM DESIGN REVIEW	An approach presentation in what manner each requirement that are selected for the coming release will be implemented (with regard to the systems architecture (Schwaber, 1995).

PROJECT PLAN REVIEW	An overall approach in what manner the selected requirements will be developed (Schwaber, 1995).
SOURCE CODE	A code language that can be compiled and interpreted by a computer (Schwaber, 1995).
EXECUTABLE VERSION	A compiled version of the developed requirements' source code (Schwaber, 1995).
SPRINT REVIEW REPORT	A conclusion of the overall process of a particular sprint, also including a review of the identified risks prior to starting the actual sprint (Schwaber, 1995).
RELEASE DOCUMENTATION	Documentation that accompanies a release which gives the organisation and the customer insight into the elements of a particular release (Schwaber, 1995).
MARKETING MATERIAL	Marketing material to promote the release to existing and potential customers, with the goal to sell the release to these customers. In this case aimed towards a software product (Schwaber, 1995).

Table 5: Concept table of Scrum PDD

4.1.2 XP PDD

The XP PDD is based on a paper by Beck (1999). It describes the extreme programming method and can also be considered a high level overview of agile in general compared to the Scrum PDD and its corresponding paper.

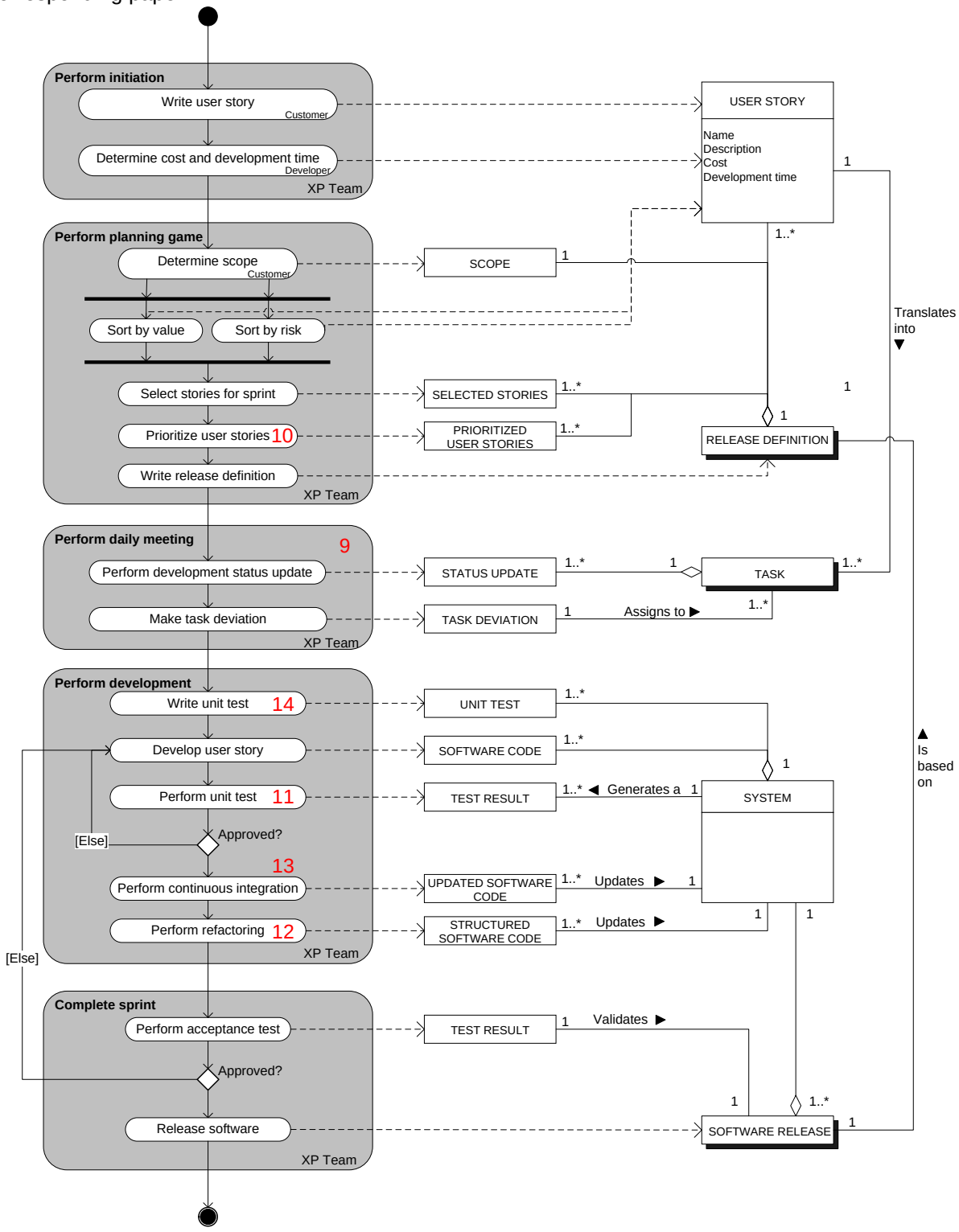

Figure 8: Process Deliverable Diagram of Extreme Programming

4.1.2.1 Activity table

Three sub-activities ('Write user story', 'Determine cost and development time', and 'Determine scope') are indicated with a separate role since these are specifically mentioned in the paper of Beck (1999). Activities and sub-activities other than the previous mentioned ones are performed by the (complete) XP team. This XP team consists of developers, testers, customer, tracker (keeps track of the schedule), coach (guides and mentors the team), and boss (Beck, 1999).

Activity	Sub-activity	Description
Perform initiation	Write user story	Writing a user story will result in a USER STORY, which describes in detail what should be developed. The USER STORY is always written by the customer. A USER STORY has a name, description (of functionality), indication of costs, and the amount of time it will take to develop.
	Determine cost and development time	After the USER STORY is written, the cost and development time are determined. This determination is done by the developer(s).
Perform planning game	Determine scope	The SCOPE is determined to identify what the amount of development time is (i.e. a month) and how many USER STORIES can be included based on the COST and DEVELOPMENT TIME. The SCOPE is always determined by the customer.
	Sort by value	The USER STORIES are sorted by their value, in an ascending order.
	Sort by risk	The USER STORIES are sorted by their risk, in an ascending order.
	Select stories for sprint	Based on the previous steps, the USER STORIES are selected for the oncoming sprint.
	Prioritize user stories	The SELECTED STORIES are then prioritized, this will identify in which order they shall be developed during the sprint. The complexity of the USER STORY is the base of the prioritization. In most sprints the more complex USER STORIES are developed first, followed by the more 'easy to develop' USER STORIES.
	Write release definition	The RELEASE DEFINITION is written, based upon the SELECTED STORIES.
Perform daily meeting	Perform development status update	During the daily meeting every team member will give a STATUS UPDATE about what he/she did yesterday, what problems he/she encountered and what on the schedule for the coming day.
	Make task deviation	When a TASK (or more tasks) is completed, a new task deviation will be made. Developers can choose their USER STORY freely.
Perform development	Write unit test	A UNIT TEST is written prior to writing actual software code. This will test the software code if it meets the criteria in order to be included in the SOFTWARE RELEASE (i.e. bugs, functionality, etc.)
	Develop user story	The actual USER STORY is developed, which results in a DEVELOPED USER STORY (i.e. software code).
	Perform unit test	The UNIT TEST will test the developed USER STORY, and results in a TEST RESULT. This TEST RESULT will determine if the USER STORY is approved or not. If it is not approved, the USER STORY will have to be fixed/edited in the activity 'develop user story'.
	Perform continuous integration	This activity is performed to update the software code (i.e. DEVELOPED USER STORIES) in the SYSTEM every few hours. By doing this, the SYSTEM will always have up to date code.
	Perform refactoring	This activity is performed to structure the software code (i.e. DEVELOPED USER STORIES) in the SYSTEM. This will create a good and workable overview of the code for the team.
Complete sprint	Perform acceptance test	The acceptance test will test the SOFTWARE RELEASE and will generate a TEST RESULT that will identify if the SOFTWARE RELEASE is approved or not. If it is not approved, the responsible

		failures will have to be fixed/edited in the activity 'develop user story'.
	Release software	The sprint is finished and the end result is a SOFTWARE RELEASE, which will be implemented at the customer.

Table 6: Activity table of XP PDD

4.1.2.2 Concept table

Concept	Description
SCOPE	The work that needs to be accomplished to deliver a product, service, or result with the specified features and functions (PMI, 2008)
VALUE	The utility of a good or service. In the case of Extreme Programming it relates to the value of a user story (such as desired functionality) (Beck, 1999).
RISK	The probable frequency and probable magnitude of future loss (Jones, 2006). In extreme programming risk indicates the probability that these aspects could occur.
USER STORY	A user story describes functionality that will be valuable to either a user or purchaser of a system or software (Cohn, 2004). In the case of extreme programming, attributes of a user story are the name, description, cost, and development time.
SELECTED STORIES	The user stories for the next release are chosen based on the most valuable features from among all the possible stories, as informed by the costs of the stories and the measured speed of the team in implementing stories (Beck, 1999).
PRIORITIZED USER STORIES	The selected stories for a sprint are prioritized to which order they shall be developed (Beck, 1999). The complexity of the user story is the base of the prioritization. In most sprints the more complex user stories are developed first, followed by the more 'easy to develop' user stories.
TASK	The team breaks the stories down into tasks, units of implementation that one person could implement in a few days (Beck, 1999).
STATUS UPDATE	Each day begins with a stand-up meeting of the team, lasting no more than 15 minutes. During this meeting, each team member reports on what they did yesterday, what they'll do today, and any problems they encountered during development (Sharp and Robinson, 2008).
TASK DEVIATION	Programmers sign up for the tasks they want to be responsible for implementing (Beck, 1999).
UNIT TEST	This kind of software test executes a large number of test inputs that extensively exercises the unit under test (Xie and Notkin, 2006).
DEVELOPED USER STORY	Software code that represents a user story. This software code could be a functionality, bug fix, etcetera.
TEST RESULT	As developers complete each task, they integrate its code and tests with the current system. All tests must run or the code cannot be integrated (Beck, 1999).
UPDATED SOFTWARE CODE	New code is integrated with the current system after no more than a few hours (Beck, 1999).
STRUCTURE SOFTWARE CODE	The design of the system is evolved through transformations of the existing design that keep all the tests running (Beck, 1999).
RELEASE DEFINITION	The customer chooses the user stories for a sprint, coming together in a release definition (Beck, 1999). The release definition describes the requirements that shall be developed in a particular sprint, so the customer will know exactly what kind of updates there are in the next software release.
SYSTEM	Over the course of the iteration, the developers implement their tasks. As they complete each task, they integrate its code and tests with the current system. So the system contains the code and tests (Beck, 1999).
SOFTWARE RELEASE	A software release is a formalized sellable version of a product (van de Weerd et al., 2006). In extreme programming the software release is the end product of a sprint, containing the developed user stories selected by the customer.

Table 7: Concept table of XP PDD

4.1.3 PDD validation

This section of the thesis will describe the validation of the two created PDD models. The validation was performed with two (experienced) scrum coaches in a meeting where the models were explained and discussed. Both scrum coaches are listed anonymous in this thesis and are therefore referred to as scrum coach A and scrum coach B.

Scrum coach A

Amount of experience	3 years in total with scrum in a business environment, 1.5 years specifically as a scrum coach
Amount of teams assisted in scrum adoption and usage	6
Experience with agile methods	Mainly scrum, some aspects of XP and kanban

Table 8: Portfolio of the first scrum coach

"The PDD models give a clear insight into the attention points which are mostly handled implicitly in practice. Both models have a high level of detail which especially for this research is beneficial. Both papers from Schwaber and Beck are visualized correctly in both PDD models. A downside of these papers is that they are somewhat dated." (the reason for using these papers is described in paragraph 4.1.1). *"Another remark is that, although both papers also do not describe this in much detail, the iterative character of agile is not that visible."*

Scrum coach B

Amount of experience	5 years in total with scrum in a business environment, 1.5 years specifically as a scrum coach
Amount of teams assisted in scrum adoption and usage	20+
Experience with agile methods	Scrum and kanban

Table 9: Portfolio of the second scrum coach

"Both models are really interesting and show the overall scrum process in a high level of detail. As a scrum coach one does most of the steps visualized in the model without really being aware that these steps are performed. The models also give insight what happens underneath the so called hood."

"This also counts as a remark on the models. Since scrum is a different way of working compared to traditional processes it is quite a change for some companies because of its relative simplicity. Organisations are not used to solving software problems in a simple manner. The papers give a detailed analysis of the method which almost makes agile development complex again instead of retaining its relative simplicity. One should not over-analyse these kinds of methods, although the models visualize what the papers describe."

"A second remark is that both papers are rather dated" (as with scrum coach A, the reason for using these papers is described in paragraph 4.1.1).

Both scrum coaches mention the high level of detail of the PDD models and the correspondence with the original papers. Scrum coach A and B had remarks about the age of both papers, but this issue is already addressed in both validation texts. Given the fact that, especially the paper of Schwaber, it is described in a scientific and formal manner one should notice that it is the way in which it is performed in practice but could also be described in a more simple manner. The scientific nature of both papers was one of the beneficial aspects in order to keep a certain amount of formalism concerning thesis project purposes.

5. Eligible method fragments

Process changes relate to insertion, change or deletion of an agile method fragment. Prior to measuring the effects of the various agile method fragments, it had to be clear what exactly makes agile method fragments eligible to be able to perform productivity effect measurements on. The following criteria were set up:

- Results from literature with regard to agile method fragments influencing productivity;
- Agile method fragments which are inserted, changed or deleted (i.e. process changes);
- Qualitative indications of influencing productivity from interview sources.

5.1 Literature background

Literature was analysed based on a specification of influencing productivity either positive, neutral or negative. By making use of a split search strategy, namely on process changes in general influencing productivity and more specifically on agile method fragments influencing productivity, it was possible to identify literature on a general level and a more specific level.

In general, process changes are intended to lead to software process improvement. Paulk et al. (1993) describe that software process improvement is aimed towards improving product quality, increasing productivity, and reduced cycle time for product development. Several studies acknowledge this (Diaz and Sligo (1997), Haley (1996), Humprey et al. (1997), and Curtis (2000)). Kellner et al. (1999) mentions that a large factor concerning process improvement is that adoption of new technology in the software development process will affect parameters like productivity. A study by Grover et al. (1998) states that redesign of a software development process and productivity have a complex relationship which is hard to put in numbers yet a mediating or moderating effect is possible. The analysis of Green et al. (2005) takes a more personal approach as their results show that increases in productivity are due to acceptance at developer level. The study also provides acceptance models aimed towards formal SPI methods in order to have them perceived which will result in a positive impact on productivity.
From literature it can be concluded that there are multiple approaches and conclusions as to whether process changes in software development influences productivity.

More specific literature concerning agile method fragments and their influence on productivity are described per method fragment in the next section. Not all agile method fragments have literature contribution since there has not been much research performed on this level of detail, namely method fragment level and its influence on productivity.

Iterative development (sprints)
No hard figures on influencing productivity for the sprints method fragment, however studies indicate that the amount of sprints completed, thus experience, leads to increased productivity per sprint until it stagnates at a certain point in time. (Sutherland, 2005, Adams and Capiluppi, 2011).

Test driven development
According to George and Williams (2004) test driven development leads to a 16% increase in time taken to develop an application compared to teams who did not define tests prior to starting development on functionality. Muller and Hagner (2002) acknowledge this and indicated a slight increase in terms of time for completing a task.

Collective ownership
A study by Fitzgerald et al. (2006) concluded that if one developer is busy, another can make the necessary adjustments and changes. This means that developers do not have to wait on one another and can continue working, which could influence productivity in a positive manner.

Pair programming
A survey by Williams and Kessler (2000) concluded that pair programming results in productivity gains and quality improvements. Additionally, Parrish et al. (2004) stresses that pair programming requires

significant resources and a prescribed protocol in order to influence productivity in a positive manner. A study by Nosek (1998) showed that more functionality was developed in less amount of time. In contrast however, Hulkko and Abrahamsson (2005) conclude that pair programming does not result in consistent improved productivity. Aligning with this statement is the study of Arisholm et al. (2007) which concludes that pair programming does not reduce the amount of time required to perform a task nor does it increase the amount of correct solutions.

Refactoring
A study by Moser et al. (2008) showed that during three out of five iterations the productivity (measured in LOC per hour) stayed constant while two iterations showed improvement.

5.2 Insertion, change or deletion

Next to the available literature on both process changes in general and on method fragment level in relation to productivity, the availability of process changes within the selected teams was another factor in establishing eligible method fragments. Various process changes were identified at the interviewed teams. This relates to insertion, change or deletion of agile method fragments over the course of a teams' scrum usage. Next to identifying general process changes it was important to identify the occurrence of similar process changes over several teams. This enabled comparing different teams on the same process change, resulting in more generalizable results. Table 10 depicts the selected teams which were used for the quantitative analysis, the actual selection method from which these teams resulted will be described in detail in chapter 6.1.3. Method fragments of the selected teams that caused a process change were:

Specific agile method fragment	**Occurred at team**
Iterative development (sprints)	4
User stories	2a, 2b, 7
Product backlog	2a, 2b, 8,
Sprint backlog	4, 8
User story prioritization	2a, 2b, 7
Scrum board	4, 7, 8
Pair programming	4, 8
Acceptance test	2a, 2b, 4, 8
Refactoring	7
Continuous integration	4, 7, 8
Self-contained teams	4
Retrospective	2a, 2b, 7
Demo	7
Product owner	7
Burn down chart	2a, 2b, 7, 8

Table 10: Method fragment occurrence within selected teams

As one can see in terms of process changes, there were fifteen (out of twenty-five) method fragments found to be eligible. Unfortunately, later in the analysis stage the quantitative data of team seven was not as well available and consistent as first thought and indicated. Due to this, two method fragments (and seven process changes) were excluded from further analysis namely the product owner and refactoring.
This resulted in a total amount of thirteen method fragments, which resembled twenty-eight process changes.

5.3 Qualitative indications

Qualitative indications of influencing productivity were withdrawn from the interviews that were conducted with scrum teams of the various departments of the case company. These qualitative indications are not scientific evidence with regard to productivity influence but more a supporting factor for establishing eligible agile method fragments.

Team 2a and 2b mentioned that dedicating the role of product owner to one person structured the scrum process and the product owner introduced several new scrum elements to the team.

Team 4 indicated that the introduction of the scrum board improved the overview of the project greatly.

Team 7 benefited from the fact that they shortened their prioritization sessions by setting a static allowed time. Prior to changing these sessions it would often take more time than allocated, resulting in less time to actually develop the determined functionality for that particular sprint. Team 8 dealt with the same issue and also introduced a static time for each prioritization session and created additional sessions purely for discussion purposes. Team 7 also indicated that the retrospective session allowed them to, if necessary, steer and adjust the team into a particular direction.

5.4 Weighing factor

The above section describes three ways of establishing whether an agile method fragment is eligible for measuring its influence on productivity. The first are those method fragments that, as stated in literature, are found to have an effect on productivity, however when these method fragments are not identified within the selection of teams it cannot be used since it is not possible to prove an effect with quantitative data. The literature on agile process changes only related to one identified process change within the selected teams, namely pair programming. The identified process changes were therefore leading in establishing whether method fragments were eligible for further analysis. These identified process changes were strengthened by statements from literature and the various qualitative indications resulting from interviews with the (selected) teams.

The weighing factor for deciding whether method fragment were eligible is depicted in figure 9.

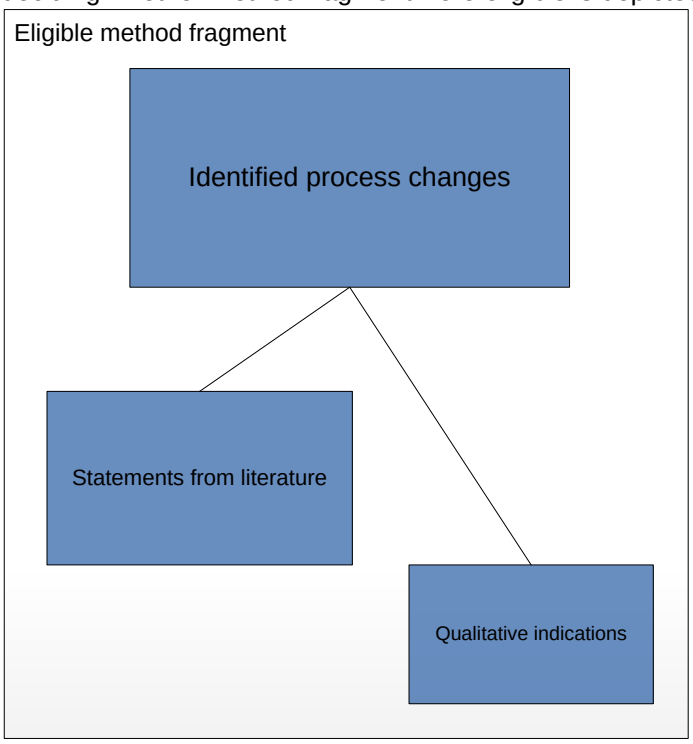

Figure 9: Weighing factor (rectangular size depicts importance)

Based on this weighing factor one can conclude that all the method fragments which were involved in the various process changes of the selected teams, were found eligible for the data analysis stage of this research project.

6. Quantitative analysis and effect on productivity

The first part of this chapter will describe the preparation that was performed in order to execute the actual quantitative analysis. Next the interview protocol which was used for all conducted interviews and the selection method which was used to select the best five teams for the quantitative data analysis stage are described. The quantitative analysis method will be explained in the part thereafter. The latter part of this chapter describes the actual quantitative analysis and corresponding effects on productivity. This part has a separate introduction.

6.1 Quantitative analysis preparation

6.1.1 Interview protocol

This section describes the protocol that was used for the total of ten interviews.

Yin (1994) describes various aspects of case studies. The teams which were interviewed for this research project can also be considered cases. The aspects which are under study are called unit of analysis and are considered the source of information. The unit of analysis concerning the cases of this research are:

- Quantitative data (LOC, functionality, PRD)
- Process changes (introduction, change, or removal of method fragments)
- Documents (release notes, retrospectives)
- Qualitative data (information resulting from various interviews)

These units of analysis have two kinds of designs according to Yin, namely holistic designs and embedded designs. A holistic design is used when the research uses a single unit of analysis, embedded designs are used when the research uses multiple unit of analysis. The embedded design is aimed towards looking for consistent patterns of evidence across these units.
This research tries to establish a productivity pattern across the multiple units, especially by means of quantitative data, therefore the embedded design is used for this research project. By opting for the embedded design one enables multiple sources of evidence (chain of evidence) which improves the construct validity.

6.1.2 Interview plan

The interview questions were created and were placed in categories that were set up prior to creating the actual interview questions. There were four categories in total:

- Overall scrum level
- Method fragment level
- Data level
- Organisational level

The overall scrum level referred to basic questions like the amount of experience the team had, the size of the team, the allocated sprint length, the programming language, and process changes. The method fragment level was about aspects like the way of implementing the various method fragments, whether method fragments were adjusted prior to implementing, the amount of time it took to implement a certain method fragment, and whether (the usage of) the method fragment had been adjusted over time. The data level consisted of a list of quantitative data criteria in order to check if the team in question had this kind of data available. Next to this list, the complexity of functionality was questioned (story points, allocated hours, bug priorities). The fourth category was about the organisational level which described the opinion of the development team regarding scrum, what kind of training the team had prior to using scrum in practice, and whether the results of scrum matched the expectations of the management/board. The interview template which lists the four categories and its corresponding questions is depicted in appendix II.

6.1.2.1 Identification of process changes

Apart from the questions in the four identified categories, a separate list of method fragments was used during the interview. This document was used to list the identified method fragments including a description which matches the descriptions depicted at the agile method fragments chapter. The main

purpose of this document was to identify the process changes of the interviewed team. By discussing every method fragment and identifying an introduction, change or deletion of every method fragment, it was possible to register whether there was a process changes and at which specific time (sprint number) this process change occurred. The template which was used to identify these process changes can be found in appendix I.

6.1.2.2 Semi structured, interviewee preparation and voice recordings

The interviews which were conducted with teams were semi structured. There was an interview template used for each interview yet, if needed, additional questions were asked to go more in-depth regarding certain topics. Since several teams had more than one year of scrum experience, it was deemed efficient to remind the interviewee (prior to the interview) to think about possible process changes over the teams' scrum usage. This was done in order to let the interviewee prepare for the main purpose of the interview, identifying process changes, and thus to have enough time to check the history of the teams' scrum usage.

Additionally, every interview was recorded in order to have the possibility to fall back on the original interview to clarify or recheck certain aspects of the interview.

6.1.3 Selection method

A goal was set to interview and analyse ten agile teams. However, in practice, it became clear that the manner in which the teams were to be analysed would result in more work than was allocated since the quantitative backlog was substantial. Therefore, in consultation with Utrecht University and the organisation of Centric, a decision was made to make a selection of five out of ten teams. This selection resulted from various strict criteria on which the teams were assessed:

- Quantity of identified process changes;
- Availability of quantitative data;
- Accuracy of identified process changes;
- Consistency of quantitative data;
- The amount of experience with scrum.

Based on the interview results an identification of the overall process changes could be made per team. This was one of the criteria to select teams for this research since process changes (insertion, change or deletion of a certain agile method fragment) could lead to improved or decreased productivity.
The more process changes a team had, the more interesting it became for selecting it for further analyses. Besides identified process changes from the interviews, sprint retrospective documentation was also used to identify changes in the overall scrum process. However, not every team documented it retrospective sessions, this could only be used for 3 teams.

In conjunction with the amount of process changes was the experience of the interviewed teams. The effect of insertion, change or deletion of a method fragment cannot be measured in a short period of time like one sprint. This needs to be measured over several sprints prior to and after the process change has occurred because the team for instance needs to learn to work with the newly introduced method fragment and as a result the effect can only be measured over several (future) sprints. This was also acknowledged by the interviewed teams since many indicated that a method fragment needed refinement and time to be incorporated efficiently in the overall development process (this is discussed in detail in chapter 7).Therefore it was also important to use experience as a weighing factor to be able to effectively measure the effect of process changes and to be able to draw reliable conclusions from them. The amount of analysed sprints after the process change was found sufficient since one has effectively four sprints of data if the sprint in which the process change occurred is also counted as a sprint where an effect is measured. Three sprints on itself equal a period of more than two months of development work. The amount of analysed sprints per method fragment is discussed in detail in paragraph 6.1.6.

Next to the amount of process changes, the quantitative data of every team was one of the selection criteria. Quantitative data that was to be examined and analysed for this research consisted of technical data, functional data, and data in terms of quality. This also refers to the triangular model for measuring productivity as depicted in figure 3. A list of quantitative data criteria was set up to analyse the availability regarding these three factors for every team:

- Sprint backlog(s)
- Start and end date of every sprint
- User stories
- Tasks
- Bugs
- Story points (user story)
- Allocated hours (user story, task, bug)
- Priority (task)
- Release notes

Apart from the availability of process changes and quantitative data, it was even more important that this data was as accurate as possible. During the interview sessions with the various teams, situations did arise where one could not indicate precisely (at the time of the interview) when a process change did occur. A solution to this problem was that interviewed teams, if needed, supplied more specific information regarding process changes (for instance the date or sprint number) at a later stage after the interview finished. This made the (precise) indication of process changes more reliable since one had the opportunity to go more in depth into its scrum history (such as retrospective documentation) rather than giving a broad time related estimation which would not enhance the reliability of this research.
The quantitative data had also to be consistent, for instance consistent registration of sprints, user stories, tasks, bugs, bug fixes, usage of story points and allocated hours.

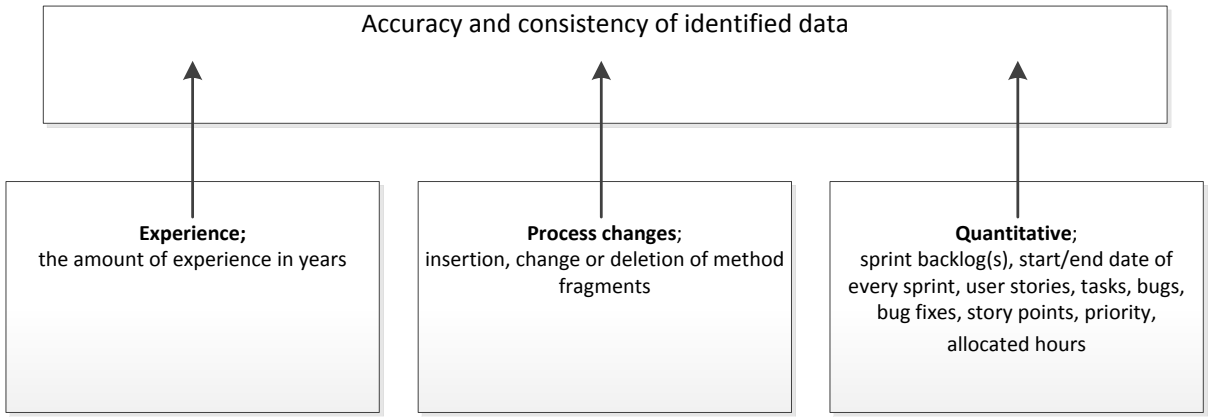

Figure 10: Criteria for selection

The model in figure 10 visualizes the selection criteria, which was used to assess every team prior to forming the eventual team selection.

6.1.4 Criteria assessment

The interviewed teams and its corresponding data were assessed based on the aforementioned criteria. The quantity of identified process changes and the overall scrum experience are depicted in the following table:

Team	Quantity of identified process changes	Amount of scrum experience in years	Amount of scrum experience in sprints
Team 1	0	0.42 year	7 sprints
Team 2a	6	1.5 years	30 sprints
Team 2b	6	1.5 years	30 sprints
Team 3	4	0.42 year	9 sprints
Team 4	7	1.08 years	17 sprints
Team 5	5	1 year	26 sprints
Team 6	1	2 years	50 sprints
Team 7	9	5 years	80 sprints
Team 8	7	1.8 years	24 sprints
Team 9	2	0.25 year	3 sprints
Team 10	4	1.33 years	30 sprints

Table 11: Quantity of identified process changes and the amount of scrum experience per team

As can be seen from table 11 the amount of process changes fluctuated between the teams. Team 1 had no process changes at all, team 6 had only one process change, the other teams had process changes ranging from four to nine. In terms of selecting teams, the more process changes a team had endured the more interesting it became for selecting it for further data analyses. See table 12 below for a more detailed overview concerning the identified process changes and precise time indication of the insertion, change and deletion of every method fragment for the ten interviewed teams. Items depicted in green are the actual process changes at every team. '+' signs indicate an addition of a method fragment, '-' signs indicate a deletion of a method fragment, and a 'c' indicates that a method fragment has changed. If a method fragment at a certain teams lists a 'yes/no' option, this means that it is not used constantly but varies over time, per sprint or percentage of time.

	Team 1	Team 2a	Team 2b	Team 3	Team 4	Team 5	Team 6	Team 7	Team 8	Team 9	Team 10
Iterative development (sprints)	Yes	Yes	Yes	C Sprint 8	C Sprint 10	Yes	Yes	Yes	Yes	Yes	Yes
User stories	Yes	+ Sprint 22	+ Sprint 22	Yes	Yes	Yes	Yes	+ Sprint 20	Yes	Yes	No
Product backlog	Yes	+ Sprint 22	+ Sprint 22	Yes	Yes	Yes	Yes	Yes	C Sprint 8	Yes	Yes
Sprint backlog	Yes	Yes	Yes	Yes	C Sprint 12	Yes	Yes	Yes	C Sprint 8	Yes	Yes
User story prioritization	Yes	+ Sprint 26	+ Sprint 26	Yes	Yes	Yes	Yes	+ Sprint 39	Yes	Yes	+ Sprint 7
Daily standup meeting	Yes	Yes	Yes	+ Summer 2009	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Scrum board	Yes	Yes/No	Yes/No	Yes	C Sprint 13	Yes/No	Yes	+ Sprint 73	+ Sprint 23	C Sprint 2	C Sprint 31
Customer involvement	No	No	No	No	No	+ Sprint 5	No	Yes/No	No	Yes	Yes/No
Pair programming	Yes/No	Yes/No	Yes/No	Yes/No	+ Sprint 4	No	No	Yes/No	+ Sprint 7	Yes/No	Yes/No
Unit test	Yes/No	Yes	Yes	Yes/No	Yes	+ Sprint 18	Yes/No	Yes	No	Yes	No
Acceptance test	Yes	+ Sprint 26	+ Sprint 26	Yes/No	+ Sprint 13	Yes	Yes/No	Yes	C Sprint 15	Yes	Yes
Test driven development	No	Yes/No	Yes/No	No	Yes/No	+ Sprint 18	No	Yes/No	Yes/No	No	No
Refactoring	Yes	Yes	Yes	+ Summer 2009	Yes	Yes	No	+ Sprint 34	No	Yes	Yes/No
Continuous integration	Yes	Yes	Yes	+ Summer 2009	+ Sprint 13	No	Yes	+ Sprint 30	+ Sprint 15	Yes	+ Sprint 8
Collective ownership	Yes	Yes	Yes	Yes	Yes	+ Sprint 2	Yes	Yes	Yes	Yes	Yes
Self-contained teams	Yes	Yes	Yes	Yes	+ Sprint 13	Yes	No	Yes/No	Yes/No	Yes	No
Retrospective	Yes	+ Sprint 18	+ Sprint 18	Yes	Yes	Yes	+ Sprint 51	+ Sprint 21	Yes	C Sprint 4	Yes/No
Demo	Yes	Yes	Yes	Yes	Yes	Yes	Yes	+ Sprint 40	Yes	Yes	Yes
Product owner	Yes	Yes	Yes	Yes	Yes	Yes	Yes	+ Sprint 39	Yes	Yes	Yes
Scrum master	Yes	Yes	Yes	Yes	Yes	Yes	yes	Yes	Yes	Yes	Yes
Simple design	Yes/No	Yes/No	Yes/No	Yes/No	Yes	- Sprint 18	Yes/No	Yes/No	Yes/No	Yes/No	Yes
Open work environment	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes
Burn down chart	Yes	- Sprint 12	- Sprint 12	Yes	Yes	Yes	Yes	+ Sprint 39	+ Sprint 8	Yes	+ Sprint 12
Class owner	No	No	No	Yes/No	No	No	No	No	No	No	Yes
40 hour week	Yes	Yes	Yes	No	Yes	Yes/No	No	No	No	No	No
Total sprints	7	30	30	9	17	26	50	80	24	4	30

Table 12: Detailed overview of identified process changes

The accuracy of process changes varied per interview, most of the teams prepared well for the interview and were therefore able to give exact dates or sprint number concerning introduction, change or deletion of certain method fragments. Only team 3 and team 7 had to perform history checkups to provide a more precise time indication.

The availability and consistency of quantitative data was examined using the separate list of criteria mentioned previously in this chapter. Every sprint of the teams was analysed on these criteria to be able to state something about the availability and accuracy of quantitative data. Nine out of the ten teams used Team Foundation Server (TFS), which is a module of Microsoft Visual Studio, for their programming, specification of functionality, and bug registration. Only team 6 used a legacy system called CVS (Concurrent Versions System) which had no remote access optionality. Overall the availability of quantitative data was considered sufficient with some teams scoring negative on certain criteria. In terms of the consistency some teams only introduced criteria items at a later stage of their usage of scrum but were stable at using them from the point that they started using them. Additionally, when such a situation did occur this was in all identified cases prior to the first identified process change of the teams in question and therefor did not affect the performed measurements. See table 13 for exact details concerning availability and consistency of every team.

Data type	Team 1	Team 2a	Team 2b	Team 3	Team 4	Team 5	Team 6	Team 7	Team 8	Team 9	Team 10
Sprints/sprint backlog	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	Yes	Yes	N/A
Sprint dates	Yes (TFS)	Yes (TFS)	Yes (TFS)	Yes (TFS)	Yes (retro)	Yes (TFS)	N/A	Yes (TFS)	Yes (retro)	Yes (TFS)	N/A
User stories	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	Yes	Yes	N/A
Tasks	Yes	Yes	Yes	Yes	Yes	No	N/A	Yes	Yes	Yes	N/A
Bugs	Yes	Yes	Yes	No	Yes	Yes	N/A	Yes	Yes	Yes	N/A
Story points	Yes	Yes	Yes, + Sprint 8	Yes	Yes, + Sprint 9	Yes	N/A	Yes	Yes	No	N/A
Priority	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes	Yes	Yes	N/A
Estimated hours	No	Yes	Yes	No	No	Yes	N/A	Yes	Yes	Yes	N/A
Release notes	Yes	Yes	Yes	Yes	Yes	Yes	N/A	Yes (TFS)	Yes	Yes	N/A

Table 13: Availability and consistency of quantitative data

6.1.5 Team selection

This paragraph will explain why a team was selected or not selected for the quantitative analysis.

The geographic location of the various teams is depicted in figure 11. The figure shows that despite the teams belonging to the same organisation, the teams are situated at different countries and locations. Teams 1 to 4 are situated at the headquarter in Gouda, team 5 in IJsselstein, team 6 in Almere, team 7 and 8 in Oostkamp (Belgium), team 9 in Deventer, and team 10 in Den Bosch.

Figure 11: Geographic location of the teams (depicted per team number)

Team 1 was not selected simply because of the lack of process changes, namely zero process changes which made it inapplicable for further analyses.

Team 2a and 2b were selected due to its process changes and its availability of data. The consistency of data of team 2b is not perfect prior to when process changes were introduced but it is stable in the period where process changes did actually occur. The experience of both teams had a positive influence in the decision for selecting these teams.

Team 3 had only one process change that could be used for data analysis since three process changes occurred prior to their usage of TFS. The lack of measurable process changes and the relatively low experience resulted in exclusion from further analysis.

Team 4 is joint second considering process changes, namely seven. The data is available from when the team started with scrum. In terms of consistency, one process change had to be measured by opting for allocated hours since it was introduced prior to when story points were used. The other six process changes were introduced after the introduction of story points and were therefor assessed by means of story points.

Team 5 experienced five process changes. Unfortunately not all data of performed projects was available and story points and allocated hours were only registered on one of the available projects. The combination of both resulted in exclusion from the selection of teams where further analyses would be performed on.

Team 6 had much experience, yet only experienced one process change. Additionally, it used a legacy source system which did not allow remote access and the team managed its bugs in a separate system not linking it to developed functionality. Based on these aspects, this team was not selected for further analysis.

Team 7 is a team from Belgium and has the most experience in total at more than five years. Next to the amount of experience, this team also had the most process changes during its scrum usage, namely nine process changes. Data was available and consistent from the start.

Team 8 is the second of the foreign teams, also from Belgium. The amount of process changes, the availability of the data, the accuracy of the data and its scrum experience made this team applicable for further investigation.

Team 9 had the least experience of the interviewed teams and had two minor process changes. The availability of data was poor due to for instance not registering its story points. Other data types were registered consistently. As a result, the team was not selected based on the aforementioned reasons.

Team 10 had experience and four process changes. Unfortunately, the team works on several different servers simultaneously which made analysing difficult. Furthermore, the analysis could only be performed purely on site instead of via remote access. Therefore, this team was not selected.

Table 14 summarizes which teams were selected and which teams were excluded from further analyses.

Selected teams	Excluded teams
Team 2a	Team 1
Team 2b	Team 3
Team 4	Team 5
Team 7	Team 6
Team 8	Team 9
	Team 10

Table 14: Overview of selected and excluded teams

6.1.6 Quantitative analysis method

The quantitative data of the selected teams was analysed in order to indicate whether the identified process changes influenced productivity in any way. A method for analysis was set up prior to starting the actual data enquiry. The following rules were established:

- The measurements will be performed three sprints previous to the occurrence of a process change and three sprints after the occurrence of a process change. The period prior to the process change is to establish the as-is situation. The period after the process change is to measure the actual effect on productivity. As explained previously, the length of the analysis period was also chosen because of the experience effect. The total amount of process changes and corresponding quantitative backlog forced scoping the amount of analysed sprints per method fragment to three sprints prior to the process change and three sprints after the process change since opting for four sprints prior to and after a change would result in an exceeding the research deadline.
- If the same process changes occur at several teams, every team will be analysed separately
- LOC of every sprint backlog item of the selected sprints will be measured

- Functionality will be measured in the form of medium to complex user stories, tasks, and bug fixes
- Only user stories with a set amount of story points are analysed
- Should story points not be available at some point, a range of hours (medium to complex) will be used
- Bug development work will be assessed on its severity level, ranging from 1 (critical) to 3 (medium)
- Bugs will be used to measure post release defects

An additional note which needs to be made in terms of LOC, is that only files with .cs as an extension were analysed. Based on several interviews, a conclusion could be made that these are the files where developers actually wrote their code. Other extensions like .sln and .csproj are automatically generated files of TFS and were therefore not applicable for analyses. However there were .cs files which were also auto-generated. Should .cs files mention that it was auto-generated, then these files were neglected in analysing LOC. The amount of LOC reflects the difference in LOC compared to the previous sprint, thus not the total amount of LOC of the source code. By opting for the difference metric instead of the total amount of LOC one can see at glance whether there is an increase, neutral or decrease in LOC.
A time indication of post release defects was based on release notes documentation. This indicated both functionality and defects fixes as well as specific dates of every release. In the analysed release notes, ITIL call numbers were found. Not all of these ITIL calls were specifically bugs/defects but also RFC (request for change) was sometimes specified in ITIL call numbers. All release notes were therefore examined in detail in order to filter out these RFC's and only measure the amount of actual bugs/defects.

6.1.7 Complexity

Complexity is a term which needed to be scoped and defined since it can be interpreted in many ways.
Complexity is a time (or space) measure. It is concerned with how long it would take (or how much capacity would be needed), at a minimum, to perform a particular task. In order to have a stable enough measure for velocity, teams need something which is close to time (Gell-Mann, 1995). User stories should correlate to relative effort and effort is comparable to time. The prioritization of user stories is done via story points. These story points should thus be based on effort and the effort should be able to take into consideration factors like time. Teams often use different type of complexity (story points and hours) scales to prioritize its user stories for a particular sprint and these scales are therefore not easily generalizable. This was also the case with the selection of teams used for the quantitative analysis of this research project. For this research we wanted to measure the same complexity of development work across the selected teams. Therefore it was crucial to establish a complexity range specifically per team in order to measure what we set out to measure. This complexity range per team allowed to group user stories and corresponding tasks into the groups 'easy', 'medium' and 'complex' and based on this order it was possible to pinpoint precisely the user stories and tasks that were to be used for analyses.

Only medium and complex user stories and tasks were used for analyses of every team, this range was used because relatively easier user stories and tasks do not adhere the development time needed to establish if there was an influence concerning productivity. An increase in medium to complex functionality enables one to state an effect concerning productivity with greater confidence. A severity level was used to group bug development work. The different levels were 1-critical, 2-high, 3-medium, 4-low. The range that was selected was severity level 1 till 3 since that matched the complexity that was selected for the user stories and tasks.

Team	Easy user stories (SP)	Medium user stories (SP)	Complex user stories (SP)	Easy tasks (H)	Medium tasks (H)	Complex tasks (H)
2a, 2b	<=2	<=4	<=8	0-4	5-20	21-70
4	<=5	<=13	<=40	0-2	3-6	7-16
7	<=2	<=5	<=13	0-16	16-40	>40
8	<=3	<=8	<=13	0-8	16-24	>24

Table 15: Complexity scales of the selected teams

Table 15 depicts the complexity scales of the selected teams. The table is split in user stories and tasks. Both have a distinction in the level of complexity namely easy, medium, and complex. The abbreviation SP relates to story points, H relates to allocated hours. As one can notice from the table, every team had a different scale regarding complexity. Only the variables which are depicted in the columns of medium and complex were used in the quantitative analysis.

6.1.8 Non completed functionality

Sometimes not all planned functionality is actually developed and completed during a sprint, situations occur in which development tasks do not meet deadlines (i.e. the end of a sprint). In this situation, the functionality in question is forwarded to the next sprint to be prioritized and planned again.
At the time of analysing the data of the various method fragments and corresponding teams it occasionally happened that development tasks went past their deadline. A policy was set up to deal with this problem. When (code)edits were performed past its deadline, then the date at which the functionality is considered done (i.e. closed status) was used to allocate in which sprint this functionality had been developed and thus completed.

6.2 Effect on productivity

This section of the thesis will describe the main part of the research project, namely the quantitative analysis and corresponding effect of the method fragments on productivity. It entails a detailed description, explanation, and discussion of the data analysis and corresponding effects. The results are depicted per method fragment.

The results from the quantitative analysis will be described on three levels, corresponding with the triangular model in figure 3. These levels are the technical level (LOC), the functional level (user stories, tasks, bug fixes), and the quality level (post release defects, will be abbreviated as PRD in the remainder of this chapter). The data of these three levels will be displayed per method fragment as well as per team. The calculation of the LOC variables is not included in this thesis due to the LOC data and calculation being considerably large in terms of the total size and would therefore not enhance the readability of the main thesis.

6.2.1 User stories

Two of the selected teams, namely team 2a and 2b, experienced a process change concerning the addition of user stories. Both teams introduced changes to its scrum process in a parallel manner, the usage of user stories was added simultaneously in sprint number 22.

Team 2a / User Stories	Sprint 19	Sprint 20	Sprint 21	Sprint 22 (change)	Sprint 23	Sprint 24	Sprint 25
LOC addition	-421	245	-414	258	167	332	793
User stories	0	0	0	0	0	0	0
Tasks	0	1	0	0	0	0	0
Bug fixes	2	37	18	11	16	16	17

Version	Nr. of defects
PRD 2.7 (Nov 2011)	0
PRD 3.0 (July 2012)	4
PRD 3.0.6 (March 2013)	5

Table 16: Quantitative results from user stories analysis of team 2a

Table 16 depicts the quantitative results of introducing user stories in team 2a. The null amounts of identified user stories are due to the fact that this team did not use story points but hours in the timeframe of analysis and as a result no user stories itself were included in the measurement process. The small amount of tasks relate to the fact that the timeframe of analyses was primarily used for bug fixing, as can

been seen by the amount of bug fixes of the measured sprints. The actual timeframe of analysis had a start date of 01-07-2012 and ended on 15-10-2012.

LOC

Prior to the introduction of user stories one can see the LOC being peaky from negative to positive to negative (sprint 19 till 21). After the process change in sprint 22 one can see that the LOC is positively affected, it drops slightly in sprint 23 but thereafter it increases by a factor 4.7 over two sprints.
Negative LOC amounts do not necessarily have to mean that no functionality was developed, it can be also be due to refactoring or deletion of for instance unused functionality of the software product.

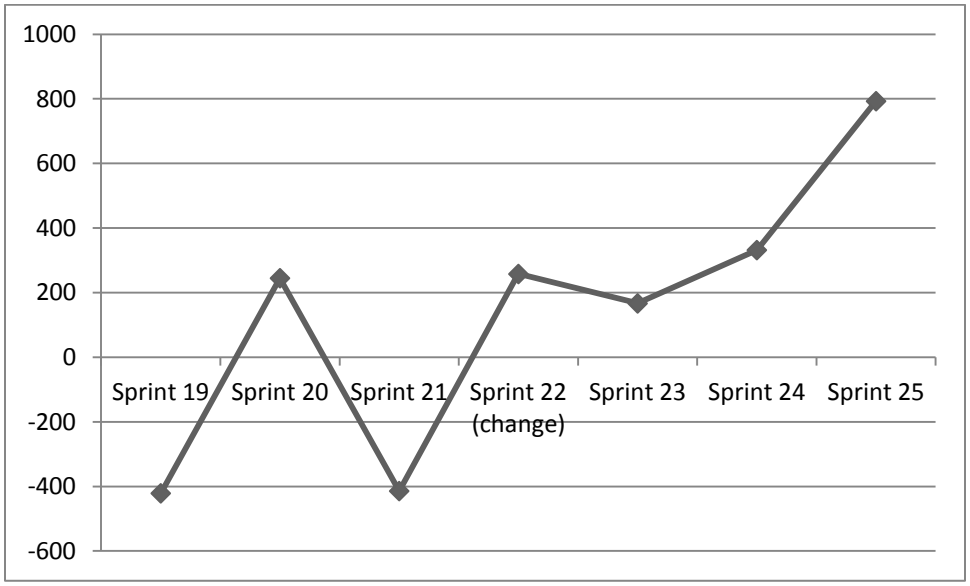

Figure 12: LOC concerning the introduction of user stories in team 2a

Functionality

Since the three types of functionality are all of the same complexity (medium to complex) it can be summed to indicate the total amount of functionality developed in a particular sprint.
One can see the functionality peaking two sprints prior to the process change, functionality decreases up till the point of introducing user stories. Right after the process change there is a slight increase in functionality, yet is does not reach the amounts that were developed prior to the process change.

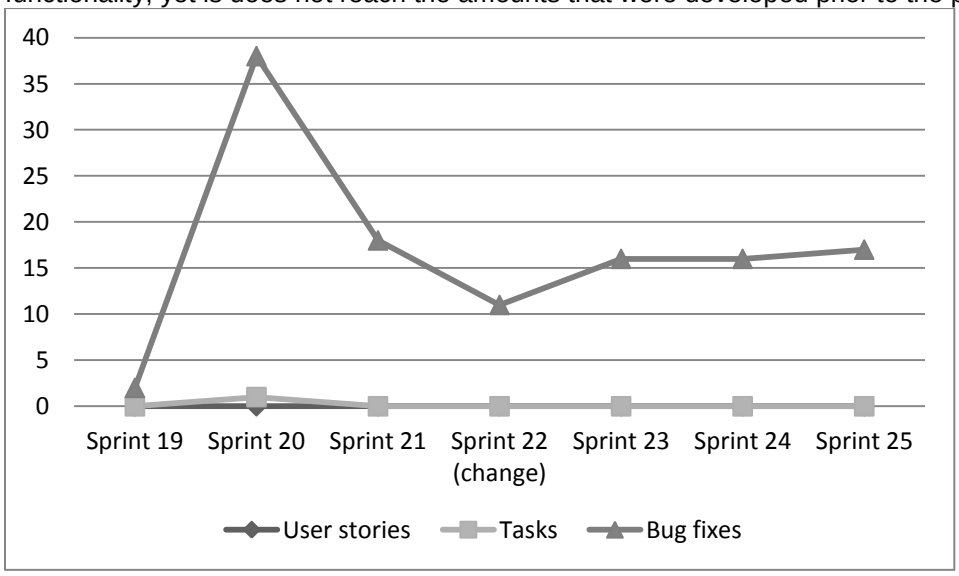

Figure 13: Functionality concerning the introduction of user stories in team 2a

PRD

Three releases of team 2a were used to measure the post release defects. Version 2.7 was released in November 2011, version 3.0 was released in July 2012, and version 3.0.6 was released in March 2013.
The amount of defects depicted at version 3.0 relate to the actual defects in version 2.7, the amount of defects depicted at version 3.0.6 relate to the actual defects in version 3.0. This way of reading the data was chosen to include three releases in order to have a release prior to the process change, a release during the timeframe of analysis, and a release after the timeframe of analysis.
The data in table 16 shows that the amount of defects increased over three releases, indicating that the quality did not improve when introducing user stories.

In order to state a conclusion about the introduction of user stories in team 2a, table 3 was used. Table 3 enables stating a conclusion based on the aforementioned three metrics. Since the LOC is positively affected, functionality is negatively affected, and the quality is negatively affected one can state that there are *slight symptoms of improved productivity, quality needs to be improved*.

Team 2b / User Stories	Sprint 19	Sprint 20	Sprint 21	Sprint 22 (change)	Sprint 23	Sprint 24	Sprint 25
LOC addition	118	245	106	23	122	134	66
User stories	0	0	0	0	0	0	0
Tasks	0	0	0	0	0	0	1
Bug fixes	5	9	1	1	25	0	13
Post release defects	No bugs reported in release notes since it was a new product which was not yet released publicly						

Table 17: Quantitative results from user stories analysis of team 2b

Team 2b has the same issue with the user stories as team 2a, hours were used instead of story points. During the sprints that were analysed for this method fragment team 2b focused, just as team 2a, primarily on bug fixing instead of developing new functionality. The timeframe of analysis was the same as that of team 2a, from 01-07-2012 to 15-10-2012.

LOC

The amount of LOC is declining from sprint 20 to sprint 22. After the introduction of the user stories method fragment, one can see an LOC increase over two sprints before declining again. LOC measurements for sprint 23 and 24 are higher than two sprints prior to the process change, however sprint 25 is lower than all sprints prior to the process change. Due to this fluctuation no sound effect can be concluded, neither positive nor negative.

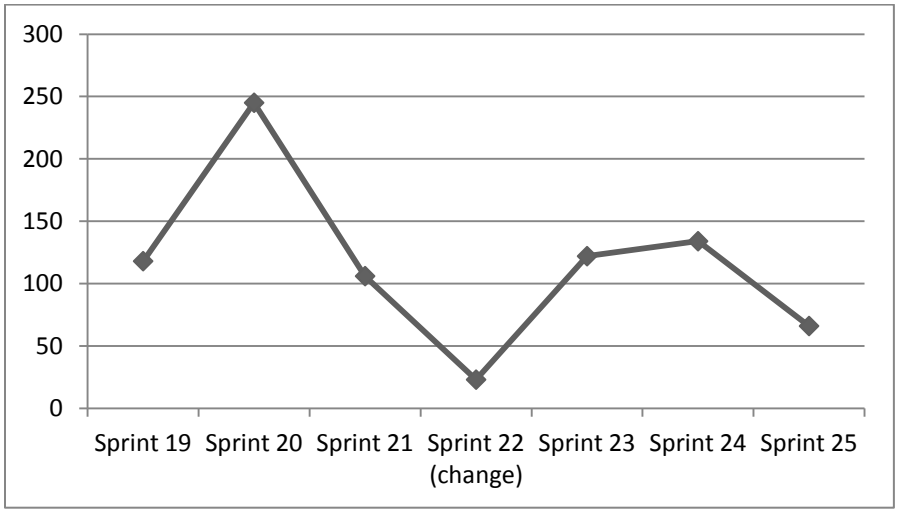

Figure 14: LOC concerning the introduction of user stories in team 2b

Functionality

From the point when user stories are introduced the amount of developed functionality is higher compared to the sprints prior to the introduction. The drop in sprint 24 was caused by a development stop in terms of functionality. The team performed bug fixing with a developer and a tester next to each other, yet the work that was performed was not specified in visual studio. Due to this development stop in sprint 24, the graph is peaky after the introduction of user stories yet the total amount of development functionality is higher compared to the period prior to introducing user stories.

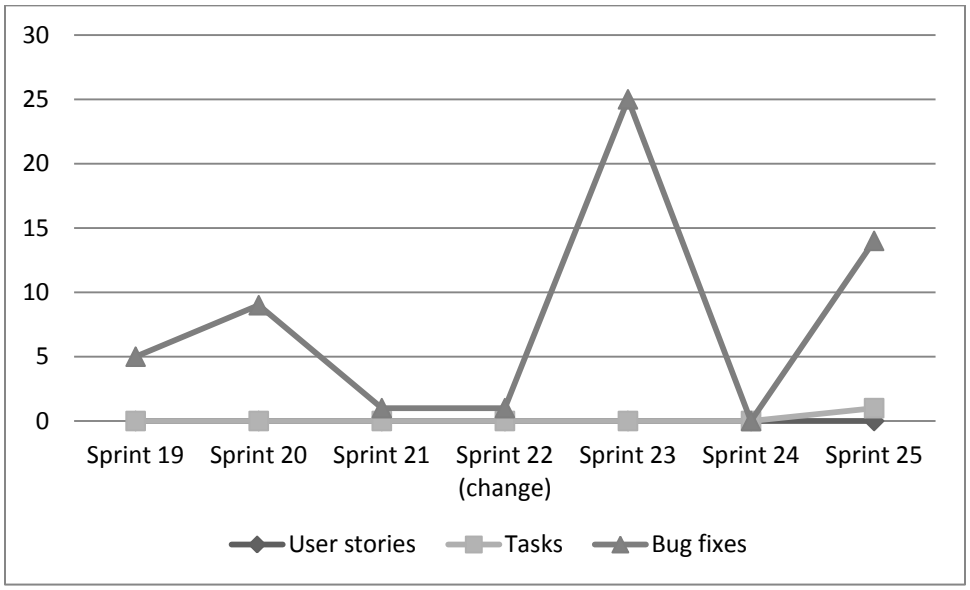

Figure 15: Functionality concerning the introduction of user stories in team 2b

PRD

There were no defects mentioned for the product in the timeframe of analysis. The team indicated there the product that was being worked on at that time was as an entirely new version and therefore did not have bugs specified to it since it was not yet released publicly.

Using table 3 one can conclude that the LOC was affected negatively, the functionality positively, and the quality cannot be concluded since there was no data on this variable that could be used. This results in *symptoms of improved productivity, quality needs to be improved.* As one can see, the effect on productivity is stronger in team 2b than in team 2a. This is because functionality is rated more important than LOC. The customer eventually pays for functionality and not for LOC.

Effect on productivity

The data and results of team 2a and 2b are contradicting, yet in both teams there is a (positive) effect on productivity. Unfortunately it is uncertain whether the effect of user stories on productivity is due to LOC or functionality. This research cannot give a definitive answer to this, analysing more teams that introduced user stories at a certain stage in its scrum usage or performing a case study specifically towards introducing user stories are options to be able to have a more sound conclusion. In terms of PRD it is not possible to draw any conclusion since team 2b has no defect reports concerning its product and therefore this metric is automatically considered negative, this also aligns with the PRD variables of team 2a which had a negative effect. Considering the contradiction in terms of the technical and functionality metric of both teams, and therefore opting for a conservative conclusion, one has to conclude that user stories have *no symptoms of improved productivity* until other research proves otherwise.

6.2.2 Pair programming

Two of the selected teams experienced a process change concerning pair programming, namely team 4 and team 8. Team 4 introduced pair programming at the beginning of sprint four, team 8 introduced pair programming at the beginning of sprint seven.

Team 4 / pair programming	Sprint 1	Sprint 2	Sprint 3	Sprint 4 (change)	Sprint 5	Sprint 6	Sprint 7
LOC addition	1877	7521	3383	1662	1992	1812	3313
User stories	4	11	12	6	11	5	2
Tasks	19	42	55	26	41	19	13
Bug fixes	0	1	0	0	5	0	0

Version	Nr of defects
PRD 3.5.8 (Feb 2012)	20
PRD 3.6.0 (July 2012)	7
PRD 3.6.1 (Oct 2012)	6

Table 18: Quantitative results from pair programming analysis of team 4

The period of analysis of this team started on 17-1-2012 and ended on 26-7-2012.

LOC

As can been seen in figure 16, the LOC peaks in sprint number 2. From that point on it declines up to sprint 6, yet the decline stabilizes after the process change. Sprint 7 is the first sprint where an increase in LOC can be noted. The data and the graph clearly indicate that there is a decrease in LOC, however sprint 7 promises that when a team has several sprints of experience with pair programming the LOC increases. Unfortunately this cannot be stated and supported with (quantitative) data since sprint 7 was the last sprint within the timeframe of analysis.

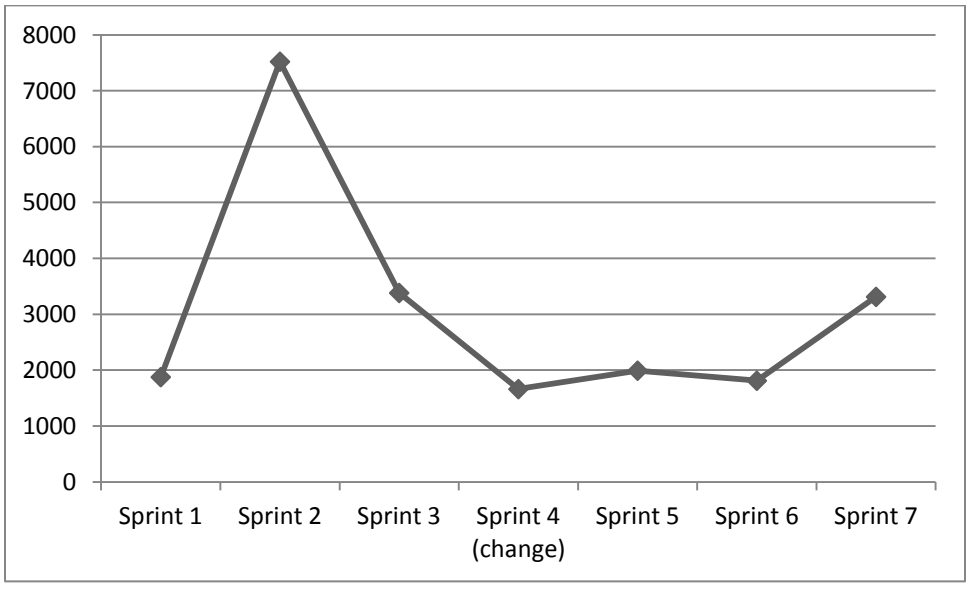

Figure 16: LOC concerning the introduction of pair programming in team 4

Functionality

In terms of developed functionality, almost the same trend can be seen as with the technical metric (LOC). Functionality peaks at sprint 3, in the form of user stories and tasks, and declines for the rest of the analysis period. Functionality decreases by a factor 2.1 up to sprint 5. An increase can be noted in sprint 5 for all three functionality metrics, yet it declines again by a factor 3.8 up to the end of the analysis period. The measurements of the last sprint indicate that all metrics are lower than the amount of the first sprint of analysis.

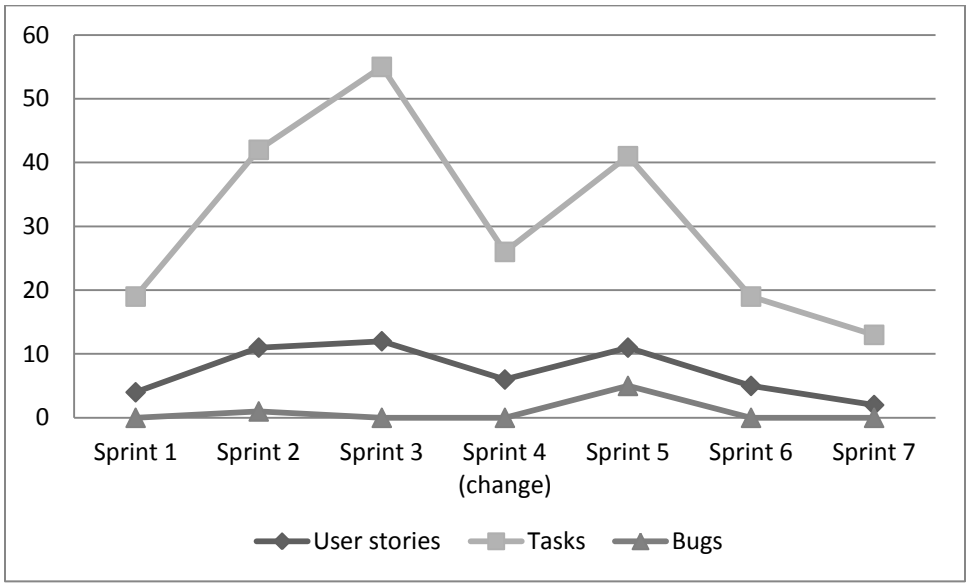

Figure 17: Functionality concerning the introduction of pair programming in team 4

PRD

One can see a trend over several releases that the quality has gone up and thus the amount of defects has gone down. The amount of defects in version 3.5.8 were twenty, version 3.6.0 indicates that the amount of defects decreased to seven. Version 3.6.1 supports this decreasing trend in post release defects since this release has six defects.

One can conclude that both LOC and developed functionality were not positively affected in team 4 concerning the introduction of pair programming. The quality was affected in a positive manner due to the decline in the amount of post release defects. Using table 3, the usage of pair programming in team 4 resulted in *no symptoms of improved productivity, quality has improved*. There are no symptoms of improved productivity because quality is not directly correlated with increased productivity since this metric is not about an increase in terms of development work but about the defect rate concerning this development work.

Team 8 / pair programming	Sprint 7 (change)	Sprint 8	Sprint 9	Sprint 10	Sprint 11	Sprint 12	Sprint 13
LOC addition	N/A	N/A	N/A	N/A	N/A	N/A	N/A
User stories	10	1	6	8	11	11	5
Tasks	29	6	24	21	33	35	31
Bug fixes	0	0	0	0	2	1	13
Post release defects	16	7	3	9	12	9	4

Table 19: Quantitative results from pair programming analysis of team 8

The LOC calculation could not be performed with the data of team 8 since this team used oracle forms (binaries) for its coding/programming and it was not possible to calculate the LOC metric since these type of files could not be analysed with the available tools. The sprint where the process change occurred is the first sprint of analysis since data prior to this sprint was not available. To deal with this problem the length of analysis after the process change was extended to conform with the length of other analysis periods. Team 8 was the only team that released software after every single (monthly) sprint, and therefore it was possible to dedicate specific defects to specific sprints. The timeframe of analysis ranged from 1-11-2011 till 31-5-2012.

Functionality

After the introduction of pair programming, the amount of developed functionality dropped substantially by a factor 5.6. From sprint 8 onwards it rises with occasional drops. In the last sprint for instance, the tasks and user stories decrease after having increased for three successive sprints. Yet the amount of bug fixes increase in the last sprint. Despite the occasional drops, both user story and bug fix metrics are higher than its initial (starting) amounts.

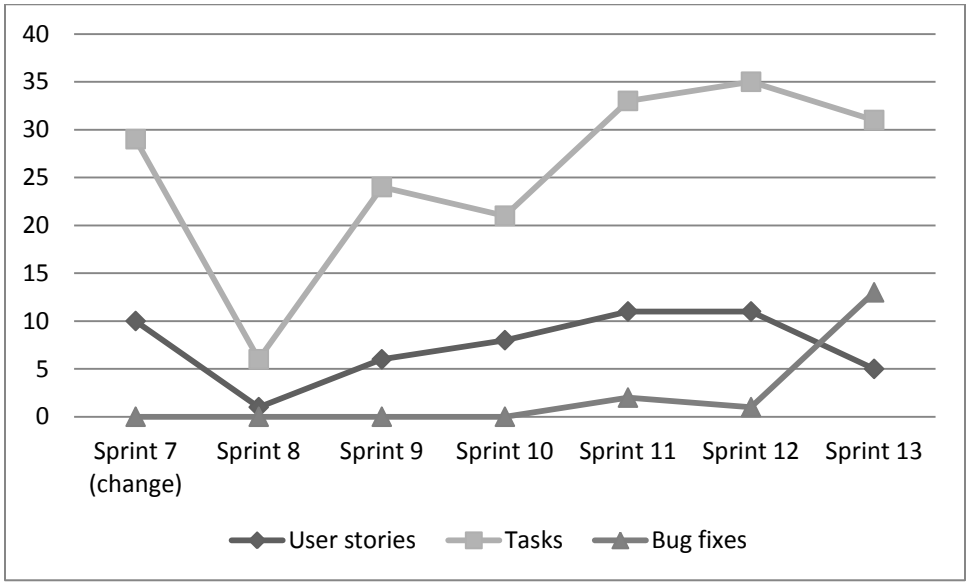

Figure 18: Functionality concerning the introduction of pair programming in team 8

PRD

Since there is data concerning PRD for every sprint, an additional graph was created. One can see that the amount of defects decline by a factor 5.3 until sprint 9. There is an increase in sprint 10 and 11 before dropping again for the last two sprints. Despite the graph peaking and dropping one can see that the amount of PRD is at any point lower than its starting position.

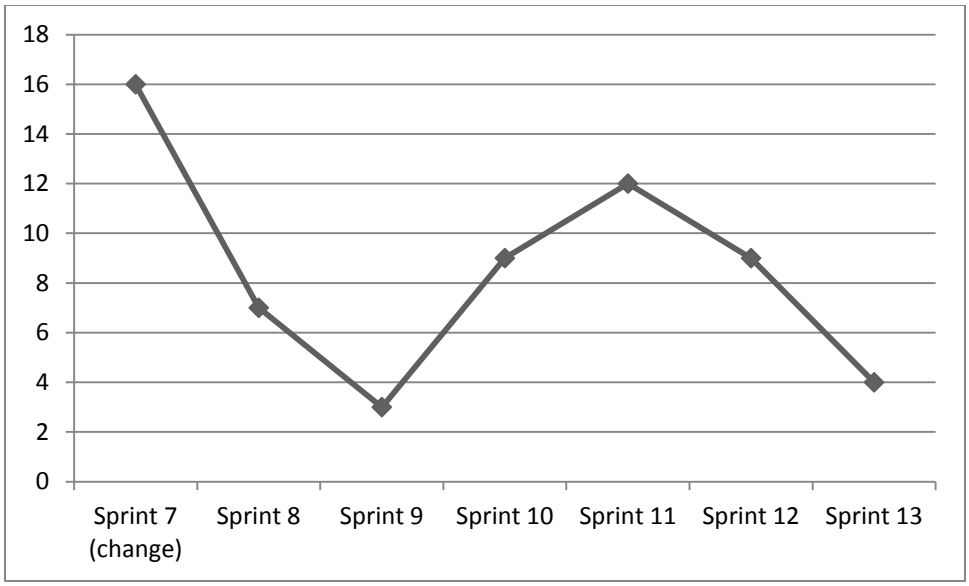

Figure 19: PRD concerning the introduction of pair programming in team 8

Data analysis of team 8 shows that there is an increase in developed functionality and the amount of PRD has decreased over the period of analysis. Since the LOC could not be calculated this was automatically considered neutral and thus negative according to table 3. The conclusion concerning the introduction of pair programming within team 8 resulted in *symptoms of improved productivity*.

Effect on productivity

The results show a contradiction in terms of developed functionality with it being negative for team 4 but positive for team 8. This disallows to draw a one sided conclusion in terms of developed functionality. Yet both team 4 and team 8 show an increase in quality, with the PRD metric decreasing at both teams. It is uncertain whether pair programming positively influences developed functionality (because of the contradicting results) but it is certain that it positively influences software quality and this result corresponds with what can found and read in literature on this specific method fragment. Adding the pair programming method fragment resulted in *no symptoms of improved productivity, quality has improved*.

6.2.3 Scrum board

Three teams experienced a process change concerning the scrum board. Team 4 switched from a digital to a physical scrum board in sprint 13. Team 7 introduced a scrum board in sprint 73. Team 8 deleted its scrum board from sprint 20 till sprint 22 and reintroduced it from sprint 23 onwards.

Team 4 / Scrum board	Sprint 10	Sprint 11	Sprint 12	Sprint 13 (change)	Sprint 14	Sprint 15	Sprint 16
LOC addition	2108	578	4245	1976	2070	9	458
User stories	3	6	2	4	3	5	3
Tasks	10	13	21	0	0	3	0
Bug fixes	1	4	13	6	8	8	4

Version	Nr. of defects
PRD 3.6.0 (July 2012)	7
PRD 3.6.1 (Oct 2012)	6
PRD 3.6.2 (Feb 2013)	1

Table 20: Quantitative results from scrum board analysis of team 4

The period of analysis ranged from 06-09-2012 till 20-02-2013.

LOC

The LOC peaks at sprint 12, prior to the process change in sprint 13, and declines by a factor 9.3 over four sprints. There is a slight increase in sprint 16 but this is well under the starting LOC amount of sprint 10. The noticeable low amount of LOC addition in sprint 15 is due to refactoring processes.

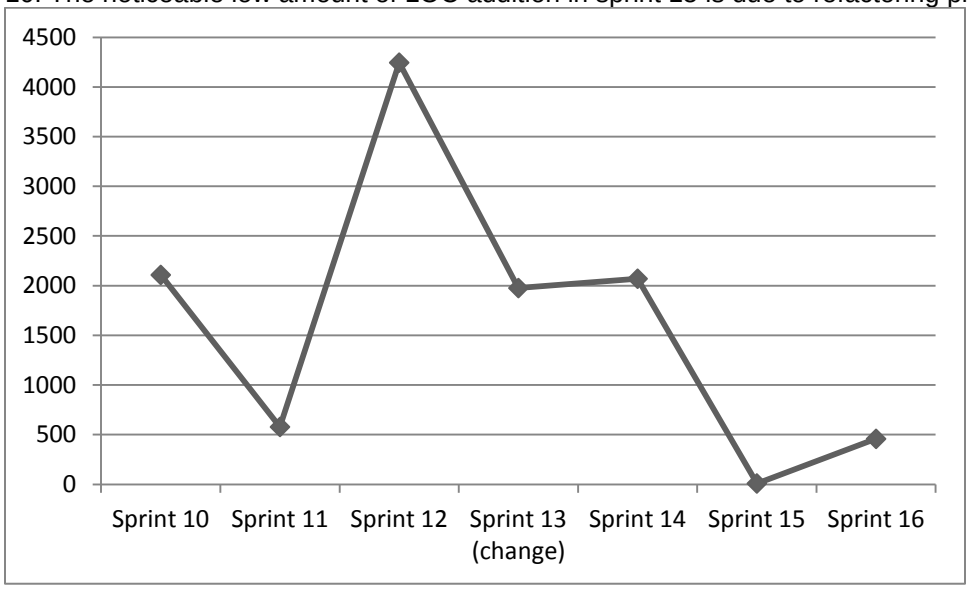

Figure 20: LOC concerning the introduction of the scrum board in team 4

Functionality

Both tasks and bug fixes peak at sprint 12 before declining in the sprint in which the process change takes place. On the contrary, user stories slightly go up in that particular sprint. After sprint 13 the bug fixes and user stories increase before decreasing slightly in sprint 16, yet both stay above the starting amount of sprint 10. Tasks only increase in sprint 15 before decreasing again in sprint 16.
Overall one can see that the total amount of developed functionality after the introduction of the scrum board, in three out of four sprints, stays below the values of the sprints prior to the introduction of the scrum board. This is why there is no significant positive effect on productivity.

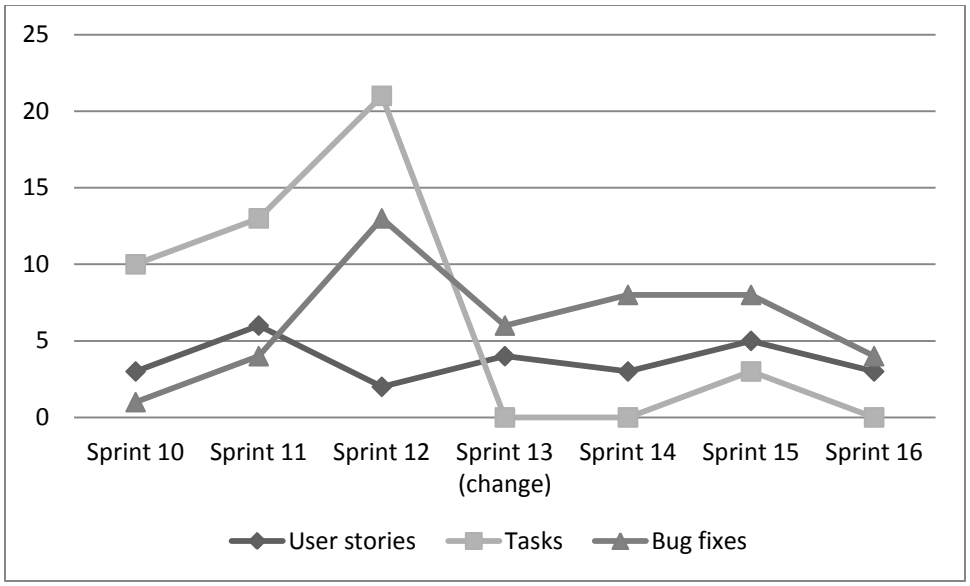

Figure 21: Functionality concerning the introduction of the scrum board in team 4

PRD

Three releases capture the period of the introduction of the scrum board. Two releases, namely 3.6.0 and 3.6.1, were also used in the process of analysing the pair programming method fragment. Additionally, release 3.6.2 was also assessed to measure what the defects were shortly after the timeframe of analysis (as was done with all PRD analyses). One can see that the amount of defects decreased to a single defect in release 3.6.2.

Based on the first two metrics, LOC and functionality, one can conclude that these were not positively affected by the introduction of the scrum board. The PRD decreased and was therefore positively affected. Since quality is the only metric which was affected positively, it resulted in *no symptoms of improved productivity, quality has improved* for reasons mentioned previously. The PRD analysis partly used the same data as the pair programming method fragment did.

Team 7 / Scrum board	Sprint 70	Sprint 71	Sprint 72	Sprint 73 (change)	Sprint 74	Sprint 75	Sprint 76
LOC addition	-42	-800	1116	1865	-955	688	832
User stories	1	3	0	1	0	0	0
Tasks	14	13	9	6	2	16	5
Bug fixes	14	7	0	7	15	59	20

Version	Nr. of defects
PRD 3.2 (Apr 2012)	20
PRD 4.0 (Aug 2012)	263
PRD 4.1 (Feb 2013)	344
PRD 4.2 (July 2013)	76

Table 21: Quantitative results from scrum board analysis of team 7

The negative LOC variables are due to refactoring processes. Another note to be made is that team 7 does not include (solved) bugs in its release notes but stores it digitally per version number.
The timeframe of analysis was from 20-8-2012 till 11-1-2013.

LOC

The LOC addition peaks in the sprint where the scrum board is introduced yet in the next sprint drops down substantially. There is an increase in sprint 75 and 76 but the graph is going too much up and down, also below the starting amount, to conclude any kind of significance.

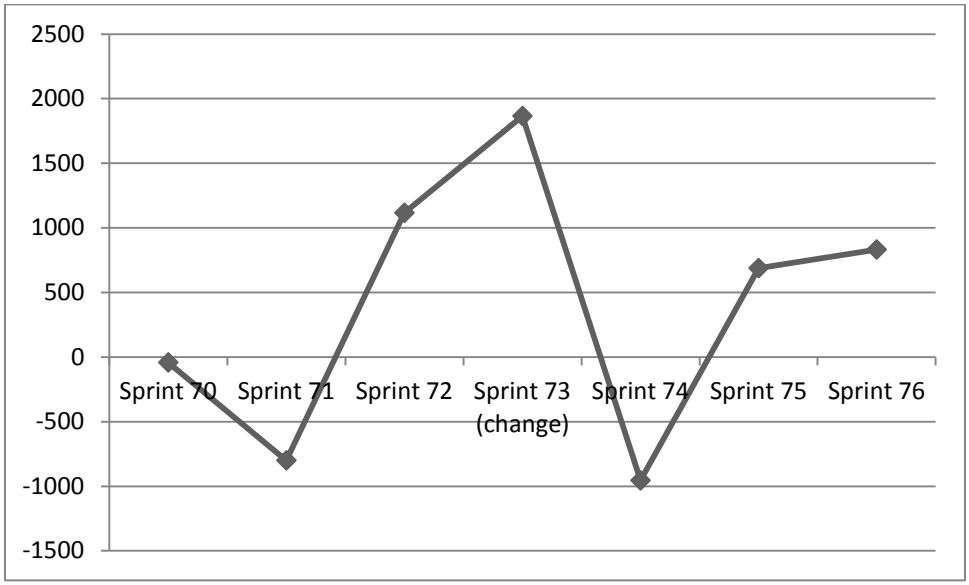

Figure 22: LOC concerning the introduction of the scrum board in team 7

Functionality

The amount of developed user stories stays relatively stable across the timeframe of analysis. The amounts of tasks are decreasing up to one sprint after the introduction of the scrum board. The second to last sprint is the first sprint where an increase can be noted before decreasing again in the last sprint. The third functionality metric, in the form of bug fixes, is increasing especially after the process change in sprint 73. The first decrease can be noted in the last sprint which was analysed, however the amount of bug fixes stay above the amounts of the sprints prior to the actual process change. Overall, the functionality is increasing for three sprints after the introduction of the scrum board yet decreasing in the last sprint to the initial amounts. The amount of developed functionality is higher after the introduction of the scrum board than prior to this period, and was therefore positively affected.

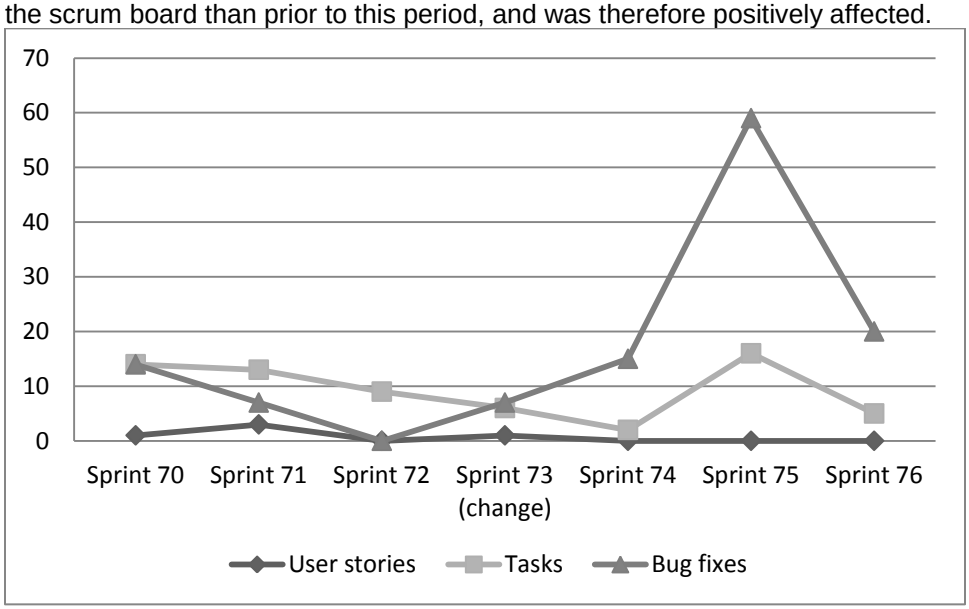

Figure 23: Functionality concerning the introduction of the scrum board in team 7

PRD

The amount of post release defects of team 7 increased substantially over the course of four releases. This was especially the case in the releases that were developed in the actual timeframe of analysis, namely version 4.0 and 4.1. One can see a decrease in the amount of defects in version 4.2, yet it is still above the initial amount of version 3.2.

Based on the measurements of the three metrics once can conclude that only the amount of developed functionality was effected in a positive manner, resulting in *symptoms of improved productivity, quality needs to be improved*.

Team 8 / Scrum board	Sprint 20	Sprint 21 (change/del)	Sprint 22 (change/del)	Sprint 23 (change/add)	Sprint 24	Sprint 25	Sprint 26
LOC addition	N/A	N/A	N/A	N/A	N/A	N/A	N/A
User stories	9	5	7	3	7	4	6
Tasks	32	30	29	27	37	36	30
Bug fixes	12	14	11	16	11	19	49
PRD	9	18	9	9	1	3	1

Table 22: Quantitative results from scrum board analysis of team 8

Team 8 deleted its scrum board for two sprints before reintroducing it in sprint 23. As mentioned previously at the pair programming method fragment of team 8, LOC could be not be calculated. The analysis period spanned from 1-10-2012 till 29-4-2013.

Functionality

One can see in figure 24 that the amount of user stories was not affected by the removal and reintroduction of the scrum board since the line is relatively stable. The amount of tasks increase the sprint after the scrum board has been added to the process before decreasing slightly over the last two sprints. The amount of bug fixes increased since the reintroduction of the scrum board by a factor 3.1.
The overall functionality went down slightly in the sprints in which the scrum board was removed and increased the sprint after the reintroduction to an amount higher than the initial amount.

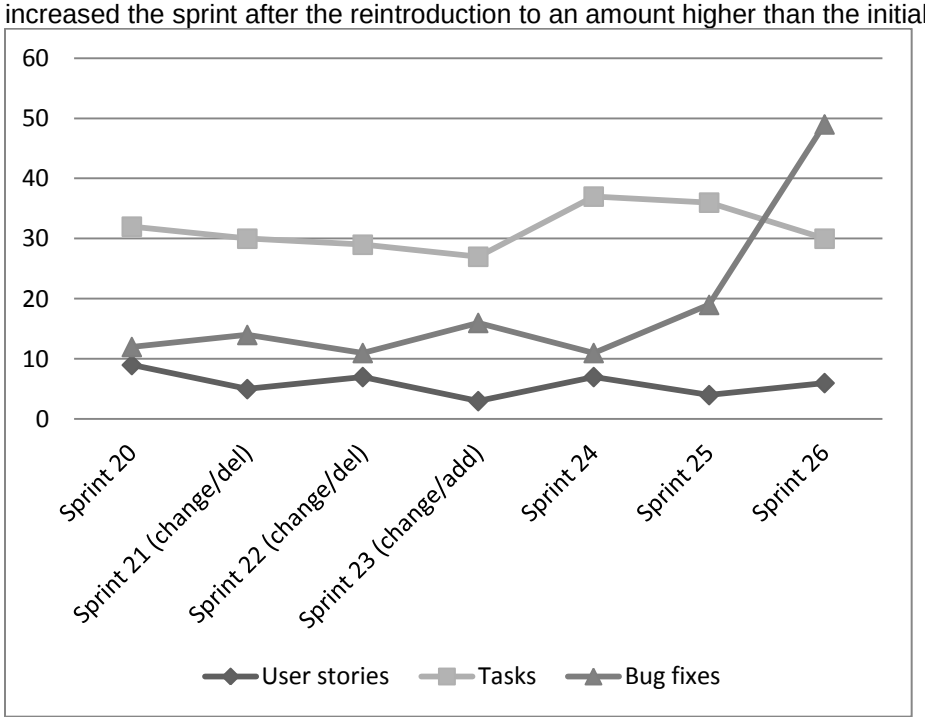

Figure 24: Functionality concerning the introduction of the scrum board in team 8

PRD

The amount of PRD increased in the first sprint of the deletion of the scrum board before stabilizing in sprint 22 and 23. After the reintroduction of the scrum board one can notice a decrease in sprint 24, 25, and 26. The usage of the scrum board in team 8 shows that, in contrast to team 7 and in conjunction with team 4, the PRD level decreases.

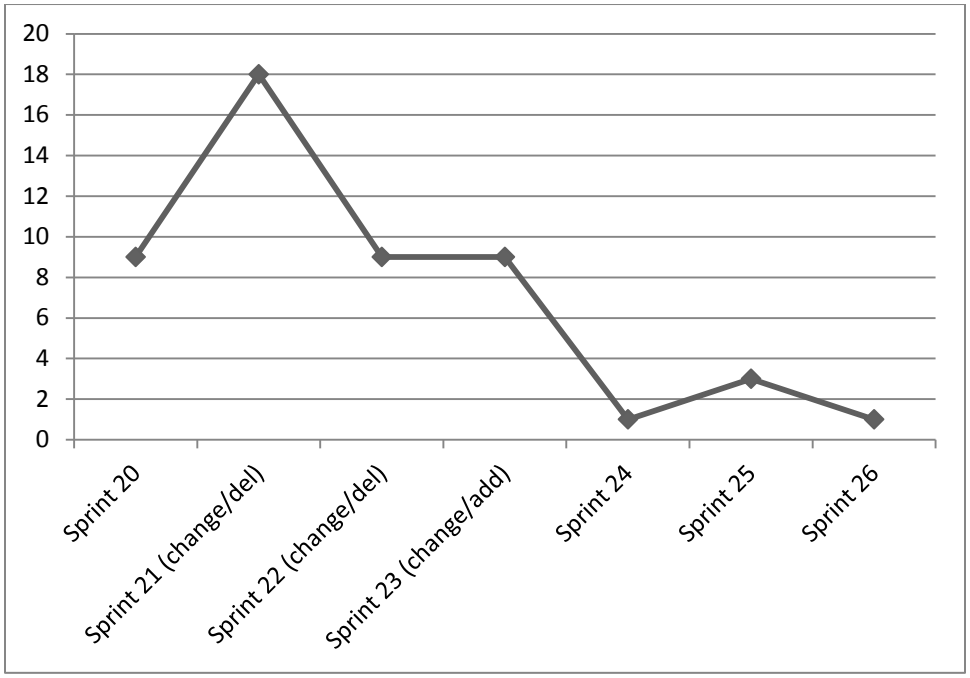

Figure 25: PRD concerning the introduction of the scrum board in team 8

Since the LOC could not be calculated in team 8 this is automatically considered negative in table 3. The amount of developed functionality increased after the reintroduction of the scrum board and was therefore affected in a positive manner. The amount of PRD decreased and thus the scrum board had a positive effect on the quality metric. Regarding the outcome of the three metrics one can conclude that the scrum board in team 8 showed *symptoms of improved productivity*.

Effect on productivity

Based on the measurements performed at three teams one can conclude that there was no positive effect on the technical metric (LOC). Functionality (user stories, tasks, and bug fixes) was affected positively at team 7 and team 8. In terms of quality (post release defects) there was a positive effect also at two of the three teams, namely team 4 and team 8. Since one team does not have an increase in developed functionality one cannot state that there are symptoms of improved productivity but rather that there are *slight symptoms of improved productivity* due to the fact that not all three teams had the same positive effect. The same scenario is present at the quality metric, although in another team order, and therefore can be concluded as *slight symptoms of improved quality*. One can conclude that the scrum board has the following effect: *slight symptoms of improved productivity, slight symptoms of improved quality*.

6.2.4 User story prioritization

Two of the five selected teams added the user story prioritization to their scrum process. Both team 2a and team 2b introduced their process changes in a parallel manner, the user story prioritization was added in sprint 26.

Team 2a / User story prioritization	Sprint 23	Sprint 24	Sprint 25	Sprint 26 (change)	Sprint 27	Sprint 28	Sprint 29
LOC addition	167	332	793	427	105	689	775
User stories	0	0	0	0	0	0	0
Tasks	0	0	0	1	9	15	7
Bug fixes	16	16	17	18	5	4	0

Version	Nr. of defects
PRD 2.7 (Nov 2011)	0
PRD 3.0 (July 2012)	4
PRD 3.0.6 (March 2013)	5

Table 23: Quantitative results from user story prioritization analysis of team 2a

One can notice that team 2a was primarily focused on bug fixing in the period of analysis. The analysis period spanned from 01-09-2012 till 15-12-2012.

LOC

The amount of additional LOC is at its highest in the sprint prior to the introduction of the user story prioritization. After the process change it drops by a factor 7.6 over two sprints, before increasing again by a factor 7.4 over the last two sprints. Although the LOC is higher in the last sprint than in the first sprint of analysis, there is too much fluctuation in the data in order to state a significant effect.

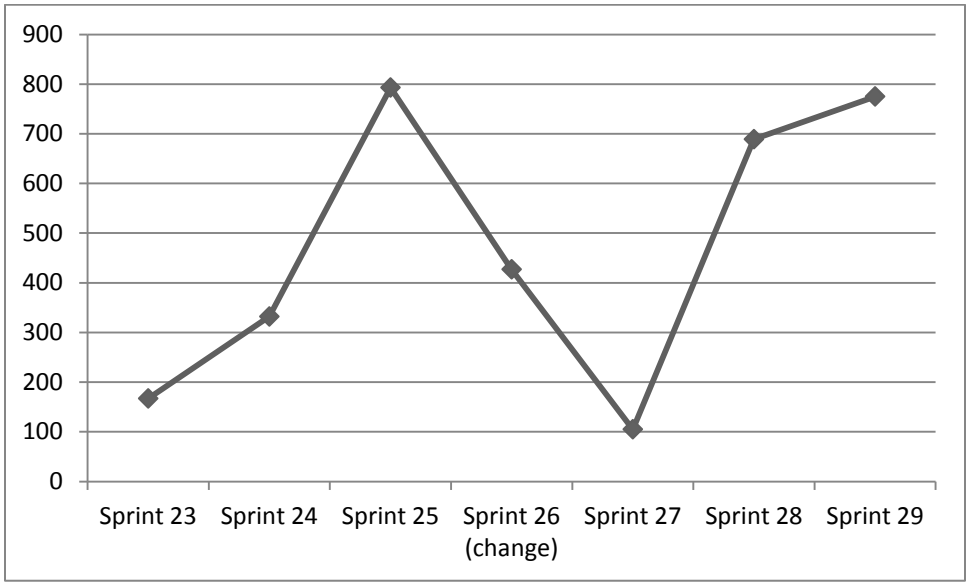

Figure 26: LOC concerning the introduction of user story prioritization in team 2a

Functionality

Only the amount of tasks went up over the analysed sprints, the amount of developed user stories and bug fixes decreased. The total amount of developed functionality has decreased since the introduction of user story prioritization. This leads one to conclude that it did not have a positive effect on the overall developed functionality.

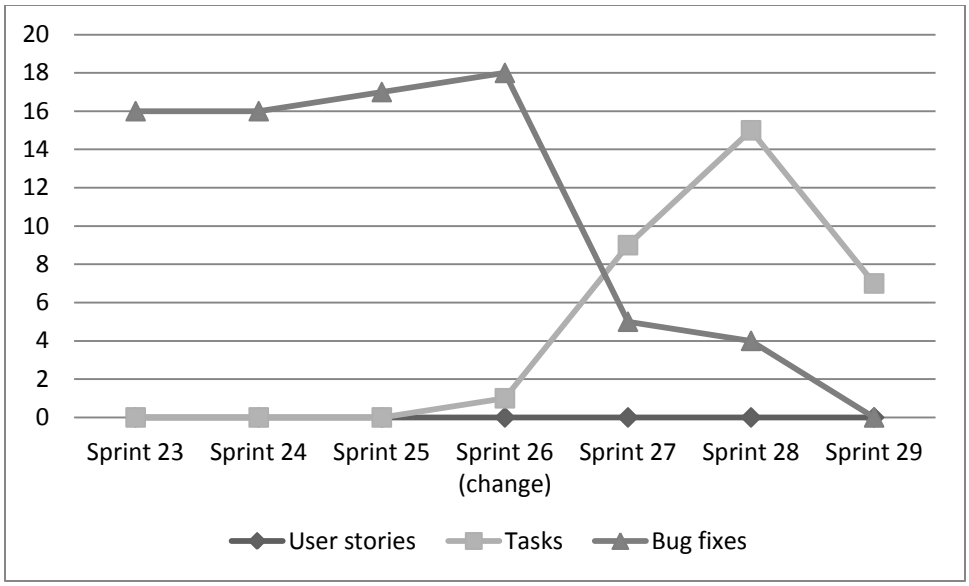

Figure 27: Functionality concerning the introduction of user story prioritization in team 2a

PRD

The releases that were analysed did not indicate an improvement in terms of a decreasing amount of post release defects. Instead, it increased slightly and therefore this method fragment did not have a positive effect on the software quality.

The three metrics were not affected in a positive way concerning the addition of user story prioritization. This results in *no symptoms of improved productivity*.

Team 2b / User story prioritization	Sprint 23	Sprint 24	Sprint 25	Sprint 26 (change)	Sprint 27	Sprint 28	Sprint 29
LOC addition	122	134	66	200	-2	46	38
User stories	0	0	0	0	0	0	0
Tasks	0	0	1	0	1	6	1
Bug fixes	25	0	13	5	5	6	4
PRD	No bugs reported in release notes since it was a new product which was not yet released publicly						

Table 24: Quantitative results from user story prioritization analysis of team 2b

In terms of activity, one can see that team 2b (in conjunction with team 2a) mostly concentrated on bug fixing instead of developing new functionality. The analysis period was the same as team 2a, from 01-09-2012 up to 15-12-2012. The null amounts of sprint 24 is due to the fact that there were only bug fixing tasks and the actual sprint and corresponding bug fixing work was not registered in visual studio.

LOC

There is an increase in LOC in the sprint where user story prioritization is added to scrum process. However, there is a decrease over the next three sprints. This also results in amounts lower than the amounts measured prior to the process change.

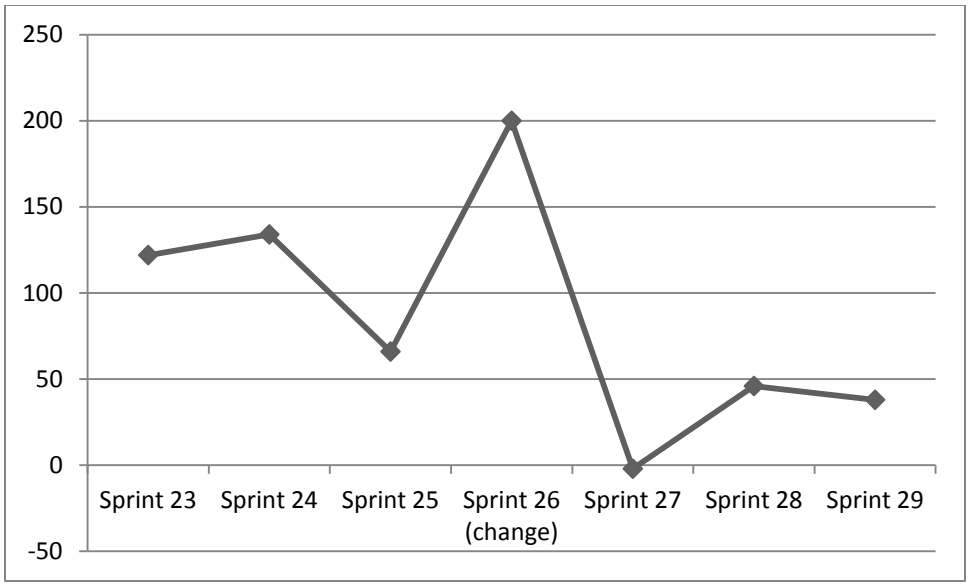

Figure 28: LOC concerning the introduction of user story prioritization in team 2b

Functionality

The amount of developed functionality was not affected in a positive manner after the introduction of user story prioritization. The total amount of developed functionality has decreased compared to the sprints prior to the introduction of user story prioritization.

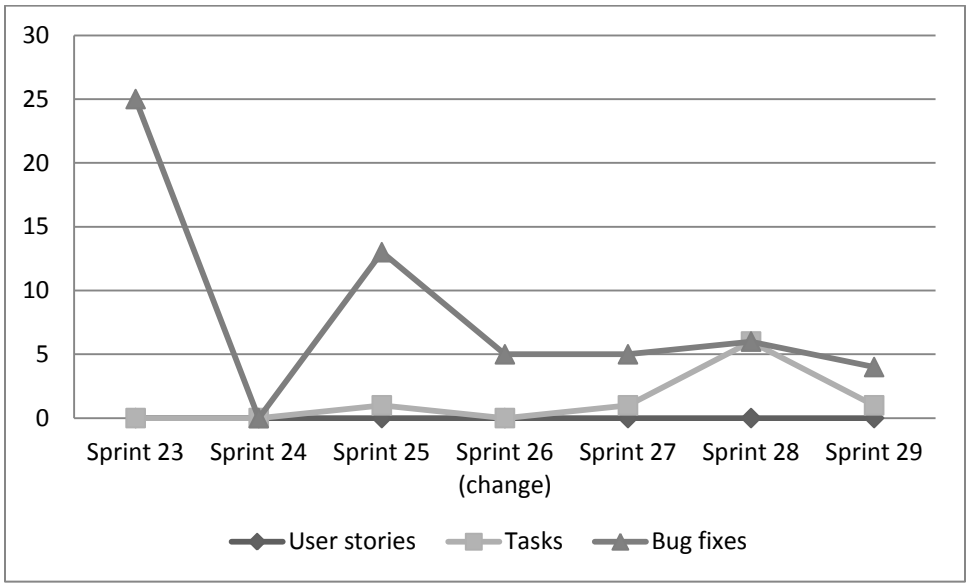

Figure 29: Functionality concerning the introduction of user story prioritization in team 2b

PRD

As mentioned earlier in this chapter, at the time of analysis, team 2b was working on a new product which was not yet released publicly. Therefore there was no data available regarding post release defects.

None of the three metrics were positively affected, resulting in *no symptoms of improved productivity*.

Effect on productivity

The results from both team 2a and team 2b are in conjunction. The data of both teams indicate that there are no positive effects on productivity when introducing user story prioritization. This allows one to state a sound conclusion over two data measurements in terms of productivity effect, there are *no symptoms of improved productivity* regarding the addition of user story prioritization.

6.2.5 Retrospective

The retrospective method fragment was introduced at two teams, namely team 2a and team 2b. Both teams introduced this method fragment in sprint 18.

Team 2a / Retrospective	Sprint 15	Sprint 16	Sprint 17	Sprint 18 (change)	Sprint 19	Sprint 20	Sprint 21
LOC addition	929	283	-407	-824	-421	245	-414
User stories	0	0	0	0	0	0	0
Tasks	0	0	0	0	0	1	0
Bug fixes	4	23	14	22	2	37	18

Version	Nr. of defects
PRD 2.7 (Nov 2011)	0
PRD 3.0 (July 2012)	4
PRD 3.0.6 (March 2013)	5

Table 25: Quantitative results from retrospective analysis of team 2a

The timeframe of analysis ranged from 01-05-2012 up to 15-08-2012. One can see from the table that the team had been bug fixing rather than developing new functionality.

LOC

The technical metric LOC is decreasing sharply for four sprints in concession. The amount of LOC increases in sprint 19, the first sprint after the process change, and also in sprint 20. Notably is that the variables of sprint 19 and 20 are almost equal to the two sprints prior to the process change, namely sprint 16 and 17. After two sprints of increasing LOC, a decrease can be noted in sprint 21. Despite the increase for two sprints after the process change, the LOC amounts stay slightly below the amounts of the measurements performed prior to the process change. The decrease in sprint 21 adds to the fact that the LOC was not positively affected.

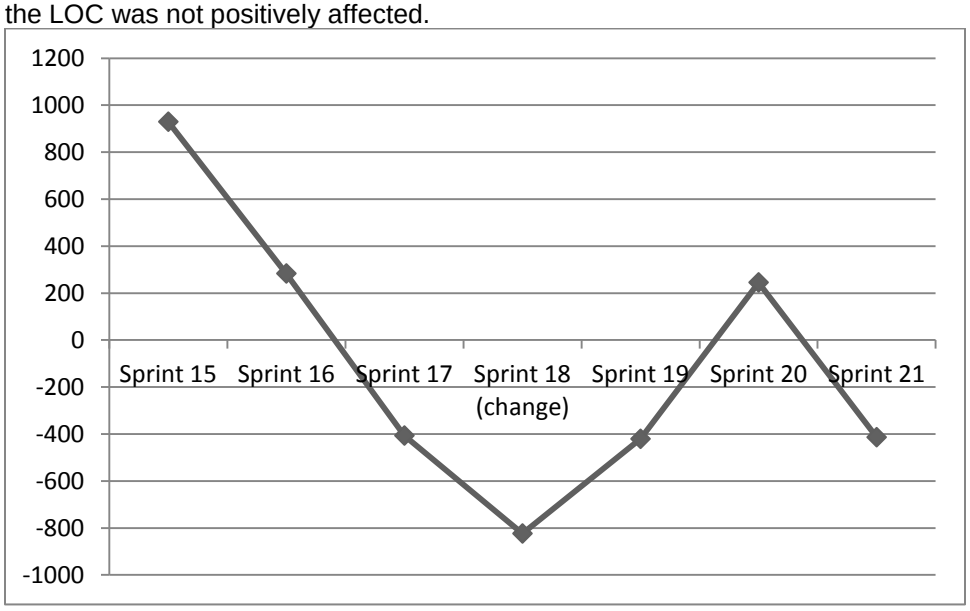

Figure 30: LOC concerning the introduction of the retrospective in team 2a

Functionality

The amount of user stories and tasks stay almost constant while the amount of bug fixes fluctuates. The amount of developed functionality goes up in the sprint in which the retrospective is added to the overall scrum process, yet it decreases again in the sprint thereafter. The peak in developed functionality can be seen in sprint 20 before decreasing to the amounts that were measured prior to the process change. In total there was more functionality developed after the introduction of the retrospective than there was prior to using this method fragment.

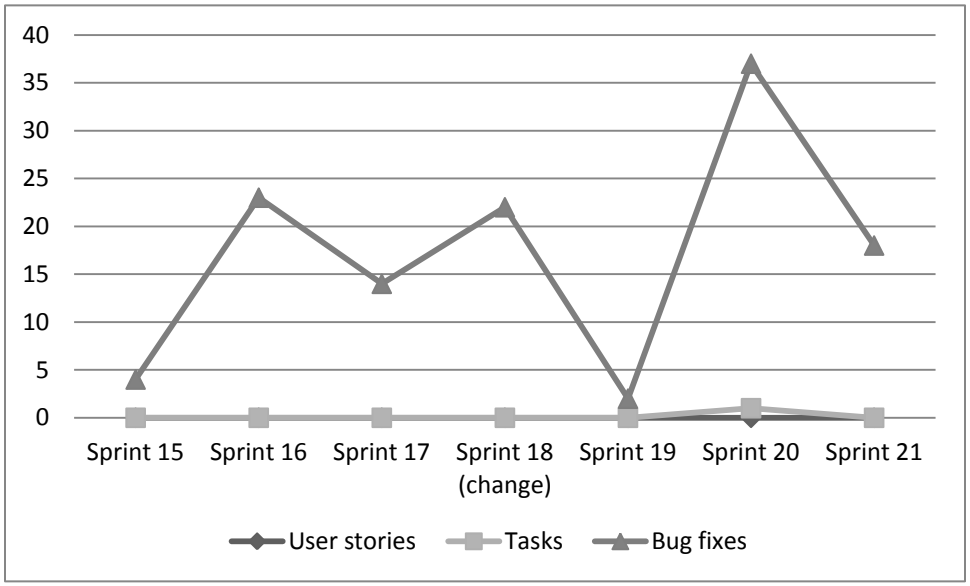

Figure 31: Functionality concerning the introduction of the retrospective in team 2a

PRD

The product versions that were released in the timeframe of analysis were the same as for some other method fragments of this team. One can see the amount of post release defects slightly increasing over three releases. The software quality was therefore not positively affected by the retrospective.

The retrospective resulted in a negative effect on the LOC, a positive effect on the overall developed functionality, and a negative effect on the quality. Using table 3, this method fragment resulted in *symptoms of improved productivity, quality needs to be improved*.

Team 2b / Retrospective	Sprint 15	Sprint 16	Sprint 17	Sprint 18 (change)	Sprint 19	Sprint 20	Sprint 21
LOC addition	-57	263	283	240	118	245	106
User stories	0	0	0	0	0	0	0
Tasks	26	2	10	3	0	0	0
Bug fixes	0	0	5	2	5	9	1
PRD	No bugs reported in release notes since it was a new product which was not yet released publicly						

Table 26: Quantitative results from retrospective analysis of team 2b

A notable fact is that the focus of team 2b switched from developing new functionality to bug fixing over the course of the analysed period. The period of analysis was from 01-05-2013 till 15-8-2013.

LOC

The LOC increases sharply with a factor 4.6 in sprint 16. The amount of LOC increases slightly in sprint 17, before decreasing slightly in sprint 18. After the introduction of the retrospective, the amount of LOC varies over three sprints and also stays below the LOC values measured prior to the process change.

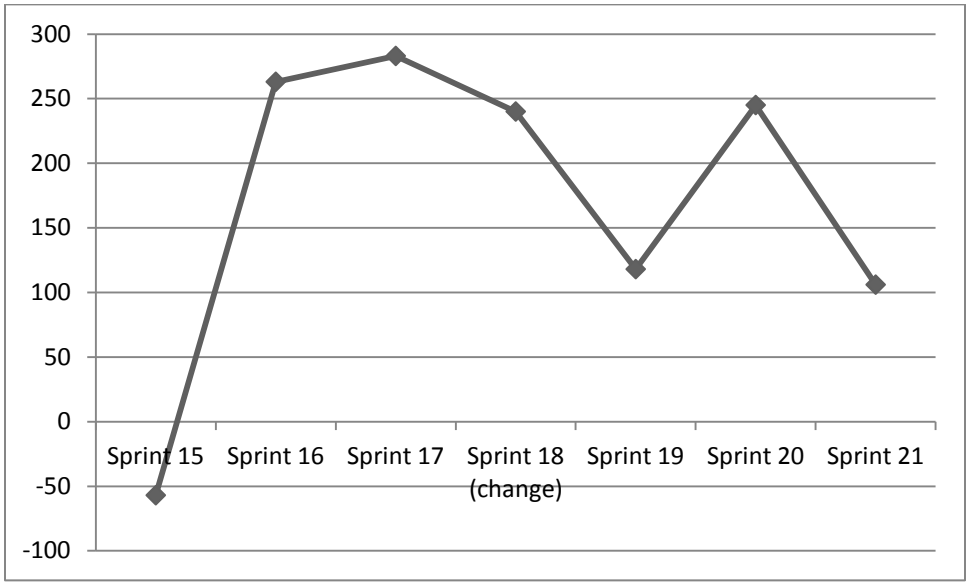

Figure 32: LOC concerning the introduction of the retrospective in team 2b

Functionality

The amount of developed tasks decreases in sprint 16 and increases in sprint 17. After sprint 17 the amount of developed tasks only decreases thus also after the introduction of the retrospective. The amount of bug fixes increases after the process changes but decreases again in the last measured sprint. The total developed functionality after the introduction of the retrospective is lower than the amount prior to this process change.

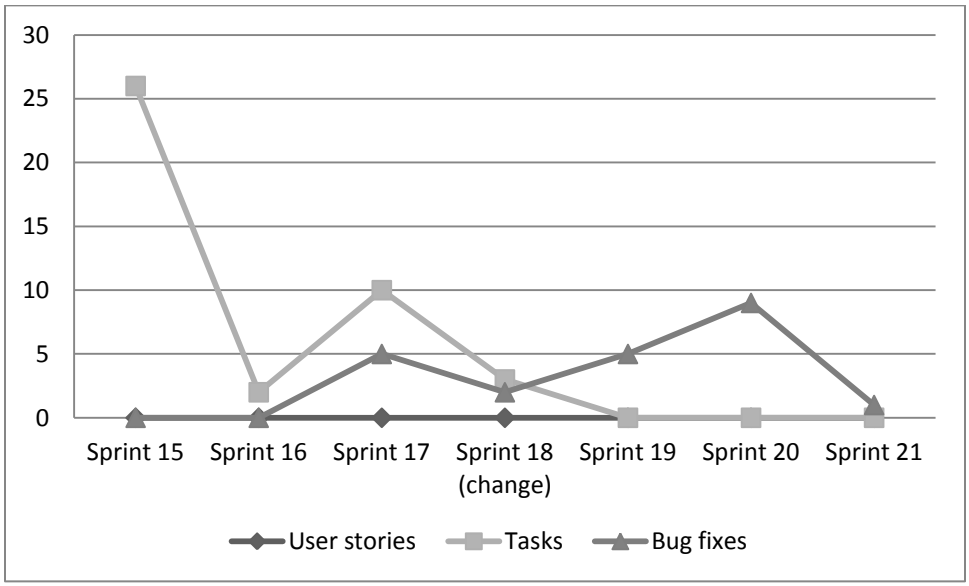

Figure 33: Functionality concerning the introduction of the retrospective in team 2b

PRD
No PRD could be measured within this team due to the product which was under development not being released publicly yet.

Considering the three measured metrics, one has to conclude that the retrospective had *no symptoms of improved productivity*.

Effect on productivity
The symptoms of improved productivity at team 2a were due to the developed functionality being positively affected. Since the same metric did not yield the same result at team 2b, the positive effect regarding developed functionality is neglected. The measurements of both teams in terms of LOC indicated that neither had a positive effect on productivity. The quality metric at team 2a was not affected in a positive manner and could not be established at team 2b. This leads one to conclude that the retrospective method fragment resulted in *no symptoms of improved productivity*.

6.2.6 Product backlog
The product backlog method fragment was added to the scrum process at three teams, namely team 2a, team 2b, and team 8. Team 2a and 2b added the method fragment in a parallel manner in sprint 22. Team 8 introduced the product backlog in sprint 8.

For teams 2a and 2b the product backlog was introduced parallel to the user stories method fragment. In order to save space and improve readability of this thesis, the measured data of this particular period will only be depicted at the user stories method fragment in paragraph 6.2.1. The conclusion per team was that team 2a had *slight symptoms of improved productivity, quality needs to be improved* and team 2b had *symptoms of improved productivity, quality needs to be improved*.

The data from team 8 will be described since the period of analysis in which team 8 introduced the product backlog has not been described yet.

Team 8 / Product backlog	Sprint 7	Sprint 8 (change)	Sprint 9	Sprint 10	Sprint 11	Sprint 12	Sprint 13
LOC addition	N/A	N/A	N/A	N/A	N/A	N/A	N/A
User stories	10	1	6	8	11	11	5
Tasks	29	6	24	21	33	35	31
Bug fixes	0	0	0	0	2	1	13
PRD	16	7	3	9	12	9	4

Table 27: Quantitative results from product backlog analysis of team 8

Data prior to sprint 7 was not available for analysis, therefor the choice was made to opt for a longer analysis period after the actual process change. This was done in order to have the same amount of sprints as the other analysed method fragments have. As one can see from table 27, the team increased its bug fixing in the latter sprints next to regular functionality development. The timeframe of analysis was from 01-11-2011 to 31-05-2012.

Functionality

The amount of developed tasks is decreasing prior to the introduction of the product backlog before increasing accompanied with slight decreases over the next five sprints. Overall, it increases by a factor 5.2. The amount of user stories also increase steadily after the process change, yet a decrease occurs in the last sprint of analysis. Bug fixes increase in the last three sprints of analysis.

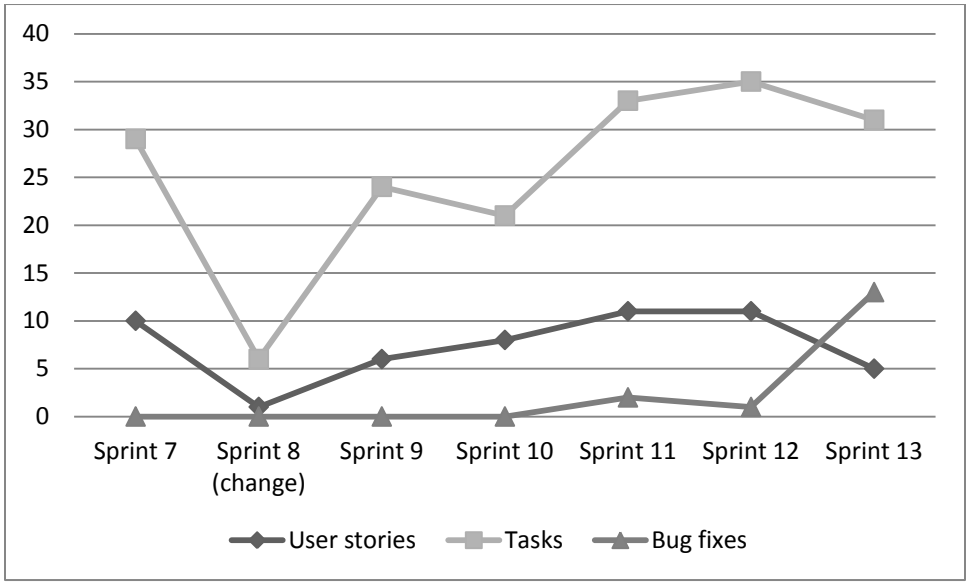

Figure 34: Functionality concerning the introduction of the product backlog in team 8

PRD

The measured PRD is the same as that of the pair programming method fragment since the product backlog was introduced one sprint later than pair programming. It therefore depicts the same (PRD decreasing) graph.

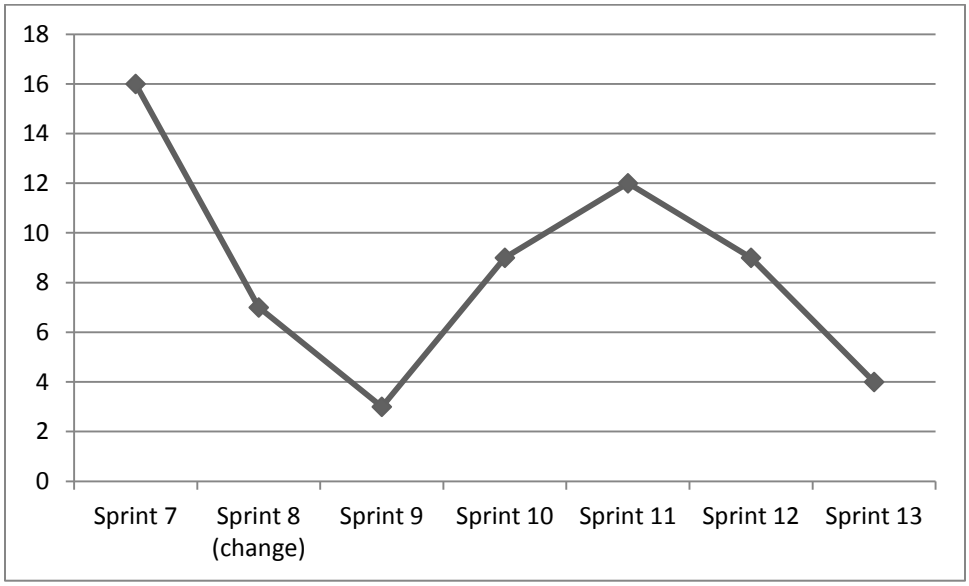

Figure 35: PRD concerning the introduction of the product backlog in team 8

The addition of the product backlog resulted in an increase in developed functionality and in a decrease of PRD. Since both metrics indicate a positive effect on productivity one can conclude that the product backlog had *symptoms of improved productivity*.

Effect on productivity

The product backlog was introduced parallel to the user stories method fragment at team 2a and team 2b. For the user stories method fragment it was concluded that there was a contradiction whether the positive effect was due to the increased LOC or the increased developed functionality. The data from team 8 confirms the increased functionality of team 2b. Due to team 2a not having the same effect, an increase in developed functionality, one can conclude that the increase in developed functionality at team 2b was not due to the introduction of user stories but rather due to the introduction of the product backlog. Since team 2a does not have an increase in developed functionality one has to conclude that the product backlog resulted in *slight symptoms of improved productivity, quality needs to be improved*. The quality metric is contradicting at team 2a and team 8 and therefore no sound conclusion can be given in terms of the PRD. This means that it is automatically considered as negative in table 3.

6.2.7 Sprint backlog

The sprint backlog method fragment was added at two teams, namely team 4 and team 8. Team 4 started using a sprint backlog in sprint 13, team 8 introduced it in sprint 8. Because team 8 introduced it in sprint 8, the data will not be repeated in this paragraph since data of sprint 8 is already depicted at the product backlog method fragment (paragraph 6.2.6). Team 4 partly uses data of the scrum board since it was added one sprint prior to the introduction of the scrum board method fragment.

Team 4 / Sprint backlog	Sprint 9	Sprint 10	Sprint 11	Sprint 12 (change)	Sprint 13	Sprint 14	Sprint 15
LOC addition	4483	2108	578	4245	1976	2070	9
User stories	3	3	6	2	4	3	5
Tasks	4	10	13	21	0	0	3
Bug fixes	1	1	4	13	6	8	8

Version	Nr. of defects
PRD 3.6.0 (July 2012)	7
PRD 3.6.1 (Oct 2012)	6
PRD 3.6.2 (Feb 2013)	1

Table 28: Quantitative results from sprint backlog analysis of team 4

The analysis period concerning the sprint backlog was from 07-08-2012 to 23-01-2013.

LOC

The LOC amount declines up to sprint 11 by a factor 7.8, before increasing sharply in the sprint where the sprint backlog is introduced by a factor 7.3. After the process change has occurred, the amount of LOC decreases over the next three sprints to end at the lowest amount of LOC measured of the seven sprints in total.

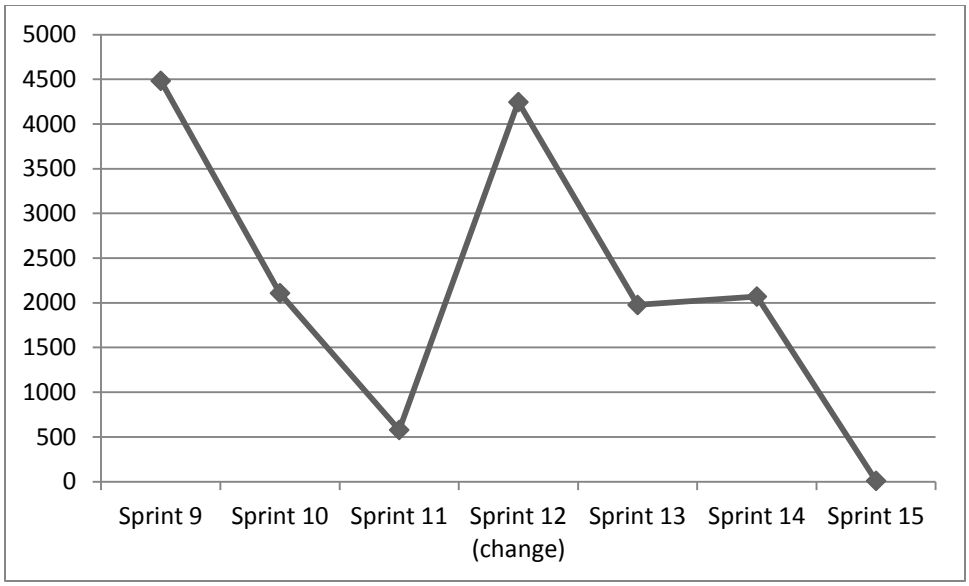

Figure 36: LOC concerning the introduction of the sprint backlog in team 4

Functionality

The tasks and bug fixes peak in the sprint where the sprint backlog is introduced and were also increasing steadily over the sprints prior to the process change. The user stories are at its lowest amount in the sprint where the process change occurs. Sprint 13 notes a decrease regarding the amount of tasks and bug fixes, while the user stories increase slightly. Over the next two sprints the amount of developed functionality increases slightly but the total amount is lower compared to the sprints prior to the process change.

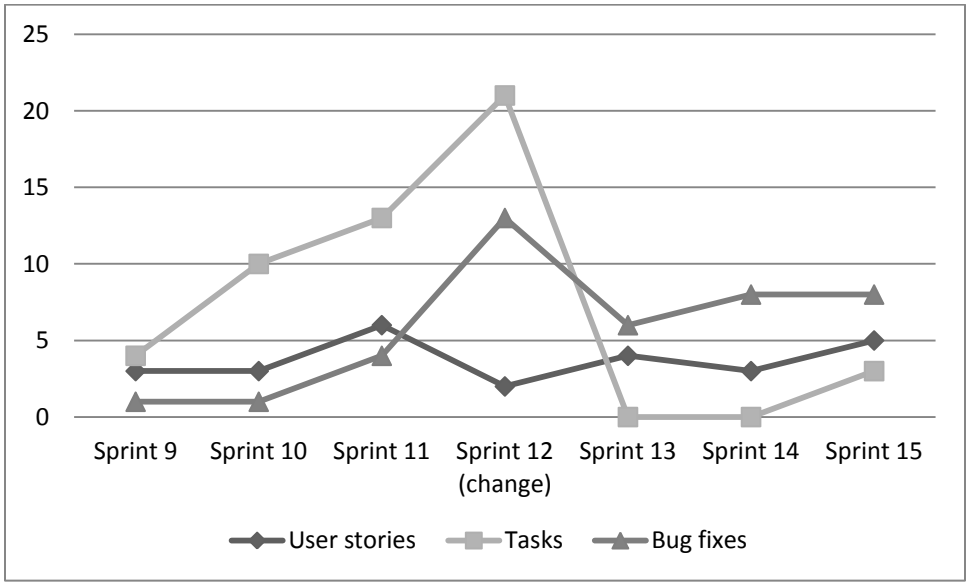

Figure 37: Functionality concerning the introduction of the sprint backlog in team 4

PRD

The amount of post release defects is declining over three releases which indicates that there is a positive effect on the software quality regarding the introduction of the sprint backlog. The decline in PRD goes from seven in version 3.6.0 to one in version 3.6.2.

The three metrics of team 4 result in only the quality metric being positively influenced. Using table 3, this means that the sprint backlog for team 4 had *no symptoms of improved productivity, quality has improved.*

Effect on productivity

The data of team 8 indicated that the developed functionality had increased and the PRD decreased, resulting in *symptoms of improved productivity*. The developed functionality metric contradicts with the measurements at team 4, so the functionality effect is discarded because of the contradiction between the two teams. Both teams have equal results in terms of the quality metric, measurements performed at both teams indicated that the amount of PRD went down over the period of analysis. This means that the sprint backlog resulted in *no symptoms of improved productivity, quality has improved*.

Considering the parallel introduction with the product backlog, the sprint backlog was the method fragment that had the positive effect on the PRD since one can see the same effect at team 4 and is therefore in conjunction with the data of team 8.

6.2.8 Continuous integration

The continuous integration method fragment was introduced at team 4 and team 8. Team 4 introduced this method fragment in sprint 13 while team 8 introduced it in sprint 15. Continuous integration was added parallel to the scrum board at team 4 and therefor has the same data. By not repeating the exact same data, space is saved and readability of this thesis is improved.

Team 8 / Continuous integration	Sprint 12	Sprint 13	Sprint 14	Sprint 15 (change)	Sprint 16	Sprint 17	Sprint 18
LOC addition	N/A	N/A	N/A	N/A	N/A	N/A	N/A
User stories	11	5	4	0	6	8	4
Tasks	35	31	28	12	22	32	18
Bug fixes	1	13	16	47	13	10	14
PRD	9	4	4	5	8	2	7

Table 29: Quantitative results from continuous integration analysis of team 8

The period of analysis ranged from 02-04-2012 till 30-11-2012.

Functionality

The amount of bug fixes increase in the sprint in which continuous integration is introduced by a factor 2.9, before decreasing in sprint 16 by a factor 3.6. It further declines in sprint 17 before slightly increasing in the last sprint. The amount of tasks is declining prior to and also during the process change. After the introduction of continuous integration the amount of tasks increase over two sprints and decrease in the last sprint. The user stories stay relatively stable, decreasing prior to the process change and slightly increasing after the process change. Overall the developed functionality is lower after the addition of continuous integration than before this process change.

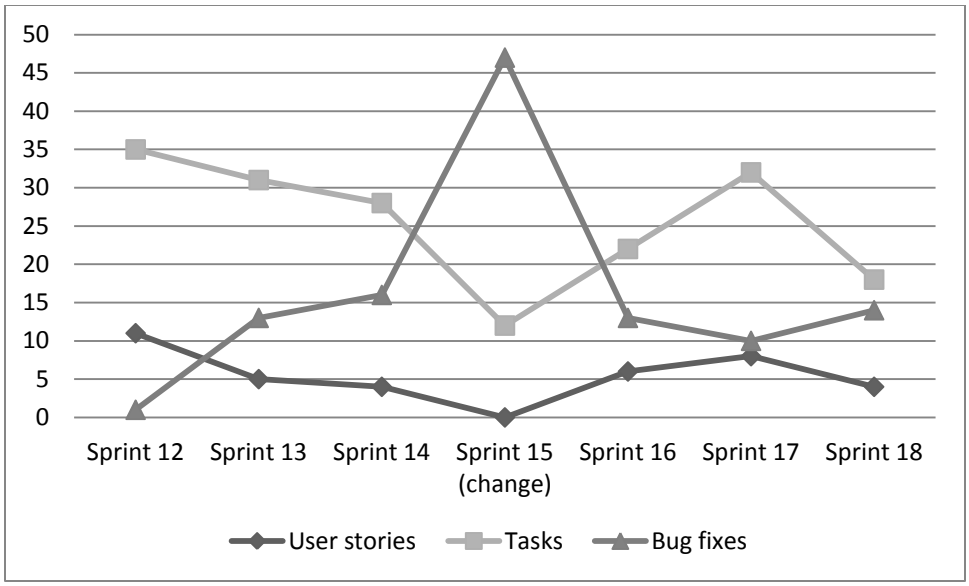

Figure 38: Functionality concerning the introduction of continuous integration in team 8

PRD

One can note a decreasing trend in PRD prior to the process change, and fluctuating results after the process change. The PRD is neutral since the amount of defects are exactly the same before the introduction of continuous integration as well as after it was introduced. However, neutral means that there was no positive effect (see also table 3).

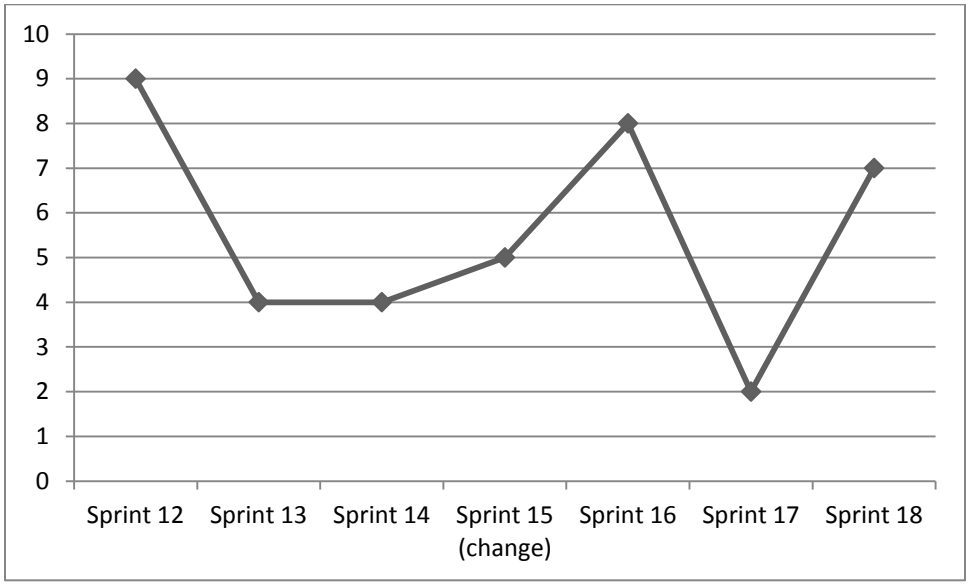

Figure 39: PRD concerning the introduction of continuous integration in team 8

Continuous integration at team 8 did not influence the developed functionality in a positive manner. The amount of PRD had a neutral result and thus was not positively affected as well. One can conclude that continuous integration at team 8 had *no symptoms of improved productivity*.

Effect on productivity

Both team 4 and team 8 did not yield a positive effect on developed functionality. Only team 4 registered a decrease in PRD, yet this result was not accomplished at team 8. The analysis of continuous integration resulted in *no symptoms of improved productivity*.

6.2.9 Burn down chart

Three teams introduced the burn down chart into their scrum process. Team 2a and team 2b deleted the burn down chart in sprint 12, team 8 added it in sprint 8. This means that the data for team 8 will be the same data as that of the product backlog. The data of team 8 will therefore not be repeated in this paragraph.

Team 2a / Burn down chart	Sprint 9	Sprint 10	Sprint 11	Sprint 12 (change)	Sprint 13	Sprint 14	Sprint 15
LOC addition	733	592	921	1797	351	1508	929
User stories	0	0	0	0	0	0	0
Tasks	16	16	31	3	10	2	0
Bug fixes	0	0	0	8	6	11	4

Version	Nr. of defects
PRD 2.7 (Nov 2011)	0
PRD 3.0 (July 2012)	4
PRD 3.0.6 (March 2013)	5

Table 30: Quantitative results from burn down chart analysis of team 2a

From table 30 one can see the tasks amount decreasing over the several sprints while the amount of bug fixes increase. The analysis period spanned from 01-02-2012 to 15-05-2012.

LOC

The LOC increases steadily up to sprint 12. After the addition of the burn down chart the data fluctuates and as a result the graph is peaky. Due to this fluctuation, no sound conclusion can be given regarding a possible effect on productivity.

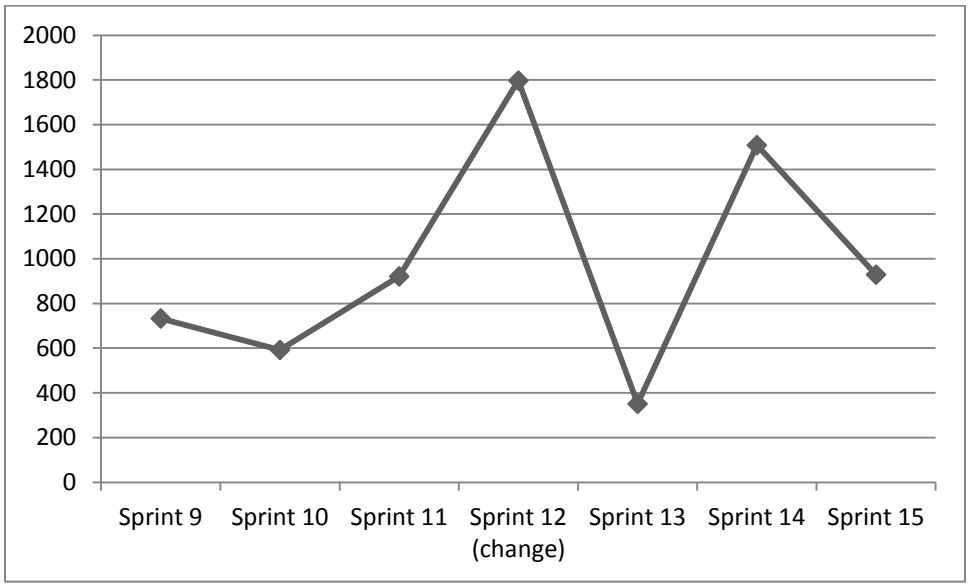

Figure 40: LOC concerning the introduction of the burn down chart in team 2a

Functionality

In terms of developed functionality one can see the amount of bug fixes increasing slightly after the introduction of the burn down chart. The amount of tasks decreases by a factor 10.3 in the sprint in which the process change occurs. Sprint 13 is the only sprint in which an increase in the amount of tasks can be registered, sprint 14 and 15 record a decrease up to the lowest amount of all sprints. The overall developed functionality decreases after the process change.

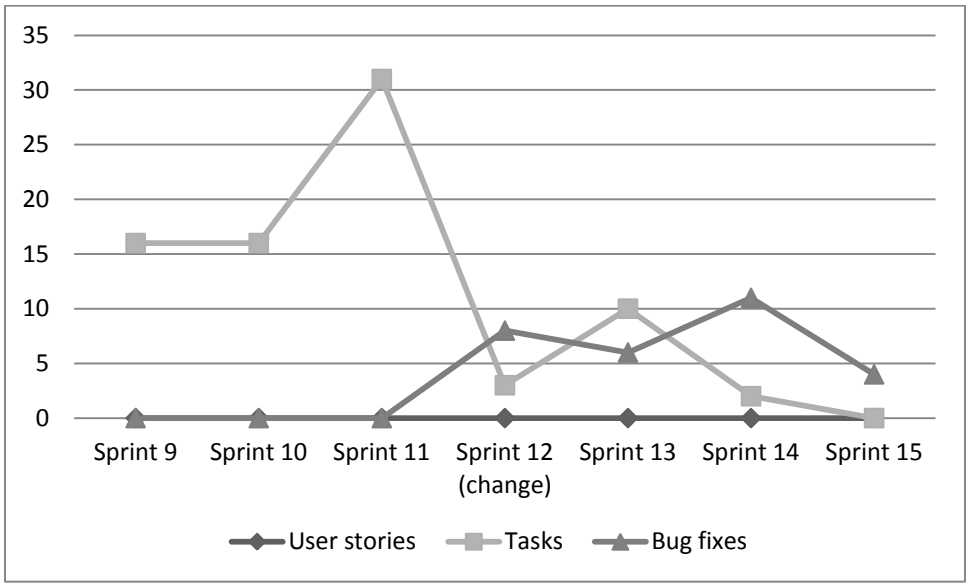

Figure 41: Functionality concerning the introduction of the burn down chart in team 2a

PRD

The release versions that were applicable for measuring the PRD metric indicated that the amount of PRD rose slightly and as a result the software quality was not positively affected.
The data of team 2a results in no positive effect on either the LOC, the developed functionality, or the amount of PRD. This means that there are *no symptoms of improved productivity*.

Team 2b / Burn down chart	Sprint 9	Sprint 10	Sprint 11	Sprint 12 (change)	Sprint 13	Sprint 14	Sprint 15
LOC addition	468	1753	4914	1124	371	453	-57
User stories	0	0	0	0	0	0	0
Tasks	5	2	11	4	1	0	26
Bug fixes	4	0	0	1	14	1	0
PRD	No bugs reported in release notes since it was a new product which was not yet released publicly						

Table 31: Quantitative results from burn down chart analysis of team 2b

The timeframe of analysis was from 01-02-2012 till 15-05-2012.

LOC

The highest amount of additional LOC is measured in sprint 11, after this sprint there is a sharp decline by a factor 4.4 in the sprint where the process change occurred. The LOC amount keeps declining in the sprints after the process change and therefore did not result in a positive effect.

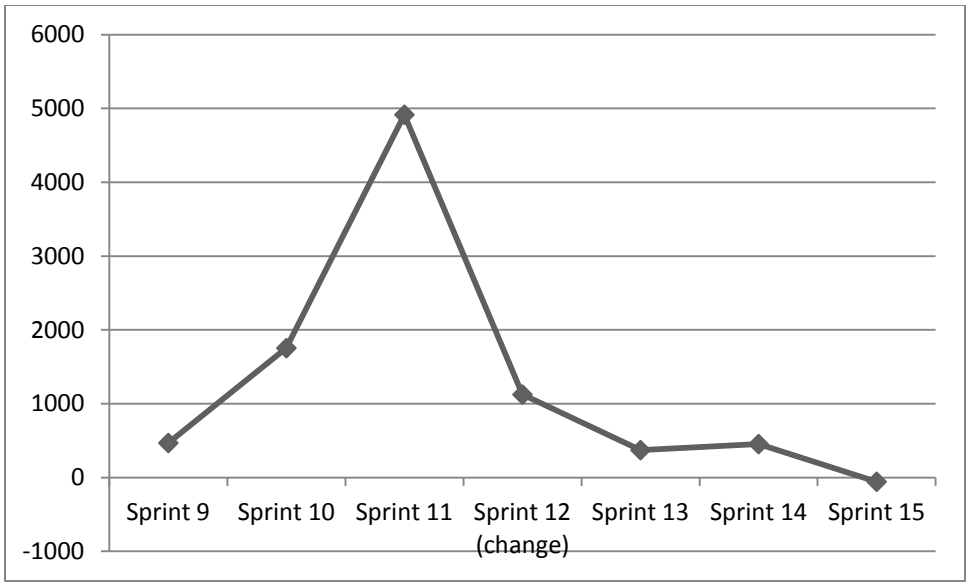

Figure 42: LOC concerning the introduction of the burn down chart in team 2b

Functionality

The amount of developed functionality is peaking after the introduction of the burn down chart in terms of the tasks and bug fixes. Despite the increasing and decreasing variables, the total amount of developed functionality is higher after the process than prior to it.

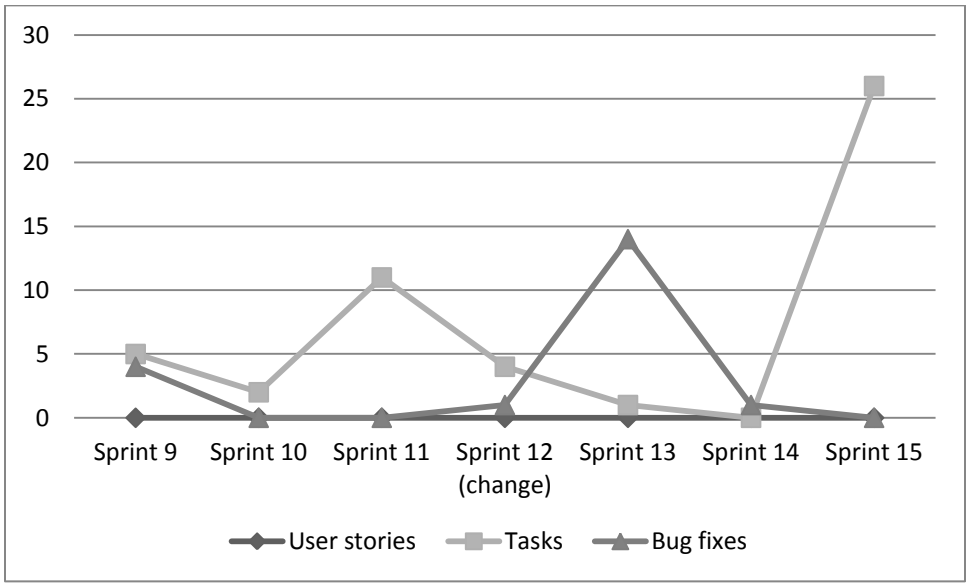

Figure 43: Functionality concerning the introduction of the burn down chart in team 2b

The introduction of the burn down chart at team 2b resulted in a positive effect on the developed functionality. The LOC was not positively influenced by the addition of the burn down chart, the PRD could not be measured. Due to the positive influence on the developed functionality the burn down chart at team 2b resulted in *symptoms of improved productivity, quality needs to be improved*. Since the PRD metric could not be measured it is automatically considered negative in table 3. This is the cause of why the concluding effect states that the quality needs to be improved.

Team 8 / Burn down chart	Sprint 7	Sprint 8 (change)	Sprint 9	Sprint 10	Sprint 11	Sprint 12	Sprint 13
LOC addition	N/A	N/A	N/A	N/A	N/A	N/A	N/A
User stories	10	1	6	8	11	11	5
Tasks	29	6	24	21	33	35	31
Bug fixes	0	0	0	0	2	1	13
PRD	16	7	3	9	12	9	4

Table 32: Quantitative results from burn down chart analysis of team 8

The data of team 8 regarding the burn down chart addition was the same as that of the product backlog method fragment of this. This was due to the fact that these two method fragments were introduced parallel to each other. The data, description, and explanation will not be repeated in this paragraph to improve readability and to save space in the main thesis.

The effect of the introduction of the burn down chart was that the developed functionality increased after the process change. A similar positive effect can be noted concerning the quality metric, the amount of PRD decreased. The measurements at team 8 indicated that there were *symptoms of improved productivity*.

Effect on productivity

The data measured at the three teams which deleted and added the burn down chart to its scrum process resulted in various statements. The deletion of the burn down chart at team 2a indicated that the developed functionality decreased, team 8 confirmed this effect by means of an increase in developed functionality after it added the burn down chart. Team 2b did not have a similar effect as it had an increase in functionality after it deleted the burn down chart from its scrum process. The other two metrics, LOC and PRD, were contradicting at the two teams at which it was measured.
Since the developed functionality was positively influenced at two of the three teams one can conclude that the burn down chart resulted in *slight symptoms of improved productivity, quality needs to be improved*.

The burn down chart was added parallel to the product- and sprint backlog in sprint 8. However, the effect of the burn down chart does not have similarities with these two method fragments because of the burn down chart data of team 2a and 2b. As a result, the burn down chart influence on productivity stands on its own.

6.2.10 Demo

The demo method fragment was only introduced at team 7. The team added the demo in sprint 40.

Team 7 / Demo	Sprint 37	Sprint 38	Sprint 39	Sprint 40 (change)	Sprint 41	Sprint 42	Sprint 43
LOC addition	3938	732	-352	-1649	-102	1004	2506
User stories	0	0	2	1	3	0	0
Tasks	3	6	5	7	7	4	2
Bug fixes	3	6	0	0	1	26	4

Version	Nr. of defects
PRD 2.4 (Sept 2010)	27
PRD 3.1 (May 2011)	40
PRD 3.2 (Apr 2012)	20

Table 33: Quantitative results from demo analysis of team 7

The timeframe of analysis ranged from 07-03-2011 till 29-07-2011.

LOC

The amount of LOC is decreasing up to sprint 40. The first increase can be noted one sprint after the process change. The next two sprints also record an increase in the amount of LOC. One can see that all sprints after the process change lead to a (linear) increase in terms of LOC.

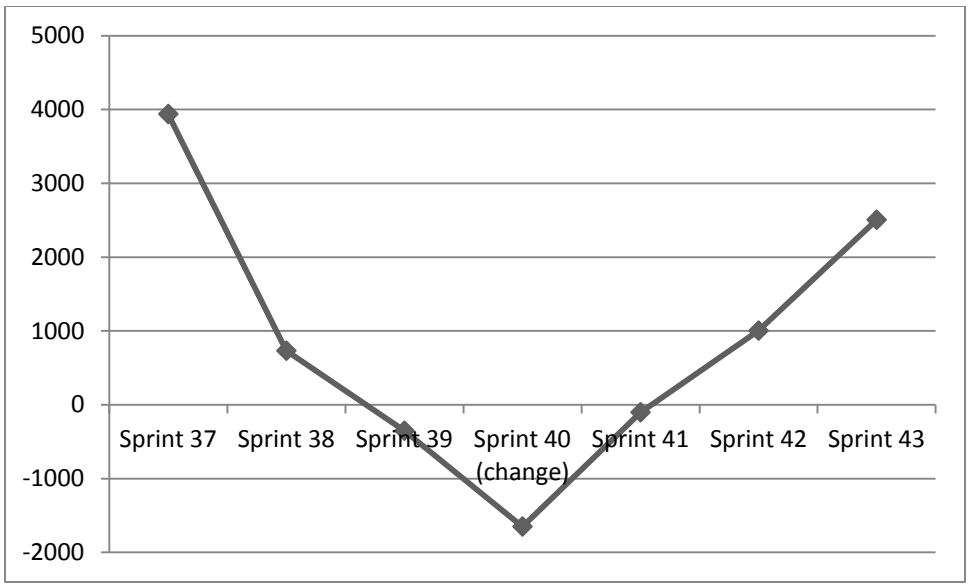

Figure 44: LOC concerning the introduction of the demo in team 7

Functionality

In terms of developed functionality one can notice the amount of tasks at its highest after the process change, stabilizing for two successive sprints, and decreasing over the last two sprints. The same scenario occurs regarding the user stories, the highest amount is reached in sprint 41 before decreasing the sprint thereafter. Bug fixes are stable in sprint 40 and 41, before peaking in sprint 42 and decreasing again in sprint 43. The total amount of developed functionality was higher after the introduction of the demo than in the sprints prior to it.

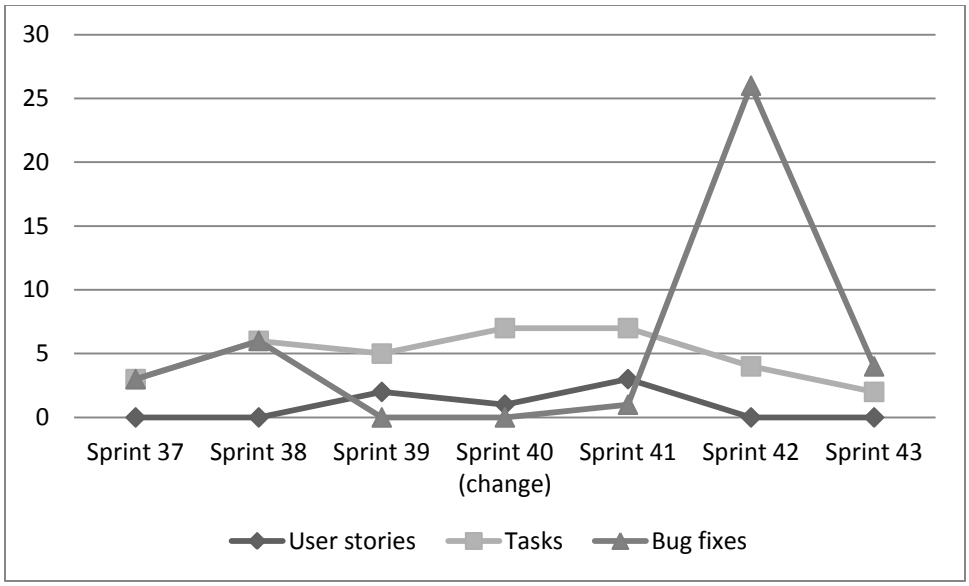

Figure 45: Functionality concerning the introduction of the burn down chart in team 2b

PRD

The amount of PRD increased in the software version which was released to 40 defects, the next version registers a decrease to 20 defects. The fluctuation in the measured PRD data disallows one to conclude a sound concluding effect.

Effect on productivity

The technical and functional metric were positively influenced by the introduction of the demo method fragment. Not all three metrics were influenced in a positive manner since the PRD had too much fluctuation. Using table 3, one can conclude that the introduction of the demo resulted in *symptoms of improved productivity, quality needs to be improved*.

6.2.11 Acceptance test

The acceptance test was introduced at four teams. Team 2a, team 2b, team 4, and team 8 were the teams that opted for an introduction of this particular method fragment. The data of all four teams have already been described and discussed since in every team the situation arose that the acceptance test method fragment was introduced parallel to at least one other method fragment.

Team 2a and team 2b added the acceptance test parallel to the user story prioritization method fragment in sprint 26. The result was that there were *no symptoms of improved productivity*.

Team 4 introduced the acceptance test parallel to continuous integration in sprint 13, resulting in *no symptoms of improved productivity, quality has improved*.

The acceptance test was also added simultaneously with continuous integration at team 8, namely in sprint 15. The effect could be concluded as *no symptoms of improved productivity*.

Only team 4 had a positive effect in the form of a decrease in PRD, the other three teams did not record a positive effect on productivity. The decrease in PRD of team 4 is neglected by the data of the other teams, the acceptance test therefore results in *no symptoms of improved productivity*.

6.2.12 Sprints

Team 4 was the team which had measurable data regarding a change in its sprints. Team 4 performed the change in sprint length in sprint number 10.

Team 4 / Sprints	Sprint 7	Sprint 9	Sprint 10 (change)	Sprint 11	Sprint 12	Sprint 13	Sprint 14
LOC addition	3313	4483	2108	578	4245	1976	2053
User stories	2	3	3	6	2	4	3
Tasks	13	4	10	13	21	0	0
Bug fixes	0	1	1	4	13	6	8

Version	Nr. of defects
PRD 3.6.1 (Oct 2012)	7
PRD 3.6.2 (Feb 2013)	6
PRD 3.6.3 (Apr 2013)	1

Table 34: Quantitative results from sprint analysis of team 4

Sprint 8 was neglected in the data measurements since this sprint was an extra one weekly sprint after sprint 7 and was therefore too short in terms of sprint length to be analysed. The decision was made to add another sprint after the process change to match the amount of analysed sprints of other method fragments. The period of analysis ranged from 10-7-2012 till 22-12-2012.

LOC

The LOC peaks in the sprint prior to the process change and decreases up to one sprint after the sprint change by a factor 7.8. Sprint 12 records a sharp increase before decreasing again in sprint 13. The amount of LOC stabilizes in sprint 14. A fluctuation of LOC can be seen from the first sprint to the last sprint.

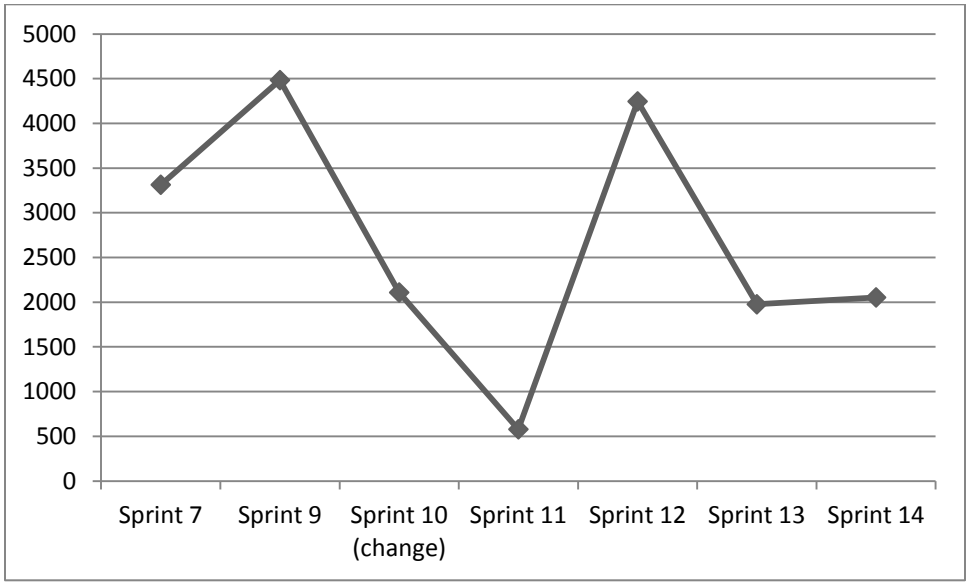

Figure 46: LOC concerning the introduction of sprints in team 4

Functionality

The amount of tasks increases steadily after the sprint change up to sprint 12. One can see the same scenario in the amount of bug fixes although the amount in the last two sprints is higher than the amount of tasks. The user stories fluctuate after the process change. The total amount of functionality is increasing after the sprint change.

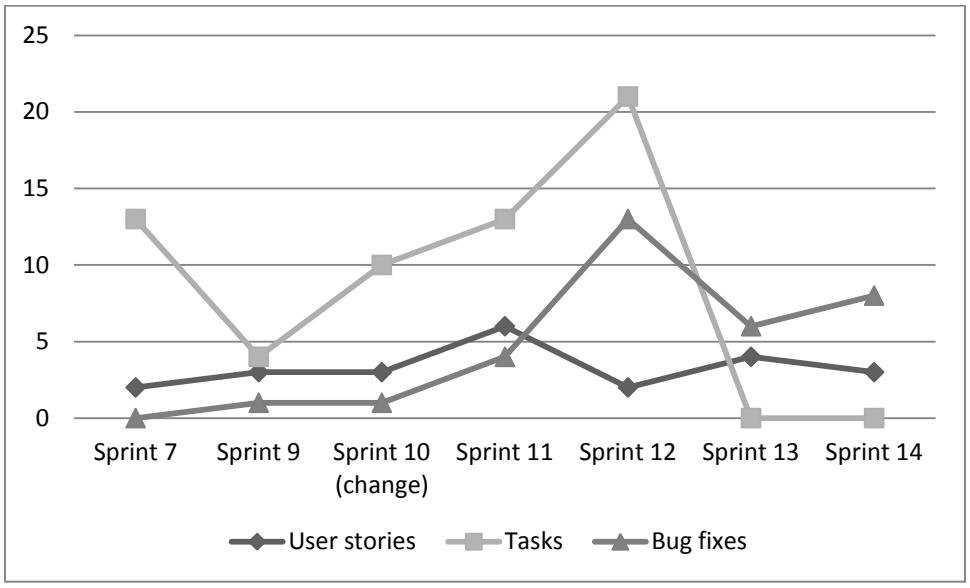

Figure 47: Functionality concerning the introduction of sprints in team 4

PRD

The amount of post release defects decrease over three sprints. The sprint method fragment uses the same software versions as for instance the sprint backlog, acceptance test, and continuous integration. The amount of defects decreased over three releases from seven to one.

Effect on productivity

The LOC was considered negative due to the fluctuation. Both the developed functionality and the PRD were influenced in a positive manner. These measurements conclude that sprints result in *symptoms of improved productivity*.

6.2.13 Self-contained team

The self-contained team method fragment was only introduced at team 4. This method fragment was introduced parallel to the method fragments that were introduced in sprint 13, such as the acceptance test. The analysis period spanned from 06-09-2012 to 19-02-2013.

The data of team 4 indicated that there were *no symptoms of improved productivity, quality has improved*. One can notice, since this method fragment is parallel introduced to the scrum board method fragment, that it has the same effect as the scrum board method fragment. The quality enhancement however was due to the scrum board since this effect was confirmed by the data of team 8 and there was no additional data at other teams to confirm the effect at the self-contained team method fragment.

6.2.14 Concluding effect summary

Various method fragments were introduced in a parallel manner. Some of these 'groups' of method fragments could not be separated in terms of dedicating an effect to a single method fragment within a group. Fortunately, the groups of method fragments that could not be split up in terms of its influence on productivity were all not influencing productivity in a positive way such as the acceptance test method fragment. For some groups of method fragments that were introduced simultaneously it was possible to dedicate a positive effect on productivity to a single method fragment, such as the scrum board, the product backlog, and the sprint backlog.

Dedicating a positive effect on productivity to one method fragment is beneficial for formulating advice to organisations since it is more specific than dedicating it to a pairing or a combined group of method fragments. A remark is that the dedication of an effect to a single method fragment was mostly due to one team confirming the data and measurements of two other teams at another method fragment. An example of this is the parallel introduction of user stories and the product backlog, with the data from team 8 of the product backlog confirming the functionality increase of team 2b. This research was scoped and therefor the decision was made to opt for a selection of teams for analysis purposes. With more teams than the five selected teams one could prove the effect even more. An option for organisations which would like to implement only method fragments that positive influence productivity would be to implement the actual pairing or group of method fragment instead of the single method that was found to have an individual effect. Due to the fact that in some parallel introduction situations only one other team confirms the productivity effect, organisations might opt for the safest option and implement the pairing or group of method fragments. More research similar to this research, and with a broader selection of teams, would enable more proof and evidence that an effect is due to a single method fragment instead of a pairing or group of method fragments.

Method fragment	Proven effect on productivity
User stories	*No symptoms of improved productivity*
Pair programming	*No symptoms of improved productivity, quality has improved*
Scrum board	*Slight symptoms of improved productivity, slight symptoms of improved quality*
User story prioritization	*No symptoms of improved productivity*
Retrospective	*No symptoms of improved productivity*
Product backlog	*Slight symptoms of improved productivity, quality needs to be improved*
Sprint backlog	*No symptoms of improved productivity, quality has improved*
Continuous integration	*No symptoms of improved productivity*
Burn down chart	*Slight symptoms of improved productivity, quality has improved*
Demo	*Symptoms of improved productivity, quality needs to be improved*
Acceptance test	*No symptoms of improved productivity*
Sprints	*Symptoms of improved productivity*
Self-contained teams	*No symptoms of improved productivity, quality has improved*

Table 35: Summary of the proven effects of the various analysed method fragments

Table 35 summarizes the effects of the various method fragments on productivity. One can see that out of the total of 13 analysed method fragments, 8 were found to have an effect. 5 of those 8 method fragments had an effect on actual productivity improvement, 3 had an effect regarding the improvement in quality of the developed software. The strongest effect, namely *definitive symptoms of improved productivity*, was not registered at any of the method fragments. Yet, there are effects which are just below the strongest effect and overall an effect in actual productivity improvement was found in 38,5% of the analysed method fragments.

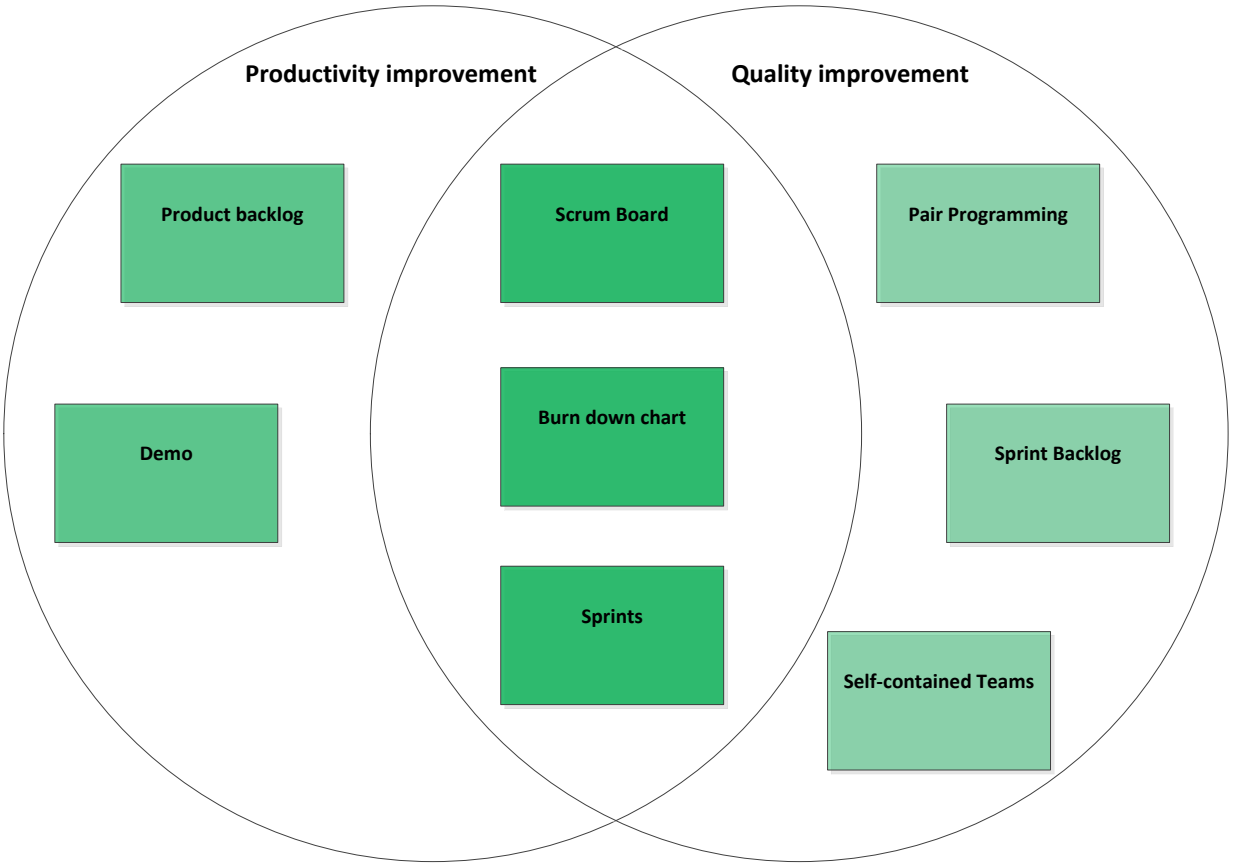

Figure 48: Venn-model depicting method fragments which had an effect on productivity and to which degree

Figure 48 visualizes the method fragments which had an effect on productivity and additionally indicates to which degree it had an effect. For instance the method fragments scrum board, burn down chart and sprints are in both the productivity improvement as well as the quality improvement area since these method fragments were found to have a positive effect on both metrics. Since these method fragments have the strongest effect, they are also of a darker colour compared to other method fragments. One can see the method fragments which only have an effect in terms of quality improvement being the lightest of the three colours.

6.3 Interpretation of the results

The data, results, and effect of the various method fragments are described and discussed previously in this chapter. The overall interpretation of the results is just as important for organisations, especially when introduction of productivity improving method fragments is the primary focus. This paragraph will provide insight into why certain method fragments did have an effect on productivity, why other method fragments did not, whether these results were to be expected, and the degree of generalizability.

Why did certain method fragments have an effect?
The quality improvement concerning the introduction of pair programming is due to the fact that two programmers work together at one workstation and review each other's code while it is being written. This (constant) code reviewing process has a positive effect on software quality. The code reviewing process consists of error correction, suggestions for alternative ways to write code, code layout changes, etcetera. This positive effect on software quality is also supported by studies specifically performed on pair programming, such as Williams and Kessler (2000), Parrish et al. (2004), and Hulkko and Abrahamsson (2005).

The scrum board' improved productivity and quality is because the team has a better overview of what has to be done and corresponding status of those development tasks. Stand up meetings at the scrum board also result in a progress indication of the overall sprint. The different status zones of a scrum board (i.e. new, in progress, ready for test, and done) enables a team to have an overview of the overall progress of a sprint and who is working on what user story. There is no universal layout of the scrum board established and therefore no single layout or design is responsible regarding a positive influence but rather that the team must refine and adjust the scrum board layout, if necessary, to its own preference. This was also acknowledged by some teams that were interviewed, for instance team 4. Details about the adjustments this team made to the scrum board can be found in chapter 7.1.

The product backlog enables a team to have an overview of the requirements of future sprints, as a result, the team has knowledge and awareness of what kind of functionality is planned for development and will be added in the (near) future. The product backlog can also be considered a longer term planning compared to sprint backlogs.

The sprint backlog had a positive influence because due to this method fragment the team knows in detail what has to be developed during the coming sprint. It breaks the work that has to be done down into smaller pieces so that developers know in detail what has to be developed. When not using sprints and sprint backlog level of detail one could assume aspects that were not registered in detail in for instance a user story. These assumptions can be false, resulting in for instance the customer receiving software functioning in a way it had not anticipated. The breakdown level of detail of a sprint backlog enables a team to know precisely what the content is that has to be developed, in what order it should be developed, and when the specified development work should be completed (at the end of a certain sprint).

The improved productivity regarding the burn down chart is because it motivates one to keep a decreasing trend in terms of remaining development work by showing the progress per day in a graphical manner. Team 9 for instance drew the burn down chart every day next to the scrum board to update and show the team its progress throughout the sprint in question. This also enables a team to interpret graphically whether they are on schedule or not and, additionally, helps to build some pressure on the team when the burn down chart is not decreasing enough over the course of a sprint.

The demo positively influenced productivity because the reviewing process allows team members to solve problems and unlock solutions for the next sprint, improving the agile process for usage within the team. The sprints, and thus iterative development, had a productivity improvement since it allows one to concentrate on a set amount of development per iteration compared to for instance the waterfall method. Sprints allow clear deadlines when working software must be delivered, along with a detailed work specification in the form of the sprint backlog.

The self-contained team' method fragment improved software quality by means of the multi-disciplinary aspect and therefore every team member performs development tasks as well as testing tasks. Every team member fulfils several roles, resulting in various views on the overall development. By enabling team members to fulfil multiple roles, one does not have to wait on a specific team member to perform a particular task. An example for instance is that a specific team member which tests all developed software solely performs this (testing) task. When this team member is not available or is overloaded with test duties, there is a possibility that not all aspects of the sprint backlog are completed in time or were not properly tested. This is considered not beneficial since one has to wait on a specific person to perform a particular task. Team members will always have various levels of expertise regarding different types of tasks but this method fragment allows a team to perform several types of tasks by every team member.

Why did some method fragments have an effect and others not?
As depicted in table 34, there were varying effects measured regarding the influence on productivity. The fact that eight method fragments were found to have an effect on productivity was due to the quantitative data proving just that. The data is the evidence for stating the actual effect. Method fragments that did not have a positive effect on productivity was due to the data indicating a non-positive effect. Apart from the data fact, this research is unable to state why method fragments like user stories and acceptance test did not influence productivity in a positive manner. This research focused specifically on method fragments that influenced productivity in a positive manner, examining and identifying why certain method fragments had a non-positive effect was out of this research' scope.

Expectation of results
Since not much research has been conducted on the level of agile method fragments and its influence on productivity only two expectations in terms of results could be withdrawn from literature. Sources, mentioned previously, indicated that pair programming had a positive influence on software quality. Concerning sprints it was mentioned that the total amount of sprints that were performed improved productivity in a positive manner instead of the sprints method fragment itself. Based on literature sources one could not expect the results that were achieved due to the fact that not much research has been performed specifically on method fragment level. Apart from the literature sources there were also the qualitative expectations mentioned in paragraph 5.3 concerning the scrum board, user story prioritization, and retrospective. Yet these indications were not aimed towards productivity improvement but rather for improving the overview and steering the team.

The result of pair programming was expected, and conforming with literature, as the software quality was positively affected. The effects of other method fragments were not expected because there was no data or results for these method fragments and additionally it was not expected prior to this research that eight out of thirteen method fragments were found to have some sort of effect on productivity.

Degree of generalizability
Generalizability is, as with many research projects, an issue. Due to the project scope only a limited number of teams could be included for further analysis. However, the data of teams confirms each other at several method fragments. The confidence in the measured effects could be strengthened when the data of even more teams would be analysed. This will be further discussed in the limitations section of this thesis.

6.4 Influence on success

The conceptual model of process changes and its influence on productivity, as shown in figure 1, depicts several factors for the influence on success. The acceptation level at the selected teams was considered sufficient. Teams were open to change, and as a corresponding result the overall resistance was low. Chapter 7.2 describes the experienced resistance within teams and it indicates that it did not result in any major problems. There was no fluctuation in the way of implementation across the selected teams with all teams opting for introducing new method fragments at once and not in phases. This is described in more detail in chapter 8. The amount of experience could also be of influence. There are differences in terms of the overall experience of the selected teams, ranging from five to just above one year, but all have more than one year of scrum experience.

Since the selected teams have more than one year of experience, this was not considered a negative influence. Regarding the organisational fit, the combination of departments being open to change and relatively small development teams form a good basis for adopting agile development.

7. Process change experiences in practice

This section of the thesis describes the experience of the development department regarding the introduction, change, or deletion of the various method fragments. It depicts the overall difficulties and changes that were performed at several method fragments in order to let it work more efficiently within a team. The specific changes to method fragments are ordered per team. Other aspects of this chapter are the experienced resistance to change and general improvements on the overall process. This chapter will describe the experiences of all interviewed teams, thus not only the teams which were selected concerning the quantitative analysis. Additionally, this chapter uses quotes as an extra supporting factor to indicate its experience regarding the various process changes. These quotes are depicted in italic and between apostrophes. The quotes resulted from interviews conducted with scrum masters of every team.

7.1 Changes to method fragments

Team 2a/2b

Team 2a and 2b indicated that it *"took time to deal with an introduced method fragment/process change, before it was efficiently integrated in the overall process"*. As a reminder, this is also a reason why the data measurements were performed three sprints prior to the process change and three sprints after the process change, taking the experience of a new method fragment into consideration.

Some method fragments needed adjustment. *"The user stories and tasks in the sprint backlog were too big in terms of size and therefore required refinement. We found a solution for this problem in that the items in the sprint backlog were split up into smaller items."* The team indicated that this resulted in better development time estimation and a detailed overview of what had to be achieved in a particular sprint.

Team 4

Team 4 found the process of defining the user stories difficult: *"we were adjusting and changing user stories during the actual sprint. The product owner recognised this as a problem and instead of delegating this task, he took it as one of its own tasks".* This also aligns with the fact that the team adjusted the allocated amount of story points per sprint. The user stories were made smaller in terms of size and therefore the amount of story points decreased per user story from a mean of 50 to a mean of 20 story points. Despite these changes the team indicated that they *"are still struggling sometimes about what is considered a good user story"*.

The scrum board method fragment was adjusted many times concerning its layout. *"The scrum board quickly became messy and confusing since the overview was lost. After numerous changes, we found that separating the user stories into dedicated grids was the way that worked best and the decision was made to stick with this way of using the scrum board"*. A dedicated grid means that every status zone of the scrum board (i.e. new, in progress, ready for test, done) has a certain amount of reserved places (i.e. grids) to place user stories upon. As indicated, the team found that the overview was improved by using this layout.

Team 4 also indicated that every process change needed time to implement within the team: *"every process change had a learning curve and especially finetuning the method fragment after its introduction was found to be crucial."*

Team 6
The user story prioritization method fragment was changed because the team did not estimate and prioritize its user stories in an efficient manner. The prioritization and estimation of development hours were not in line with what was experienced in practice during actual development work in the sprints. The team explained: *"we opted for insight into the selected user stories for the upcoming sprint prior to the actual prioritization session. This solution resulted in the developers having more time to assess the selected user stories in terms of required development hours and overall priority."*

Team 7
Team 7 experienced a change in terms of the user story prioritization method fragment: *"the duration of the user story prioritization sessions was too long in terms of discussion length and therefore the prioritization session was shortened to avoid spending too much time on discussion"*. The team indicated that this way the time lost to actual development work was reduced.

Team 8
Team 8 were satisfied with the demo but chose to include consultants and project managers, next to the development team. *"This was done to spread more awareness regarding the amount of development work that was performed each sprint".* The same scenario was repeated for the daily meetings: *"a servicedesk employee was invited to join to also raise (daily) progress awareness at the support department"*. In terms of the user story prioritization sessions, team 8 experienced the same as team 7 regarding the discussion length of these sessions: *"therefore we opted for separate sessions for discussion purposes (if they were needed)".*

Sprint size was changed at several teams such as team 3, 4, and 8. The teams indicated when scrum was introduced, bi-weekly sprints were beneficial in terms of actively steering and supporting the team and implementing rapid changes after each sprint. When the team had several sprints of experience, three weekly sprints were mostly opted for since one can develop more and the team has the possibility to perform extra tests/bug fixing on the developed software.

7.2 Resistance to change
There is much literature to be found on the introduction of agile methodologies within organisations and corresponding resistance from its employees and management. To name two studies from the substantial amount of literature available is a study by Vijayasarathy & Turk (2008), they proved with their research that personal interest was the factor that had the most influence on agile adoption within organisations. Cohn (2009) for instance also addresses resistance on a personal level.
In practice, team 3 indicated: *"some team members needed convincing that agile development was beneficial compared to the traditional waterfall method".* Team 6 had roughly the same situation since *"developers had several different opinions at the start of its scrum usage, however this did not cause problems in practice"*. Team 7 were more specific at locating the source of resistance: *"some team members needed time to adjust to the scrum method. These team members were the ones which had worked the longest with the waterfall method and had difficulty to change to a new way of working"*.

Team 4 and its management board experienced scrum as difficult in the early phases of usage. The team explained: *"we found a solution in the form of a scrum coach and as a result the overall scrum process became smoother"*. Some team members of team 8 experienced scrum as difficult which led to resistance: *"after some time this resistance was gone due to the fact that these team members saw the added value of the method. This was achieved by explaining benefits of the method and showing examples"*. Pair programming followed the same scenario: *"it was accepted and supported when the added value was acknowledged by the team"*.

7.3 General improvements regarding the introduction of agile development

Teams also mentioned overall improvements not directly aimed at specific method fragments, but rather concerning agile development in general. The teams compared this development method to their previous way of working, for instance the waterfall method. The most mentioned improvement was the overall collaboration and the important social aspect of agile development. Both the selected teams for further data analysis as well as the other teams stated this. Development team's collaborated better and overall communication improved greatly.

Teams also mentioned other aspects that were found to be beneficial. Team 3 for instance stated: *"the overall development progress is much more visible and can be communicated and reported better"*. Team 6 explained: "*usage of scrum brought a clear structure to the development work"*. Team 8 claimed: *"the number of bugs decreased partly due to the usage of scrum and partly due to the improved interaction and collaboration of the team"*. Another improvement that was mentioned was: *"it is much clearer for developers what is expected from them"*, as stated by team 10.

7.4 Importance of experience

This chapter of the thesis depicted the experience of the development department and listed these on method fragment level and on a general level. Based on quantitative and qualitative data one can conclude that introduced method fragments should not be discarded quickly if they do not perform immediately after its introduction, one should give the team a certain amount of time (in sprints) to get experience. Many teams indicated that refinement of the method fragments and plain working experience with these method fragments, were crucial in order to let it work in an efficient manner within the development process. Due to this refinement and adjusting the method fragments to work in an optimal manner within a particular team, one can conclude that most method fragments resulted in so called situational methods. Harmsen et al. (1994) defines a situational method as an information systems development method tuned to the situation of the project at hand.

8. Manner of integration

This section of the thesis will describe in what manner method fragments that positively influenced productivity were integrated within the organisation. This description is only based on the way of integrating which was performed at the case company which all interviewed as well as the selected teams belonged to.

8.1 Integration scenario

The way of integrating method fragments into the scrum process was assessed during the interviews which were held with all the teams. All teams described the way in which they introduced method fragments into their existing scrum process. This paragraph only depicts the way of integrating of the selected teams since the process changes and corresponding method fragment of these teams were analysed.

The five selected teams all indicated that they introduced method fragments to the scrum process at once and not in separate phases. Another scenario that did occur at some of the other teams was that a method fragment was introduced in a certain sprint, then was not used for two of three sprints, and then added again. This was not done because the method fragment did not prove to be effective or beneficial but rather that it takes time to implement the method fragment correctly and during some sprints the workload was too high to dedicate time to the refinement and finetuning of a particular method fragment.

8.2 Integration per team

All selected teams implemented method fragments at once. A timeline can help indicate at what stage these method fragment introductions took place in a teams' scrum usage. Four of the five selected teams are described in this paragraph since the process changes within these teams are the most spread out.

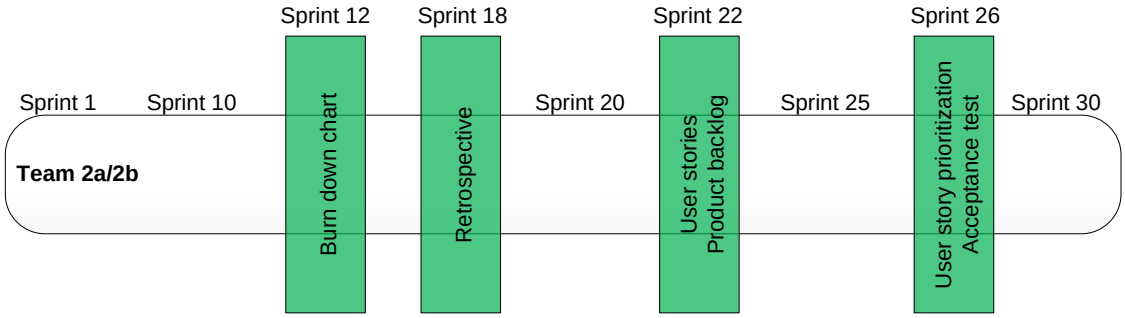

Figure 49: Integration timeline of team 2a and team 2b

Figure 49 depict the scrum usage of team 2a and team 2b. One can notice two introductions concerning single method fragments and two introductions where two method fragments are introduced in a parallel manner. Team 2a and 2b experienced process changes relatively spread over its scrum usage.

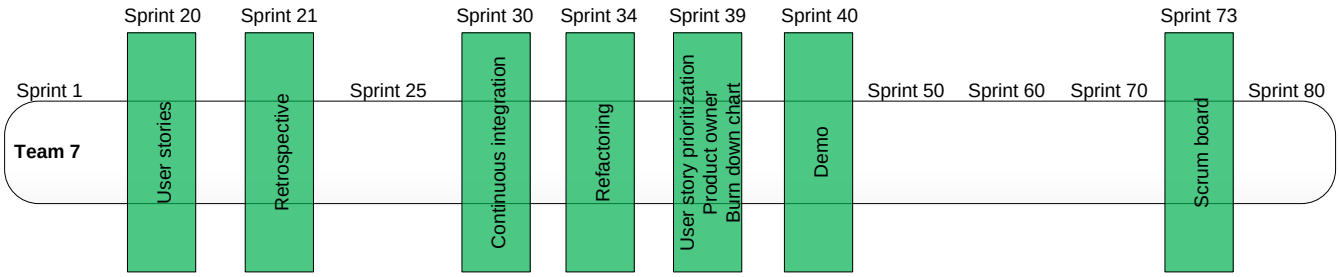

Figure 50: Integration timeline of team 7

Team 7 also spread the introduction of new method fragments. The team added single method fragments per sprint bar three. This occurred in sprint 39 where three method fragments were added namely user story prioritization, product owner, and the burn down chart.

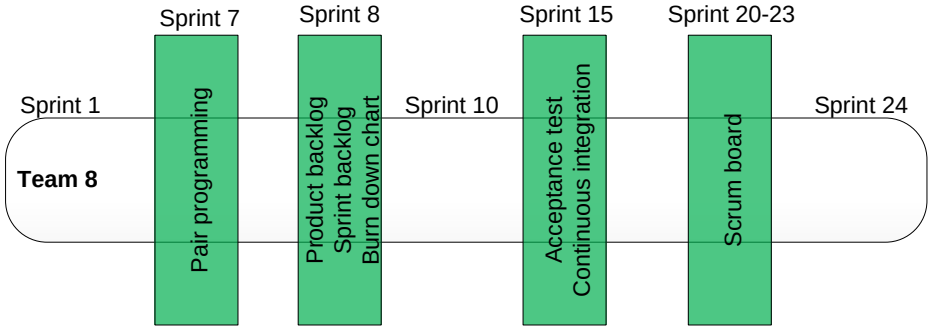

Figure 51: Integration timeline of team 8

Due to a sprint length of four weeks, team 8 has the least amount of completed sprints of the selected teams. The scrum board process change spanned over three sprints since it was deleted from sprint 20 till 22 and reintroduced from sprint 23 onwards. Three method fragments were added in a parallel manner, namely in sprint 8.

8.3 Process change preparation

Organisation tend to opt for training of teams regarding agile usage in order to prepare a team in the best possible way prior to the introduction of an agile method. Many of the interviewed teams did not have such a dedicated training prior to using scrum, only the teams with the lowest amount of experience opted for a scrum coach at the start of their scrum usage. Team 1, 3, 4 and 9 were the teams which used a scrum coach. The usage of a scrum coach was experienced as an added value since a team has an expert which can make rapid adjustments based on its experience and expertise. Other teams had a general introduction, in the form of a presentation, by their manager and were additionally given a book (the power of scrum) in order to prepare themselves and get acquainted with agile development. One can see differences in terms of preparing a team for agile usage. The teams which only had a plenary presentation and a book to prepare for agile adoption however did not complain about lacking knowledge at the start of their scrum usage. The teams which had a scrum coach mentioned that the usage of a scrum coach was definitely an added value, especially during the start-up phase, for actively steering the team and constant feedback on the process.

9. Validity

This section of the thesis will describe the validity aspects of this research project.

9.1 Case study

No case study was conducted due to the fact that the quantitative analysis stage of this research was substantial in terms of size. The size and duration of this analysis period was also the cause that a selection of teams had to be made. Prior to the start of this research project the issue was already addressed that a case study may not be possible due to the extensive quantitative analysis. In practice, this was indeed the case. A (graduation) research project has a planning and schedule to conform with in order to not have a large delay. The period after the quantitative analysis had been completed was not deemed sufficient in terms of length in order to set up a valid case study. The initial plan for the case study was to introduce only the method fragments that have an effect on productivity into a team which did not use these method fragments, to measure whether these also have an effect in another team including active feedback from the team about the integration and usage of the newly introduced method fragments. The same strategy should have been taken as the other method fragments, namely seven sprints of analysis, and should therefore take about two months of (active) analysis work.

9.2 Expert sessions

Expert sessions were held instead of conducting a case study based on the aforementioned reasons.
Scrum coaches were used for the validation of the created PDD models. Their vision and review are depicted in paragraph 4.1.3.

The second expert session was a presentation of the research and its results for the software development managers of Centric. Not only the managers of the Gouda headquarter were present but from all locations of Centric. The content of the presentation was the problem statement, research trigger, research- and sub questions, triangular model of productivity, criteria and selection of teams, the quantitative analysis, and the effects and results. The public found the results very interesting and well supported by scientific arguments. Questions after the main presentation included:
1) The way of using complexity for measuring the functionality metric. The scaling of complexity per team was explained.
2) If there were certain factors that influenced the productivity improvement. There are factors of influence on the success (as depicted in figure 1), explained influences such as acceptance level (resistance to change) and organisational fit. It was also explained that the effect of these factors, within the selected teams, was found to be non-significant and therefore could be neglected.
3) Whether the amount of experience between the selected teams was taken into account. Explained that all the selected teams at least had one year of experience and therefore the experience of every selected team was found to be sufficient.
4) How consistent the data of the selected teams was registered. It was explained that the data (such as sprint backlog functionality, bugs, story points, and allocated hours) of the selected teams was registered in a consistent manner, and was therefore considered sufficient prior to starting the actual quantitative analysis.
5) Whether the length of analysed sprints after the process change is enough to state an effect. One has effectively four sprints if the sprint in which the process change occurred is also counted as a sprint where an effect is measured. As explained before, three sprints on itself equals a period of more than two months of development work. It should be noted that an increase in the amount of analysed sprints would enable one to state an effect with (even) greater confidence and corresponding increase in validity. This however was not possible since there were a total of twenty-eight process changes to be analysed. In order to have a chance of complying with the planning, a set period length of analysis was pre-determined at three sprints prior to the process change and three sprints after the process change.

Several experts in the field of software productivity and agile development have assessed the triangular model for measuring productivity which is proposed and used in this thesis. These experts, including high-end industry names like Jeff Sutherland and Pekka Abrahamsson, validated the productivity model by means of their expert opinion. Full details regarding their validation will not be repeated in this paragraph and can be found in chapter 3.3.1.

10. Conclusion

This chapter will conclude the research project by means of answering the central research question.

The central research question for this research project was:

"To which extent do process changes in agile software product development influence productivity?"

The first step of this research was to define and scope the term productivity (SQ1). Productivity is a broad term and defined in various ways. This research identified three main metrics by means of literature sources. This resulted in a triangular model for measuring productivity consisting of a technical metric in the form of LOC, a functional metric in the form of user stories, tasks and bug fixes, and a quality metric in the form of post-release defects. The quality metric was also assessed in the measurement process because if improved productivity leads to a decrease in software quality, this improved productivity is not beneficial for an organisation. The three metrics for measuring productivity are depicted in figure 3.

In the search for agile method fragments, this research identified 25 method fragments concerning the agile methods scrum, XP and FDD (SQ2). These identified agile method fragments were assessed on whether they were eligible for productivity effect measurements. Based on literature, process changes, and qualitative indications a weighing factor was set up to conclude that identified process changes were leading in making method fragments eligible for productivity measurements since only one method fragments had been described in literature concerning its effect on productivity. These identified process changes were strengthened by statements from literature and the various qualitative indications resulting from interviews with the (selected) teams (SQ3).

The identified process changes, and thus eligible method fragments, within the selected teams were assessed on its influence on productivity. A selection of five out of ten teams was made due to the quantitative backlog of every team being substantial, analysing every team would delay the research schedule greatly. A detailed quantitative analysis method was created prior to starting the actual quantitative analysis, consisting of rules and decisions and taking into consideration factors like complexity of functionality. Different scenarios for every metric were set up (table 3) from which a concluding effect on productivity could emerge.
Results showed that 8 out of 13 method fragments had an effect on productivity, in varying strength.
5 of those 8 method fragments had an effect on actual productivity improvement, three had an effect regarding the improvement in quality of the developed software. The strongest effect, namely *definitive symptoms of improved productivity*, was not registered at any of the method fragments. Yet, there are effects which are just below the strongest effect, namely *symptoms of improved productivity.* Overall an effect in actual productivity improvement was found in 38,5% of the analysed method fragments. The influence on productivity per method fragment is depicted in table 34 (SQ4).
It was important to state, next to quantitative data indicating an effect, in what way an organisation should interpret a method fragment' influence on productivity. Since it is important for an organisation which would like to implement productivity improving method fragments to also understand why a particular method fragment influences productivity in a positive manner. This leads to an organisation obtaining the knowledge into the exact reason why certain method fragments influenced productivity in a positive way.
Since not much research has been performed concerning productivity analysis on method fragment level, only one method fragments could be checked on conformation with available literature. The pair programming method fragment did conform with results from other studies with the software quality increasing after the introduction of this method fragment (Williams and Kessler, 2000, Parrish et al., 2004, and Hulkko and Abrahamsson, 2005). Next to the identified productivity effects there were various qualitative indications of general improvement from teams concerning method fragments like the scrum board and the retrospective. Improvements in general, and not specific to one method fragment, regarding the usage of scrum were overall collaboration of the team, the important social aspect, the development progress is much more visible, and it is much clearer for developers what is expected from them (SQ5).

In terms of method fragment integration, all teams integrated its method fragments at once and not in separate phases. A scenario that did occur at some teams was that a method fragment was introduced in a certain sprint, then not used for two of three sprints, and re-added again. This was not done because a method fragment did not prove to be effective or beneficial but rather that it takes time to implement a method fragment correctly and during some sprints the workload was considered too high to dedicate time to the refinement and finetuning of a particular method fragment (SQ6).

This research made a contribution to the field of productivity, method engineering, and agile development by identifying which method fragments of agile development methods influenced productivity and to which degree it had an effect. It is the first research project which measured productivity on a method fragment level by means of a triangular model consisting of three metrics, which was validated by experienced authors on other (software) productivity studies.

This chapter depicts the specific answer to the central research question. Next to this central research question, six sub questions helped answering it and were summarised in this chapter. These sub questions have additionally been answered in detail in previous chapters. Figure 52 depicts which sub question is addressed and answered in which chapter. This is also beneficial in terms of the traceability of the various aspects of this research as well as displaying all links through the research project. The specific data which answered every sub question is mentioned in the chapters itself, from which the chapter number is depicted in the figure.

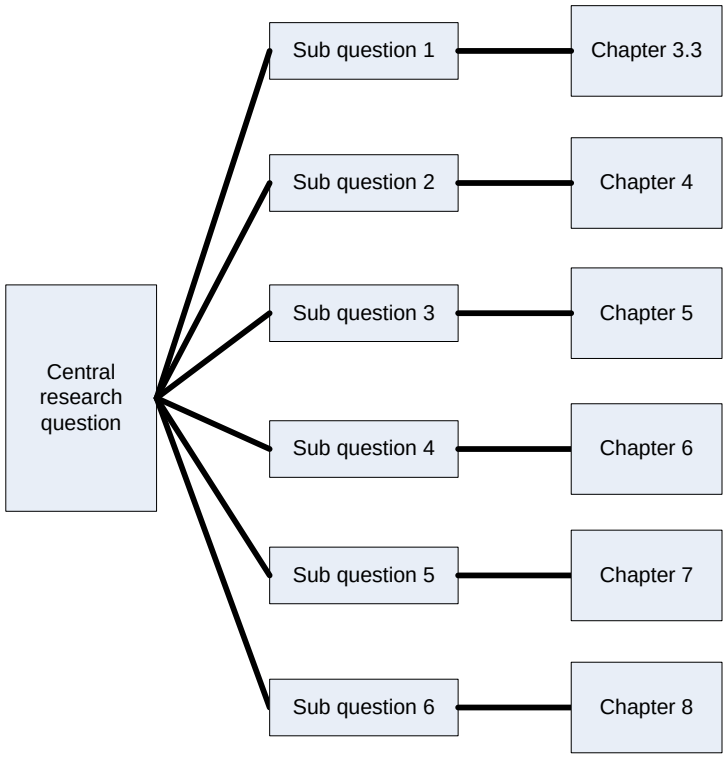

Figure 52: Traceability of every sub question

11. Limitations and future research

The limitations and future research section of this thesis will describe the limitations concerning this research project and what aspects are eligible for future research.

Generalizability is a concern for many research projects. It relates to what extent the results can be generalized to the field. Generalizability is always a concern for thesis projects since these are bounded by a mean of seven months of research. This research forced a selection of five teams in order to have a chance of remaining on schedule. This was due to the size of the quantitative backlog of every team. Should more teams have been included in the quantitative analysis phase, then the generalizability of the results could have been enhanced. However, the data of the five selected teams was substantial and therefore gave a proper indication concerning productivity influence.

The quantitative analysis used a set period length of analysis for every method fragment, namely seven sprints. This number of sprints was based on the time spent on every method fragment, with regard to the overall schedule of the thesis project. An increase in the amount of analysed sprints would have resulted in an increase of the duration of the thesis project, thus missing its deadline. Analysing more sprints per method fragment would enhance the confidence in stating an effect per method fragment. The as-is situation and the period in which the effect is measured, however, concerns more than two months of development work. This is considered sufficient to state an effect on productivity, also taking into account the experience effect. The experience effect is explained in paragraph 7.4.

The data analysis itself also had three limitations with regard to the availability and consistency of quantitative data. Team 2b had no PRD data available due to the fact that this team worked on a software product which was not yet released publicly and therefore did not have any post-release defects which were needed to assess the quality metric. Team 7 did not have its data available and consistent as first thought and indicated. Due to this, two method fragments (and seven process changes) were excluded from further analysis namely the product owner and refactoring. No LOC could be measured at team 8 since this team used oracle forms (binaries), these type of files could not be analysed with the available tools.

The inability to perform a case study is also concerned a limitation of this research project, and is already described in the validity chapter (9.1).

A third type of limitation is that teams have different skill factors concerning its development tasks. This was not taken into account in this research since it was too complex and time consuming to identify the skill level of each team and its corresponding individuals. Some individuals and teams will be better and more advanced in certain development tasks than others and it is possible that this slightly effected productivity measurements. An individual or team that is more skilful in certain development tasks can perform such work better and possibly in a shorter amount of time compared to individuals and teams that are less skilful.

This research forms a proper baseline for future studies to perform a productivity study with the same metrics depicted in the triangular model of figure 3. The description of the productivity model and metrics share the knowledge needed for researchers to perform a similar study and the experience gives an indication of how the results can be displayed, concluded and interpreted. Future research also enables another study on the same (agile) method fragments within another organisation and corresponding teams. This would also increase the generalizability of the results of this research project, yet only when the same effects are measured. Other studies, which have a longer lifespan, could also include more sprints in the quantitative analysis phase to increase the amount of confidence in which the effects are stated.

References

Abrahamsson, P. (2003). Extreme programming: First results from a controlled case study. In *Euromicro Conference, 2003. Proceedings. 29th* (pp. 259-266). IEEE.

Adams, P. J., & Capiluppi, A. (2011). Bridging the gap between agile and free software approaches: The impact of sprinting. *Multi-Disciplinary Advancement in Open Source Software and Processes*.

Agile Manifesto (2001). *Manifesto for Agile Software Development*. Retrieved from http://agilemanifesto.org/

Albrecht, A. J. (1979). Measuring application development productivity. In *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium* (Vol. 10, pp. 83-92). Monterey, CA: SHARE Inc. and GUIDE International Corp.

Albrecht, A. J., & Gaffney, J. E. (1983). Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation. *IEEE Transactions on Software Engineering*, *SE-9*(6), 639–648.

Ambler, S. (2012). Feature driven development (FDD) and agile modeling. *Agile Modeling*. Retrieved 24 January, 2013, from: http://www.agilemodeling.com/essays/fdd.htm.

Arisholm, E., Gallis, H., Dyba, T., & Sjøberg, D. I. K. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise, *33*(2), 65–86.

Banker, R. D., Datar, S. M., & Kemerer, C. F. (1991). A model to evaluate variables impacting the productivity of software maintenance projects. *Management Science*, *37*(1), 1-18.

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice*. Addison-Wesley Professional.

Beecroft C, Rees A, Booth A (2006). Finding the evidence. In: Gerrish K, Lacey A, eds. *The Research Process in Nursing.* 5th edn. Blackwell Publishing, Philadephia: 90–106.

Beedle, M., Devos, M., Sharon, Y., Schwaber, K., & Sutherland, J. (1999). SCRUM: An extension pattern language for hyperproductive software development. *Pattern Languages of Program Design*, *4*, 637-651.

Boehm, B. W., Madachy, R., & Steece, B. (2000). *Software Cost Estimation with Cocomo II with Cdrom*. Prentice Hall PTR.

Brinkkemper, S. (1996). Method engineering: engineering of information systems development methods and tools. *Information and software technology*, *38*(4), 275-280.

Cabri, A. & Griffiths, M. (2006) “Earned Value and Agile Reporting,” *Proceedings of AGILE 2006 Conference (AGILE’06)*, pp. 17-22.

Carnwell R, Daly W (2001). Strategies for the construction of a critical review of the literature. *Nurse Educ Pract* 1: 57–63.

Cockburn, A., & Highsmith, J. (2001). Agile software development, the people factor. *IEEE Computer*, *34*(11), 131–133.

Cohen, D., Lindvall, M., & Costa, P. (2003). *A DACS State-of-the-art Report*. *A DACS State-of-the-art Report*.

Cohn, M. (2004). *User stories applied: For agile software development*. Addison-Wesley Professional.

Cohn, M. (2009). Succeeding with agile: software development using Scrum. Addison-Wesley Professional.

Cronin, P., Ryan, F., & Coughlan, M. (2008). Undertaking a literature review: a step-by-step approach. *British journal of nursing (Mark Allen Publishing)*, *17*(1), 38–43.

Curtis, B. (2000). The global pursuit of process maturity. *IEEE Software*, *17*(4), 76–78.

Diaz, M., & Sligo, J. (1997). How software process improvement helped Motorola. *IEEE Software*, *14*(5), 75–81.

Fitzgerald, B., Hartnett, G., & Conboy, K. (2006). Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems*,*15*(2), 200-213.

Fowler, M., & Beck, K. (1999). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.

Fowler, M., & Foemmel, M. (2006). Continuous integration. *Thought-Works.* Retrieved 23 January, 2013, from: http://www. thoughtworks. com/Continuous Integration.pdf.

Gell-Mann, M. (1995). What is complexity. *Complexity*, *1*(1), 16-19.

George, B., & Williams, L. (2004). A structured experiment of test-driven development. *Information and Software Technology*, *46*(5), 337-342.

Green, G. C., Hevner, A. R., & Webb Collins, R. (2005). The impacts of quality and productivity perceptions on the use of software process improvement innovations. *Information and Software Technology*, *47*(8), 543–553.

Grover, V., Teng, J., Segars, A., & Fiedler, K. (1998). The influence of information technology diffusion and business process change on perceived productivity: the IS executive's perspective. *Information & Management*, *34*.

Haley, T. (1996). Software process improvement at Raytheon. *IEEE*, *13*(6), 33–41.

Hansen, B., Rose, J., & Tjørnehøj, G. (2004). Prescription, description, reflection: the shape of the software process improvement field. *International Journal of Information Management*, *24*(6), 457–472.

Harmsen, F., Brinkkemper, S., & Oei, H. (1994). Situational Method Engineering for Information System Project Approaches. *IFIP WG8.1 Working Conference on Methods and Associated Tools for the IS Life Cycle* (pp. 169–194).

Humphrey, W. S., Snyder, T. R., & Willls, R. R. (1997). Software process improvement at Hughes aircraft. *Software, IEEE*, *14*(5), 75–81.

Hulkko, H., & Abrahamsson, P. (2005). A multiple case study on the impact of pair programming on product quality. *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, 495–504.

Ilieva, S., Ivanov, P., & Stefanova, E. (2004). Analyses of an agile methodology implementation. *Proceedings of the 30th Euromicro Conference* (pp. 326–333). IEEE.

Iversen, J., & Ngwenyama, O. (2006). Problems in measuring effectiveness in software process improvement: A longitudinal study of organisational change at Danske Data. *International Journal of Information Management*, *26*(1), 30-43.

Janzen, D., & Saiedian, H. (2005). Test-driven development concepts, taxonomy, and future direction. *Computer*, *38*(9), 43-50.

Jiang, J. J., Klein, G., Hwang, H.-G., Huang, J., & Hung, S.-Y. (2004). An exploration of the relationship between software development process maturity and project performance. *Information & Management*, *41*(3), 279–288.

Jones, J. (2006). An introduction to factor analysis of information risk (fair). *Norwich Journal of Information Assurance*, *2* (1), 67.

Kellner, M. I., Madachy, R. J., & Raffo, D. M. (1999). Software process simulation modeling: Why? What? How? *Journal of Systems and Software*, *46*(2-3), 91–105.

Kniberg, H. (2007). Scrum and XP from the Trenches. *InfoQ Enterprise Software Development Series*.

Layman, L., Williams, L., & Cunningham, L. (2004). Exploring extreme programming in context: An industrial case study. In *Agile Development Conference, 2004* (pp. 32-41). IEEE.

Lindstrom, L., & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information Systems Management*, *21*(3), 41-52.

Lyndsay, J. (2007). Testing in an agile environment. *Workroom productions.* Retrieved January 23, 2013, from: http://www.workroom-productions.com/papers/Testinginanagileenvironment.pdf

Mahmood, M. A., Pettingell, K. J., & Shaskevich, A. I. (1996). Measuring Productivity of Software Projects: A Data Envelopment Analysis Approach. *Decision Sciences*, *27*(1), 57–80.

Maurer, F., & Martel, S. (2002). On the productivity of agile software practices: An industrial case study. *In Proceedings of the International Workshop on Global Software Development*. Orlando, FL, USA.

Maximilien, E.M. & Williams, L. (2003) "Assessing Test-driven Development at IBM," *Proceedings of the International Conference of Software Engineering*, Portland, OR, pp. 564-569.

Maxwell, K. D., Van Wassenhove, L., & Dutta, S. (1996). Software development productivity of European space, military, and industrial applications. *Software Engineering, IEEE Transactions on*, *22*(10), 706-718.

Moser, R., Abrahamsson, P., Pedrycz, W., & Sillitti, A., Succi, G. (2008). A case study on the impact of refactoring on quality and productivity in an agile team. *Balancing Agility and Formalism in Software Engineering*, 252–266.

Muller, M.M. & O. Hagner (2002). Experiment about test-first programming, presented at Empirical Assessment In Software Engineering EASE ’02, Keele.

Nejmeh, B.A., W.E. Riddle, Nejmeh, B., & Riddle, W. (2006). A framework for coping with process evolution. *Unifying the Software Process Spectrum*, 302-316.

Nierstrasz, O. (2004). Software Evolution as the Key to Productivity. *Proceedings of Radical Innovations of Software and Systems Engineering in the Future* (pp. 274–282).

Nosek, J. (1998). "The Case for Collaborative Programming," Comm. ACM , *41*(3), pp. 105–108.

Paetsch, F., A. Eberlein, and F. Maurer (2003). "Requirements Engineering and Agile Software Development," *Proceedings of the 12th IEEE international Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 308- 313.

Parrish, A., Smith, R., Hale, D., & Hale, J. (2004). A field study of developer pairs: Productivity impacts and implications. *Software, IEEE*, *21*(5), 76-79.

Paulk, M., Weber, C., & Garcia, S. (1993). Key practices of the Capability Maturity Model version 1.1.

Perry, D. E., Siy, H. P., & Votta, L. G. (2001). Parallel changes in large-scale software development: an observational case study. *ACM Transactions on Software Engineering and Methodology*, *10*(3), 308–337.

Project Management Institute (corporate author) (2008). *A Guide to the Project Management Body of Knowledge (PMBOK Guide)*. Project Management Institute.

Putnam, L. H. (1978). "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering*, vol. SE-4, no. 4, pp. 345-361.

Rossi, M., Ramesh, B., Lyytinen, K., & Tolvanen, J. P. (2004). Managing evolutionary method engineering by method rationale. *Journal of the Association for Information Systems*, *5*(9).

Schwaber, K. (1995). Scrum development process. In *Proceedings of the Workshop on Business Object Design and Implementation at the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'95)*, pp. 117-134.

Schwaber, K., & Sutherland, J. (2010). What is Scrum. Retrieved January 23, 2013, from: http://www. scrumalliance. org/system/resource/file/275/whatIsScrum.pdf.

Scrum Alliance (2003). *The state of scrum: benchmarks and guidelines*.

Sharp, H., Robinson, H. (2008). Collaboration and co-ordination in mature eXtreme programming teams. *International Journal of Human-Computer Studies*, *66* (7), 506–518.

Shih, C.-C., & Huang, S.-J. (2010). Exploring the relationship between organisational culture and software process improvement deployment. *Information & Management*, *47*(5-6), 271–281.

Shukla, A. and Williams, L. (2002). Adapting Extreme Programming for a Core Software Engineering Course, *in Proceedings of the Fifteenth Conference on Software Engineering Education and Training*, IEEE. p. 184-191.

Stelzer, D., & Mellis, W. (1998). Success factors of organisational change in software process improvement. *Software Process: Improvement and Practice*, *4*(4), 227-250.

Sutherland, J. (2005). Future of scrum: Parallel pipelining of sprints in complex projects. *In Proceedings of the Agile Conference, 2005,* pp. 90-99.

The Standish Group (2012). *The CHAOS Manifesto*.

Tracz, W. (1994). Domain-specific software architecture (DSSA) frequently asked questions (FAQ). *ACM SIGSOFT Software Engineering Notes*, *19*(2), 52-56.

van de Weerd, I., Brinkkemper, S. (2008). Meta-Modeling for Situational Analysis and Design Methods. In M.R. Syed, S.N. Syed (eds.), *Handbook of Research on Modern Systems Analysis and Design Technologies and Applications* (pp. 35-54). Hershey: Idea Group Publishing.

van de Weerd, I., Brinkkemper, S., & Versendaal, J. (2010). Incremental method evolution in global software product management: A retrospective case study. *Information and Software Technology*, *52*(7), 720-732.

van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., Bijlsma, L. (2006). Towards a Reference Framework for Software Product Management. *Proceedings of the 14th International Requirements Engineering Conference*, Minneapolis/St. Paul, Minnesota, USA, 312-315.

Vijayasarathy, L.E.O.R., & Turk, D. (2008). Agile software development : A survey of early adopters. *Journal of Information Technology Management A Publication of the Association of Management*, *19*(2), 1–8.

Wellington, C. a., Briggs, T., & Girard, C. D. (2005). Comparison of Student Experiences with Plan-Driven and Agile Methodologies. *Proceedings Frontiers in Education 35th Annual Conference*.

Williams, L., & Kessler, R. R. (2000). All I really need to know about pair programming I learned in kindergarten. *Communications of the ACM*, *43*(5), 108-114.

Williams, L., Kessler, R. R., Cunningham, W., & Jeffries, R. (2000). Strengthening the case for pair programming. *Software, IEEE*, *17*(4), 19-25.

Williams, L., Krebs, W., Layman, L., Antón, A. I., & Abrahamsson, P. (2004). Toward a Framework for Evaluating Extreme Programming VTT Technical Research Centre of Finland, *Empirical Assessment in Software Eng.(EASE)*, 11–20.

Williams, L., Wiebe, E., Yang, K., Ferzli, M., & Miller, C. (2002). In support of pair programming in the introductory computer science course. *Computer Science Education*, *12*(3), 197-212.

Wood, W. A., & Kleb, W. L. (2003). Exploring XP for scientific research. *Software, IEEE*, *20*(3), 30-36.

Xie, T., Notkin, D. (2006). Tool-assisted unit-test generation and selection based on operational abstractions. *Automated Software Engineering Journal*, *13* (3), 345-371.

Yin, R. K. (2002). *Case study research: Design and methods* (Vol. 5). SAGE Publications, Incorporated.

Zaki, K. M., & Moawad, R. (2010). A hybrid disciplined Agile software process model. In *The 7th International Conference on Informatics and Systems (INFOS), 2010*, pp. 1-8.

Appendences

This last section of the thesis will list the appendences concerning this research project.

Appendix I – Process change identification

The first appendix is the list of method fragments used to identify process changes at the interviewed teams. The left column depicts the method fragment with its corresponding description, the right column depicts whether the method fragment is used, and if this is the case, when this method fragments was added, changed or deleted.

Agile method fragments	Used	When introduced?
Iterative development (sprints) One of the main propositions of agile development is that development is performed in iterative stages, also called 'sprints'. These sprints are the period of time in which software development takes place. The length of a sprint mostly ranges from two to four weeks, depending on which agile method is used. Feature Driven Development (FDD) for instance uses default sprints of two weeks while Scrum uses four weeks. However, every organisation can freely choose the sprint length according to its own preference.		
User stories A user story is a wish for a feature or a request for a solution to a problem, written from the customers' perspective. Cohn defines a user story as a short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system. User stories can also be considered requirements, since it specifies which functionality is to be developed. A large and complex user story is considered a so called epic, a theme is a collection or group of user stories		
Product backlog The product backlog contains all the specified user stories, ranked by priority, that belong to a particular software product. From the product backlog the sprint backlog is created. The product backlog can also be considered a list of functional and non-functional requirements that is turned into functionality.		
Sprint backlog The sprint backlog contains all the user stories/requirements that are selected for a particular sprint. This selection is to be developed during one sprint.		
User story prioritization This is a session held at the beginning of each sprint in which the user stories are selected for the coming sprint based on their priority. Additionally, the amount of development time is estimated for selected user stories. Agile methods have various names for this method fragment. Scrum for instance uses the Scrum sprint planning and XP uses the planning game.		
Daily standup meetings Daily standups are meetings that are performed at the start of each working day and are attended by the development team. These meetings consist of a summary of what work has been done yesterday, what problems they encountered and what work has been scheduled for the coming day. A daily standup meeting should last 15 minutes.		
Scrum board The story board is a physical (or digital) board on the wall in the office, which holds all the user stories selected for a particular sprint. The board is divided into several status columns; namely in the order of: new, in progress, ready for testing, and done.		

	Used	When introduced?
Customer involvement Involving the customer in the development process is one of the unique aspects of agile development. A customer representative is on-site to for instance answers questions and perform acceptance tests, the result is that the customer is actively involved in the development process.		
Pair programming Pair programming is about two programmers developing together. One is the so called 'driver', the other is the 'navigator'. The driver writes the actual code while the navigator reviews the code on-the-fly and gives suggestions for improvement. Research has proven that this principle leads to improved code quality. The pair programming principle belongs specifically to XP.		
Unit test Unit tests are automated tests which are run on developed functionality, the developed functionality should pass all unit tests before it moves to the next stage such as acceptance tests. Unit tests are not written by testers but by the developers themselves.		
Acceptance test These are tests that are defined by the customer and are used to validate the completion of a user story. Additionally, it can also give an indication if the system reacts in the expected way.		
Refactoring Refactoring is the process of structuring a systems internal structure. It entails structuring and improving the of the code after it has been written.		
Continuous integration This is a practice where the development team integrates its work frequently. By doing this at least daily, it leads to reduced integration problems and results in cohesive software.		
Collective ownership The code is owned by all developers and responsibility is shared across the team. Developers may make changes anywhere in the code when needed.		
Self-contained teams The team is multidisciplinary. One team supports the roles of testing, development, and functional design. These roles are not spread across separate teams.		
Test driven development This development strategy is about developing automated tests (unit tests) prior to writing functional code. One of the benefits of TDD is that it decreases the amount of defects and thus improves software quality.		
Retrospective meeting (sprint review) An evaluation of a completed sprint which aims to make the next sprint more effective. Problems, struggles, suggestions, and improvements are among things that are discussed in this session.		
Demo meeting Meeting held the end of every sprint to present the software that has been developed during that particular sprint. Also used to gather customer feedback.		
Product owner The product owner is responsible for representing the interests of stakeholders of the project and its product. Additionally, a product owner creates the initial requirements and release plans.		

	Used	When introduced?
Scrum master The scrum master is responsible for the overall scrum process, teaches the scrum principles to the team, and makes sure everyone complies with the practices of this agile method.		
Simple design Teams build software to a simple design. A team keeps the systems' design exactly suited for the functionality that the system has at that moment. There is no time wasted on a complex upfront design and this way the system is always ready for what is next in terms of development and functionality. Design is not an upfront thing but an all-time thing.		
Open work environment Agile development has a large social aspect. Developers work in a common workspace to improve overall communication. By opting for an open work environment developers can hear conversations of each other, stay up to date with the latest progress of a project, and help each other faster due to short (verbal) communication lines.		
Burn down chart A burn down chart can help a team visualize how many features are still to be implemented during a sprint. method fragment is not specifically for agile development, when used in an agile environment the features resemble user stories.		
Class owner Agile methods have a practice called collective ownership which is described previously in this chapter. FDD (Feature Driven Development) takes a different approach in that it assigns certain classes to individual developers. Thus, a developer 'owns' a class. If a feature requires changes to several classes then the owners of those classes must work together as a feature team to implement it.		
40 hour week A principle of XP is that every week should not exceed a 40 hour limit, which means no overtime. XP allocates this because programmers should not tire themselves out by working overtime. Developers experienced that during busy periods, working overtime results in poor artefacts.		

Appendix II – Interview template

This appendix concerns the interview template which was used in every interview. It consists of the four categories described in paragraph 6.1.1 and additionally was used to register the various answers of the interviewed teams.

Overall Scrum level

1) Hoeveel ervaring heeft het team met Scrum?

2) Hoe groot is het team? (is de grootte van het team in het verleden wel eens gewijzigd?)

3) Welke rollen worden toegepast?

4) Wat voor sprints worden er gehanteerd? (zijn deze bijvoorbeeld wisselend?)

5) Hoeveel sprints zijn er in totaal geweest?

6) Wat voor programmeer taal wordt er gehanteerd? (is de programmeertaal wel eens gewijzigd in het verleden?)

7) Hoe is Scrum geïmplementeerd? (in een keer of in stappen?)

8) Welke aspecten van Scrum worden toegepast? (lijst met method fragments voorleggen en aangeven welke fragments worden gebruikt)

Registreren in aparte method fragment lijst

9) Is het proces over tijd veranderd? Zo ja op wat voor manier? Kan ook juist zijn dat method fragments verwijderd zijn i.p.v. toegevoegd. (datum wijziging of na bepaalde sprint noteren op het method fragment formulier)

Method fragment level

10) Op wat voor manier zijn de verschillende method fragments geïmplementeerd? (in een keer of in stappen?) Voorbeeld pilot team pair programming tijdens een sprint, daily meeting eerste wekelijks en daarna dagelijks.

11) Zijn method fragments aangepast voorafgaand aan implementatie? Zo ja, welke?
Voorbeeld daily meetings niet dagelijks maar om de paar dagen, pair programming niet constant naast elkaar, customer niet elke dag aanwezig maar eens in de week.

12) Hoeveel tijd nam het in beslag om deze method fragments te implementeren?

13) Is het method fragment veranderd over verloop van tijd?
Voorbeelden andersom dan bij vraag 11.

Data level

14) Benodigde data overzicht, wat voor data wordt geregistreerd?

Benodigde data	**Aanwezig?**
Toegang tot bronsysteem/source code t.b.v. LOC	
Toegang tot bug registratie database	
Release notes	
Sprint backlogs t.b.v. inzicht in ontwikkelde functionaliteit	
Sprint retrospectives	

14) User story complexiteit, wordt dit geregistreerd? (bijvoorbeeld story points)

Organisational level

15) Wat is de mening van het development team met betrekking tot Scrum?

16) Heeft het personeel een training gehad?

17) Voldoen de resultaten aan de verwachtingen? (bijvoorbeeld die van de opdrachtgever)

18) Extra agile teams? (+ hoeveelheid ervaring, lijst van teams welke al bekend zijn)

Extra opmerkingen

- Interview individuele ontwikkelaars mogelijk? (visie van developers op verandering development proces).
- Afspraken voor benodigde data (contact opnemen wanneer data wordt geïnventariseerd)