Evaluating and Improving a Scrum Project Management Tool Aimed at Distributed Teams

Abstract
The intersection of two recent IT evolutions appear to be a source of conflict: large majorities of IT resources are being moved to off-shore locations while increasingly popular Agile development methodologies greatly value tight-knitted co-located development teams. These distributed Agile teams use a wide variety of digital (and physical) tools to support them during their work processes. The goal of this study was to evaluate a tool that was built specifically to support these teams in as many aspects of their daily workflow as possible.

This case tool aims specifically at the Scrum framework for software development. The tool was analyzed using three distinct methods: by comparing the tool with the Scrum guidelines and other scientific literature, by analyzing usage data that was gathered throughout several months of usage, and by observing directly the users of the tool and prospects interested in purchasing a license of the tool over a prolonged period of time. Two new features were designed and implemented, based on the analysis of the tool. These features were deemed useful, but not sufficient to increase the perceived value of the case tool significantly.

There appears to be very little reason for the distributed teams within the setting of the case company to use a tool that is as broad in functionality as the case tool. The distributed teams work independently and are relatively small, the majority of the communication takes place through synchronous communication channels, rendering the asynchronous case tool relatively useless. Further research should focus on the requirements for tool support in a more complex environment.

Version 1.0 - Public Version

Thesis Title

Evaluating and Improving a Scrum Project Management Tool Aimed at Distributed Teams

Thesis Number

Author

Twan van de Waerdt
Master Business Informatics
Department of Information and Computing Sciences
Utrecht University

Student ID

[Confidential data removed.]

First Supervisor

Prof. Dr. Sjaak Brinkkemper
Department of Information and Computing Sciences
Utrecht University

Second Supervisor

Drs. Kevin Vlaanderen
Department of Information and Computing Sciences
Utrecht University

External Supervisor

[Confidential data removed.]

Acknowledgements

Well, that was certainly harder than I thought it would be.

Over the course of the year that it took to write this thesis many people have helped me, people without whom I would never have been able to finish the paper that now lies before you.

[Confidential data removed.]

I would like to thank Sjaak Brinkkemper for initiating this project. Without Sjaak I would have never met the fantastic people I have met and experienced the things I have experienced. Thank you also for your supervision and pointing me in the right directions when I had gone astray. To the countless others at Utrecht University who have made this thesis possible: thank you.

Lastly I would like to thank my friends and family for their inspiration and support, without you I would never have been able to write this final sentence.

Contents

1. Introduction

The following chapter describes the research context and the resulting problem definition and research method.

1.1. The Agile Paradox

One of the major developments in IT, one which really took off some two decades ago, is the outsourcing of IT related business functions to offshore locations such as India and the Philippines, where skilled labor is available at a much lower cost. This fundamental change in the way organizations operate was, of course, made possible by the innovations in software, hardware and IT infrastructure. In some 20 years the IT outsourcing market has grown from virtually non-existent to a $246.6 billion market in 2011 (Gartner, 2012).

Another fairly recent development is the switch from rigid, linear (software) development methods, such as the Waterfall and Spiral methods, to more self-contained, iterative methodologies, such as Extreme Programming, Crystal Clear and Scrum. These Agile methods focus on delivering working software quickly to decrease time to market and to increase the ability to incorporate changes in customer requirements into the product. They greatly value tight, small and autonomous teams, and place a lot of emphasis on face-to-face contact and an independency from tools.

The intersection of these two IT evolutions would appear to be a source of conflict: on the one hand parts of IT development are being moved to off-shore locations, breaking apart the group of stakeholders involved in the development of a piece of software into distributed teams. On the other hand Agile methodologies, such as Scrum, require tight-knitted teams whose members are in frequent contact with each other. The result of this is that the communication between team members changes from mostly face-to-face communication to being more and more dependent on computer-mediated communication through specialized tools.

1.2. Problem Definition

These highly specialized tools have seen virtually no academic coverage so far. This is an interesting phenomena since they are used to develop significant amounts of value for tens of thousands of organizations world-wide. To an extent the degree to which Scrum is implemented within distributed teams, the ability for team members to work and communicate together, is limited by the capabilities of the software these teams use. In other words the software is a constraining or enabling factor in the productivity of these teams.

As such it remains relatively unclear what the requirements for such a tool should be, which processes should the tool support? What functionalities do the developers need? Which elements are considered most important by the business? Which elements of Agile development are impossible to virtualize and implement in such a software product? And, if that is the case, do Agile distributed teams require a different way of working compared to their co-located counterparts? By analyzing an existing Agile project management tool and interviewing developers and stakeholders, answers to these questions will be formed. The analysis of the Scrum framework will allow us to map Scrum processes and entities which are commonly used by Scrum teams in an offline setting to the processes and entities supported by the case tool. Missing features can be identified and a proposal on how to implement these missing features and concepts into the tool will be drafted. One or more of these proposed features will be implemented after which the perceived usefulness of these features will be qualitatively validated.

1.3. Research Context

Before delving into the specifics of the Research Model and Research Questions there are a number of contextual factors that must be clarified. As mentioned in the introduction this study takes place at an Agile consulting, training and offshoring organization.

[Confidential data removed.]

The case company is an organization that is services oriented. Even though the developers in the India office and the coaches and trainers in the Netherlands office have worked for many product companies, the case tool is the organization's first attempt at releasing a software product by itself. As such the case company is very much interested in analyzing the case tool and finding out exactly what the requirements for distributed Agile teams for a Agile project management tool are. The Agile mindset is not one that actively encourages the usage of tools, direct face-to-face communication is seen as a superior approach to developing software.

A single team in the India office is devoted to the development of the case tool, which is at the time of writing, being prepared to be launched as a stand-alone tool on the public market. The case tool is a Software-as-a-Service based product. It is available through a one-year license, and has to be used online, using an internet browser. For over a year the Scrum tool has served as a digital support tool to the case companies' offshore teams, to improve communication and the visibility of the teams' progress to their, mostly Dutch, customers. In 2012 the case company decided to rebuild the tool so it could be sold to other organizations as a Agile project management tool, rather than an 'extra service' offered to its offshoring customers.

There are multiple Scrum teams (all of whom are clients of the case company) who have been using TOOL over the course of many months, and so usage data of a significant number of sprints has been collected in the database. It is this data that will be used in the analysis of the usage of the tool. It is hypothesized that the usage data of the case tool will provide evidence for further, software tool specific, functionality which could not be derived from looking at the Scrum literature by itself. The members of the development teams (and their stakeholders) will be used to gather qualitative data about TOOL, through interviews and direct observations.

Furthermore, and more significantly, during the study the researcher functions as a Proxy Product Owner (PPO) for the case tool for the period of a year. A PPO is more or less the Scrum version of a product manager (see for further explanations *Theoretical Background - Scrum*), this allows the researcher to gain further in-depth knowledge of the processes and dynamics within the case company and the requirements for a Scrum tool by its customers.

1.4. Research Goals

The Research Objective for this study is an analysis of the usage of a Scrum project management tool, resulting in the creation of a set of suggestions for additions and improvements to this specific tool (both functional as well as non-functional). These suggestion should be generalizable to requirements for distributed Agile software development tools in general. Alongside these suggestions to the tool itself, along with possible enhancements to the Scrum framework when being used in a distributed setting. A number of these suggestion for additions will be converted into actual functionality for the tool, the perceived usefulness of these functionalities will then be validated by the users of the tool.

1.5. Research Questions

If we were to describe the earlier mentioned Research Objective as a Research Question, it can be stated as follows: *How can the perceived usefulness of advanced tool support for distributed Scrum teams be improved by making improvements based on the analysis of performance data and the observation of distributed Scrum teams?* This main research question is broken up in several sub-questions that will shape the structure of the study as a whole:

I. *What is the state of the art of distributed Scrum tooling practices in scientific literature, and which concepts of Scrum can be derived from literature?* Before the analysis is started we look what is the current practice in the field of Scrum, i.e. what existent knowledge is already there which can be used in this study? Scrum is a very well-defined and fairly uniform framework which makes it possible to derive a single set of processes and entities from the framework which we assume should be supported by the Scrum tool.

II. *Which Scrum concepts does the case tool support?* The previous subquestion made clear which processes and entities are known in Scrum and thus should be supported by a Scrum project management tool. In this subquestion attention is diverted to the case tool itself, as the concepts of Scrum which are and aren't supported by the tool are evaluated.

III. *What are the findings from the analysis of Scrum project data and how are these supported by observations of tool usage?* The data in the database of the case tool, gathered over a prolonged period of months, will be evaluated to find out how the tool is being used. Patterns that emerge from the Scrum project data analysis will have to be supported by qualitative data. The users, along with the direct observations of the researcher who has taken up the role of a Product Owner in a distributed Agile team, will be able to explain these patterns. Depending on the findings of the Scrum project data and the users' reflection on those findings advanced tool support features will be proposed.

IV. *Which new features and functionalities can be implemented within the course of the study and how are these features evaluated by the users?* During the process of the study some new features to the case tool's feature-set were proposed and implemented. The effect these new features had and how they were perceived by the users will be retrospectively evaluated. To ensure that the new features implemented during the research have the desired effect, they are evaluated by the users of TOOL.

1.6. Research Method

Given the Research Context and the Research Objective, a Research Method must be chosen which allows for the formulation of a Research Approach that will guide the study towards the desired results. The Research Method that has the best fit with the purpose and context of this study is Action Research. Action Research goes by the assumption that "*...complex social processes can be studied best by introducing changes into these processes and observing the effects of these changes*" (Baskerville, 1999). In that respect it takes a hands-on approach to research, it is a type of participatory research, during which the researcher becomes a part of the change process within the object of study (Hart, Boeije, & Hox, 2005) (Hult & Lennung, 1980). The research method is very much grounded in practice, it focusses on directly solving a problem whilst generating knowledge in the process. It therefore leads to concrete research results, which is why it has become more popular in Information systems research since the 90's (Baskerville & Wood-Harper, 1996).

The reason why Action Research is a good fit with the context and purpose of this study can be explained by taking a look at the characteristics of Action Research (as documented by (Baskerville & Wood-Harper, 1996)):

I. *The researcher takes an active role in the study, with an expected mutual benefit for both researcher and organization.* In this particular study the researcher takes up the role of Proxy Product Owner (see for further explanations *Theoretical Background - Scrum*). The Proxy Product Owner is the person who manages the project on a day-to-day basis, the managers at the case company (the actual Product Owners) set out the global lines for the development of the project. Taking up this role is essential in this study because it allows for independent and direct contact with all stakeholders involved and it allows the researcher to immediately incorporate the theoretical knowledge into practice. The benefit for the organization is that the knowledge gained from the study will be incorporated into the case tool, the benefit for the researcher is the creation of new knowledge on the field of tool support for Agile distributed teams.

II. *The obtained knowledge is immediately integrated, the researcher is an active participant wishing to utilize any new knowledge based on an explicit, clear conceptual framework.* The researcher in this case works embedded within the case company on TOOL, which makes it possible to integrate the findings into the tool and to observe and create new knowledge from that process.

III. *Commonly the research is of a cyclical nature, i.e. the theory of the research is applied in practice, and out of that process new insights are gained which are used to further the research* (Baskerville & Wood-Harper, 1996). Agile development is of course of a cyclical, incremental nature, which makes it a perfect combination with Action Research, it allows for progressive problem discovery and problem solving. During the research some early findings were translated into new features for the case tool, these new features were then evaluated to draw new conclusions. The findings from the research can be implemented incrementally which allows for the generation of new knowledge while the research is being undertaken. It is especially this cyclical nature of Action Research which is a good fit with the Scrum project management framework.

Action Research in its core exists of two distinct stages: A *diagnostic stage*, during which the context of the situation is investigated by the researcher, in collaboration with the subjects of the research and other resources available at the organization. From this stage theories are created which will be put into practice in the following stage, the *therapeutic stage*. During this stage the actual integration of the knowledge into practice take place, the resulting effects are studied and new knowledge is created (Blum, 1955).

Susman's description of the Action Research methodology describes a 5 phase, iterative process (Susman & Evered, 1978). See Figure 1 for an overview of the different phases of the process, note that in this study the five phases are only passed through once. The same five phase process is used to structure this paper, the current chapter is the Diagnosing chapter, which provides the context for the research. The following phases are identified by Susman:

Diagnosing
In this stage the primary problems for the case company's desire for change are identified and aligned with Utrecht University's research goals. The diagnosing activities are summarized in Chapter 1.

Action Planning
This stage describes what (and how) the researcher intends to do to fulfill the goals of the organization and the research institute. In this particular case it sets out the actions that must be undertaken to provide the answers for the research questions. The following chapter, which outlines the Research Structure is a summarization of the Action Planning activities that were undertaken.

Action Taking
As the name suggest, during this phase the planned actions are undertaken. In this specific study that means doing a literature analysis, an analysis of the case tool and qualitative enquiries at the various stakeholders, including the researchers observations. Some of the findings from these efforts were translated into additions to the case tool early on during the research, these new additions are also evaluated as part of this stage of the study. The goal of this phase is to determine what usage patterns can be discerned from the data and to acquire understanding on the reasons which cause these patterns to emerge. This phase will lead to a number of suggestions for improvements to the tool and underlying processes.

Evaluating
The evaluation phase will be of a retrospective nature. Early on during the research two distinct new functionalities (a digital Scrumboard and a feature that allows users to record their Retrospectives) have been implemented in the tool, under the direction of the researcher, with the goal of improving the level of usage of the tool. Using quantitative and qualitative techniques the effect of these changes on the Agile teams were analyzed. These findings will be taken into consideration when drafting the final conclusion on the requirements of Scrum tools for Agile distributed teams.

Specifying Learning
The learning aspect of Action Research is an activity that is undertaken throughout the research. The Specifying Learning aspect of Susman's approach to Action Research is formalized in the conclusions of this paper. According to Baskerville (Baskerville, 1999) the knowledge gained through an Action Research study can be used in three different ways:

I. The knowledge can be used by the organization to restructure and improve current processes.

II. The knowledge can be used in the diagnostic stage for the following research cycle within the same Action Research study.

III. When the changes from the therapeutic stage proved to be unsuccessful these conclusions can be used to advice further scientific research efforts.

In the concluding chapter the significance of this study with respect to these three different ways to using the gained knowledge will be highlighted.

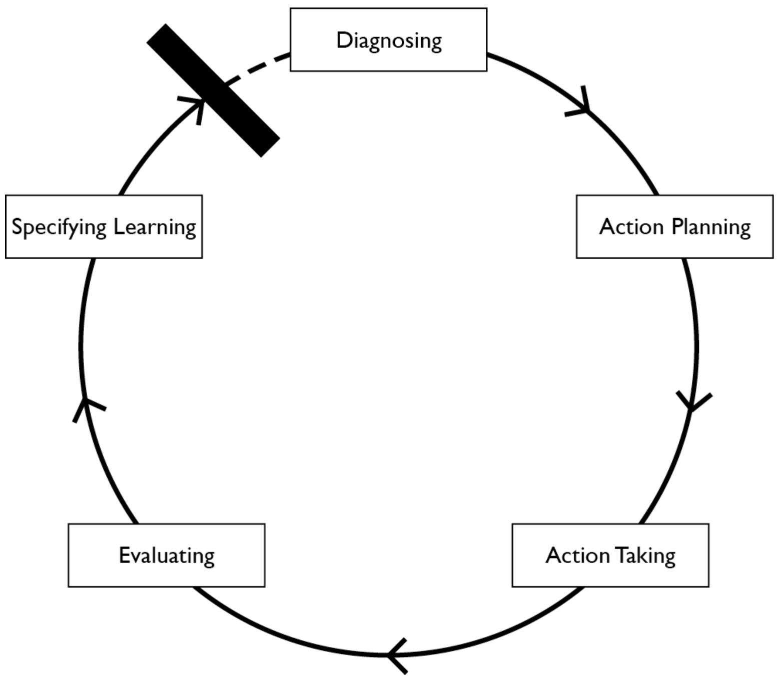


Figure 1.The Action Research approach.

Summarizing all of the above; this study is a case study that is highly exploratory in nature, tool support for Agile development methods is a field that has seen very little academic coverage, there are not many studies that could serve as a starting point for this one. Besides that, there is also a large dependency on the case tool, its users and the data those users have created in the TOOL database. A full loop of the Action Research approach will be completed. Even though four different data sources are used to help suggest improvements to the tool, it must be taken into consideration that the research effort focusses solely on a single case tool and its stakeholders. Naturally, the goal is that whatever conclusions are drawn from the investigation are generalizable to tooling support for agile distributed teams in general.The actual research follows a mixed-method approach to maximize generalizability; quantitative techniques are used to analyze the usage data, qualitative techniques are used in the form of participatory research and semi-structured interviews.

2. Research Structure

In this chapter the Research Approach used to answer the research questions stated in the previous chapter will be described and the reasoning behind their usage will be explained.

2.1. Research Approach

The Research Questions and Research Method described in the previous chapter result in following breakdown of the research.

2.1.1. A literature review

Chapter 3 will serve to answer the first sub-question: *What is the state of the art of distributed Scrum tooling practices in scientific literature, and which concepts of Scrum can be derived from literature?* This first sub-question will serve as an introduction to the Scrum framework. From general documentation on Scrum a set of standard processes and entities as used in the Scrum framework will be derived which we hypothesize a Scrum tool should acknowledge and support.

The scientific literature study will focus on requirements for Agile project management tools and will likely be rather limited due to the lack of scientific coverage of the topic of this study. Google Scholar will be used to search through the available literature. The goal of this literature review is to reach a consensus on which additional features a Scrum project management tool should have, supplementing the basic processes and entities as identified in the first part of this subquestion. We hypothesize that a theoretical framework cannot be translated to a digital tool without requiring additional elements caused by the limitations and possibilities of the usage of a digital platform.

2.1.2. A functional analysis of the case tool

In the fourth chapter the second sub-question will be answered: *Which Scrum concepts does the case tool support?* The reader will be presented with the functional architecture of the case tool and a comparison of the concepts of the tool to the concepts of the Scrum framework. Defining the architecture will help to clarify the overall structure of the tool to both the reader as well as to the researcher. A necessary intermediary step since, in order to suggest improvements to the tool and to locate areas where it is lacking, it must be known exactly what functionalities and and entities are supported. In this chapter the focus will not be too much on technical characteristics which have only marginal impact on the user experience, for instance an analysis of the used infrastructure is out of scope for this research. However, possible relations to the different entities which are suggested by Scrum and available to the users of the tool are of importance.

The functional architecture will be outlined on a high-level overview, using the Functional Architecture Modeling as proposed by Brinkkemper and Pachidi (Brinkkemper & Pachidi, 2010). A brief outlay of the tool's screens and features will also be provided. Once the features of the tool are defined it will also be possible to visually clarify which elements of the tools are lacking and which ones are not.

2.1.3. A quantitative analysis of the case tool's usage data

The fifth chapter will serve to answer the third sub-question: *What are the findings from the analysis of Scrum project data and how are these supported by observations of tool usage?* Although the case tool has only been used for internal clients, it contains a relatively large set of quantitative usage data that has been gathered through several months of usage. This usage data ranges from the titles and description teams use for work items, to the difference in the amount of work different team members pick up every iteration of the development process. This element of the study will be the most time consuming as it concerns performing data discovery operations on a dataset that is relatively unstructured.

During this phase the Goal Question Metric (GQM) approach will be used to evaluate how to the tool is being used. The GQM approach is a structured approach used for feedback and evaluation in software development (Caldiera & Rombach, 1994). This approach originates from research into defect evaluations in the NASA Goddard Space Flight Center in 1984 (Basili & Weiss, n.d.). It has since been adapted to function as a software quality measurement paradigm in software development. The GQM approach takes a top-down perspective for measurements; an organization or individual needs to achieve certain goals, the metrics should be able to show whether or not those goals have been met, and they should be able to provide guidance towards fulfilling those goals (van Solingen & van Berghout, 1999).

A goal is broken down into a specific format, so that it contains a *purpose*, an *issue*, the *object* or *process of focus* and the *viewpoint* of who or what the goal should be related to. An example of a GQM based goal is: '*improve the performance of the log-in module form a registered user's perspective*'. The breakdown of this goal would look as follows:

Purpose	Improve
Issue	The performance of
Object (or process)	The login-in module
Viewpoint	From a registered user's perspective.

Table 1. Breakdown of the Goal Question Metric Approach (Caldiera & Rombach, 1994).

A set of *questions* is selected that provide context to the goal, when these questions have been answered we should know whether or not the goal has been reached. Continuing with the example above, potential questions to evaluate whether or not this goal has been reached are: '*how long does it currently take to log in to the application?*', and '*is the log-in time improving over time?*', and '*how long does our competitor's product take to log in?*'. Finally, *metrics* are selected based on their ability to provide answers to the questions. Examples of metrics that can help to answer the first questions are: *the login time in milliseconds* and *the users' subjective perception of the login time*. A multitude of metrics are used to answer a single question in order to guarantee an accurate and objective outcome.

The GQM model thus discerns three different layers (as can be seen in Figure 2 below):

- *The Conceptual level*, in which the desired goal is described;
- *The Operational level*, in which questions are drafted that allows one to evaluate whether or not a goal has been reached, and;
- *The Quantitative level*, in which the metrics are selected that provide answers to our questions.

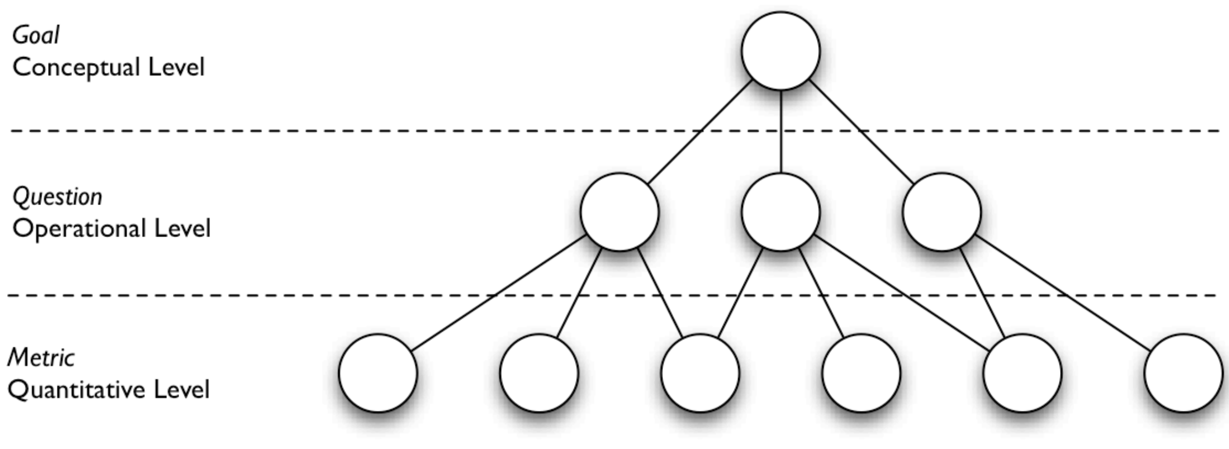


Figure 2. The Goal-Question-Metric approach.

For the analysis of the case tool's usage data the GQM approach to measuring characteristics is used. However, since the goal of the analysis is to raise questions about the usage of the tool, a bottom-up approach is taken instead - also known as the Metric-Question-Goal approach (Oman & Pfleeger, 1997). The reasoning behind this is that the analysis is started without extensive knowledge of the data contained in the data repository of the case tool. We hypothesize that questions will emerge by analyzing the data in the tool, the metrics. For example, the data might show that a certain feature of the tool is not being used, or that different teams use a specific feature in different ways. In these cases one would like to know what case tool related characteristics lead to the behavior resulting in these patterns. The case tool users, along with the experience and personal observations of the researcher, will be used to find out what is causing the behavior.

Depending on the type of patterns, the goal might be to change the tool to such a degree that it reduces or encourages the behavior resulting in these patterns. For example, if the metrics show that Backlog Items without description have a higher chance of not being completed successfully, the application could implement visual cues that encourage its users to write larger descriptions.

The data gathered from the Database Analysis will be supplemented by observations of the researcher, functioning as a Proxy Product Owner for a distributed Scrum team for the duration of one year. During this one-year period the researcher has held a large number of meetings with the developers at the India office and various other stakeholders to gather further, more in-depth, knowledge of the required functionalities of a distributed Agile project management tool. For example, when the database analysis shows that a feature is hardly used, the researcher, with the help of the users of the tool, is able to provide the answers as to why that is the case.

2.1.4. Implementation of findings into the case tool

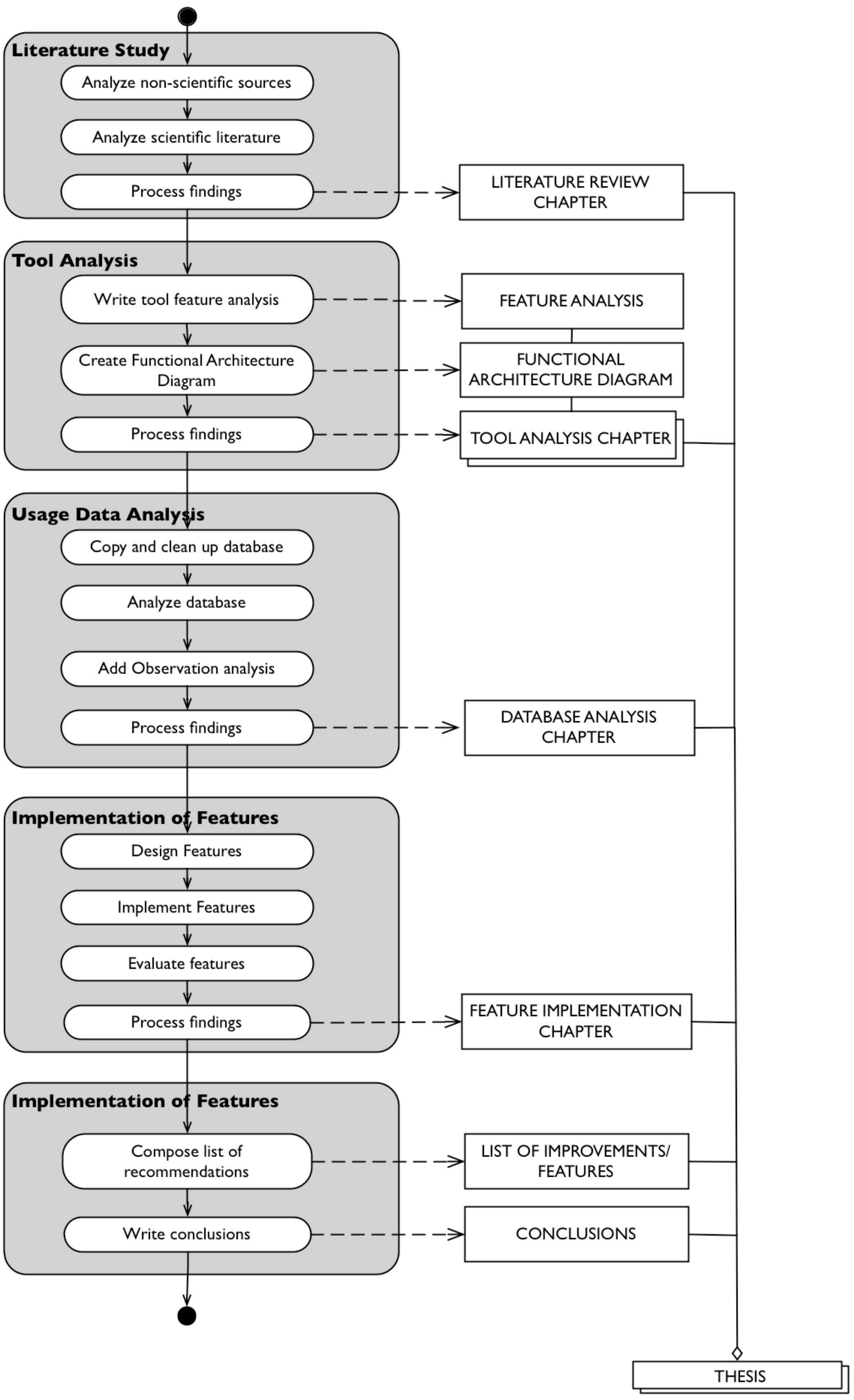
The sixth chapter will serve to answer the fourth sub-question: *Which new features and functionalities can be implemented within the course of the study and how are the new features evaluated by the users?* Naturally, not all the suggestions and findings from the research can be implemented within an acceptable time table for this study. Besides that, the case company may have conflicting schedules and product demands due to, for example, customer requests. Therefore, in coordination with the case company management and the case tool development team, only a subset of the suggestions resulting from this study will be implemented into the case tool during the course of the research. The design process behind these features will also be highlighted briefly. Focus groups, observations and non-structured interviews will be used to evaluate the new features implemented during the previous section of the study.

2.1.5. Concluding

Finally, the last chapter of the paper will serve to answer the main research question: *How can the perceived usefulness of advanced tool support for distributed Scrum teams be improved by making improvements based on the analysis of performance data and the observation of distributed Scrum teams?* All the findings from the sub-questions and the results of the implementation of the improvements are interwoven into the conclusions and recommendations of this paper.

2.1.6. Process Deliverable Diagram

Figure 3 shows a graphical representation of the research structure (in the form of a Process Deliverable Diagram). The processes of the study are depicted on the left, while the resulting deliverables are shown on the right.



Literature Study
Analyze non-scientific sources
Analyze scientific literature
Process findings
LITERATURE REVIEW CHAPTER
Tool Analysis
Write tool feature analysis
FEATURE ANALYSIS
Create Functional Architecture Diagram
FUNCTIONAL ARCHITECTURE DIAGRAM
Process findings
TOOL ANALYSIS CHAPTER
Usage Data Analysis
Copy and clean up database
Analyze database
Add Observation analysis
Process findings
DATABASE ANALYSIS CHAPTER
Implementation of Features
Design Features
Implement Features
Evaluate features
Process findings
FEATURE IMPLEMENTATION CHAPTER
Implementation of Features
Compose list of recommendations
LIST OF IMPROVEMENTS/ FEATURES
Write conclusions
CONCLUSIONS
THESIS

Figure 3. The Process Deliverable Diagram outlining the study.

2.2. Scientific Relevance

Up until now there has been very little scientific research effort focussed on the topic of distributed Agile software development, even less when it comes to the software required to support Agile concepts for development teams and their stakeholders. This study therefore could serve as a stepping stone for further research into the requirements for and usage of Agile tooling.

2.3. Societal Relevance

The most concrete societal benefit of this study is the improvement of the case tool, or the processes that take place within the case company. On a larger scale the conclusions drawn at the end of this study should be generalizable to and applicable in Agile tools in general, improving their effectiveness and efficiency, eventually leading to higher quality software being made. As Agile outsourcing grows so will the demand for tools supporting Agile processes and as such this study will become increasingly relevant.

2.4. Project Governance

The study is performed according to a modified Scrum methodology: Every week an increment of the thesis will be sent to the External Supervisor (and the Primary Supervisor if needed). On a weekly basis a Review and Retrospective session is held with the External Supervisor during which the progress of the Thesis is discussed and the steps for the following week are be discussed, on an as-needed basis.

2.5. Challenges

Three major challenges for this study can be foreseen. The first one is that there is a limited amount of scientific literature that can be used to support the outcome of the study. The result is that this study requires a precise, well documented research procedure to ensure maximum validity and generalizability.

The second challenge is the large dependency on the data contained in the database of the tool. The tool is not set up with a research purpose in mind, so there may be a lot of work required in preparing the database for the analysis. At the time of writing the database is by and large a black box that will be uncovered further and further as research progresses.

The final challenge is the culmination of the inevitable consequences of the Action Research approach. There is a large dependency on the case company during this research. Changes in customers base, development team, customer feature requirements and other contextual factors will all influence this study. It is up to the researcher to structure the research in such a way that the effect of these factors are minimal.

2.6. Validity and Reliability

Naturally, the validity of the research has to be warranted to the highest degree possible. By using a structured research approach (AR), and by strictly following its proven guidelines and processes the structural soundness of the research itself is governed. By collecting data from a multitude of sources (quantitative data from the tool itself, qualitative data from its users, the researcher who takes an active and observing role in the process and reliable literature sources) the validity throughout the research and its resulting conclusions is ensured to be as high as the context of the study allows it to be.

3.Literature Review on Scrum

The study is started with a brief introduction of the Scrum framework. The standard Scrum literature is used to summarize the main concepts and origins of the framework. This short description is then used to identify which processes and entities are defined by the Scrum framework which (as is hypothesized) should be supported by the Scrum tool. A clear distinction between processes on the one hand and entities on the other hand is made, as this allows for an easy comparison between the current state of the tool with the framework, and makes possible the identification of missing elements.

After this first step the current (scientific) literature of Agile tool support for distributed Agile teams is analyzed. The goal of this literature review is to explore the current state of Agile tooling and to find out which already existing conclusions on the field of Agile tooling requirements can be used in this research, preferably with a focus on distributed teams. We hypothesize that a framework cannot be translated into a digital platform without requiring additional requirements (both functional as well as non-functional). As mentioned in the Research Approach section, the expectations of the results of this Literature Review are modest due to the lack of scientific interest in the field of Agile tooling.

3.1. Procedure

For the introduction to the Scrum framework standard literature on the topic was used, these sources include *the Scrum Handbook*, written by Scrum co-founder Jeff Sutherland (Sutherland, 2010), *the Scrum Guide*, written by Scrum co-founders Ken Schwaber and Jeff Sutherland (Schwaber & Sutherland, 2011), *Scrum and XP from the Trenches*, written by Scrum coach and author Henrik Kniberg (Kniberg, 2007), *Agile Project Management with Scrum*, by Ken Schwaber (Schwaber, 2009), together with the researcher's personal experience being a Scrum Proxy Product Owner and the assistance of several certified Scrum coaches at the case company.

Google Scholar was used to look for the available scientific literature on distributed Agile Tooling. The search was started with very specific keywords such as: "distributed Agile tooling", "distributed Scrum tools", and so on. Note that the parenthesis were added to force Google Scholar to search for specific phrases. The titles of the resulting papers and their abstracts were scrutinized by the researcher to verify their relatedness to this research. Papers that didn't meet the requirements for this study (i.e. when the focus was not on tool support, when Agility was not part of the software process described) were discarded. Articles that were derived from scientific sources (scientific journals) were valued higher than articles from other sources (i.e. tool vendor publications, whitepapers from commercial organizations). It must be noted that the latter source was significantly greater than the first. The inclusion of articles from non-scientific sources was avoided for as long as it made sense to do so.

3.2. The Scrum Framework

This study is completely based on Agile software development practices, more specifically the Scrum approach to software-project management. Before delving into the specifics of the research, some clarification on Scrum is in order to illustrate why Agile project management tools are distinctly different from regular (software development) project management tools.

Scrum finds its origin in the late 80's and early 90's of the 20th century, when Hirotaka Takeuchi and Ikujiro Nonaka published their *'New Production Development Game'* (2006). Takeuchi and Nonaka aimed to improve production processes by working with cross-functional teams. These teams oversee the complete development of a product from beginning until the end. Back then it was the norm that specialized teams performed only a limited amount of actions within the development process, and so the product was handed down numerous specialized teams on its path towards completion. Takeuchi and Nonaka argued that this negatively influenced the productivity and delivered quality of the teams.

Some years later, in 1995, Jeff Sutherland and Ken Schwaber jointly presented a management framework derived from Takeuchi's and Nonaka's work, which they called Scrum, that focussed specifically on software product development. Since then Scrum has evolved thanks to experience and knowledge gained through the implementation of the framework in thousands of organizations worldwide. The basics, however, remain the same; a single cross-functional team is responsible for the development of software in short iterations.

So what is so fundamentally different about Scrum? Historically, software development was largely based on techniques derived from dozens of years of evolution in the manufacturing industry. These tangible products the manufacturing industry produced are often built in a distinct number of stages; when a stage has been completed work continues in the next stage. This is also known as '*Linear Development*' or '*Waterfall Development*' in IS development. Just like water that flows from basin to basin, without a way of flowing back up again, the product being developed flows from state to state. Once a product has passed a state it does not return to that particular state, much like an assembly line.

The states that are often used in (linear) software development are (Royce, 1970):

I. *Requirements gathering and specification*. During this phase the required characteristics of the product are collected/written down. It is, in essence, a very precise description of 'what the product is', without committing to a particular design. Requirements range from functional requirements ('what does the product do?'), to non-functional requirements ('how does the product do it?') to hardware requirements. It is the resulting Software Requirements Specification that makes up the scope, sets the boundaries for the rest of the development process.

II. *Design*. The requirements have to be incorporated into a single design. This encompasses every component of the product, from the underlying database structure to the Graphical User Interface the customer uses to interact with the product.

III. *Implementation*. The next state concerns the building of the product, and, if necessary, the integration of the product into existing systems.

IV. *Verification*. The verification state involves the testing of the product. Does it work as specified in the requirements and design documents? Does the product work seamlessly with the legacy systems? Are the customers satisfied with what was produced?

V. *Maintenance*. Finally, when the system or product has been put into place, it has to be maintained to ensure that it continues functioning as expected.

It can be argued that maintenance, although it is a critical aspect of the lifecycle of a software product, is not part of the development process. Scrum also does not explicitly recognize a maintenance stage within its framework. Software development is an on-going process, work that needs to be done to maintain the software can be done within this process.

The benefits of extensive specification, describing all aspects of the software product before starting to develop it, are clear: once a product has been completely described, it becomes transparent what is required to build it, hence budgets and time estimations can be made accordingly. Another benefit is that the earlier in software development faults are found, the less expensive it is to resolve them. Fixing an error in a requirements stage is much less costly than fixing a bug in a piece of software that is already in production (Shull et al., 2002).

This type of production, where work takes place in separate consecutive compartments, works very well for tangible (simple) products, as those are made out of mechanical parts and the behavior of these parts can be predicted to a large degree. Software development however, deals with intangible and often highly complex products. To illustrate, Rzevski (2011) distinguishes seven criteria of system complexity factors in his complexity framework, IT systems comply with these criteria to a high degree:

I. *Interdependence*, a system consists of numerous components which interact with each other.

II. *Autonomy*, these components have a certain autonomy in their behavior, i.e. It is not just direct control that influences their behavior.

III. *Emergence*, the overall behavior of the system as a whole emerges from the interaction of the components and is unpredictable.

IV. *Non-equilibrium*, the behavior of the system in not in equilibrium due to the fact that disruptive events take place which do not allow the system to remain in balance.

V. *Nonlinearity*, due to nonlinear relations, insignificant inputs can be amplified into extreme events.

VI. *Self-organization*, a system can change its behavior and/or configuration autonomously.

VII. *Co-evolution*, a system naturally evolves along with its environment.

This is just one example of characteristics of the complexity of a system, and does not, for example, take into account complexity factors that are derived from other aspects of development, such as project management. Needless to say, software development is a complicated type of development, as research on IT project failure rates often supports (Rubinstein, 2007; KMPG 2005; KPMG 2010; West & Grant, 2010).

These combined complexity characteristics lead to a number of different types of IS project and software development failure. Lyytinen and Hirschheim (1987) differentiate between four different categories of IS project failure:

I. *Correspondence Failure*, in this situation the system does not meet its requirements.

II. *Process Failure*, in this situation the budgeted time or costs of the project are overrun and the project is either not delivered at all, or delivered with significant overspending.

III. *Interaction Failure*, in this situation the delivered project is not being used, or is seen as unsatisfactory by the user, i.e. used with malcontent.

IV. *Expectation Failure*, in this situation the system is unable to meet the expectations of the stakeholders. A project can include all the promised functionalities, but can still fail to meet expectations, for instance when rival products have been released which providing a higher value.

It is this aforementioned complexity and the resulting symptoms Iterative Development and Incremental Development methods try to mediate. Both of these methods try to reduce complexity by breaking up tasks in smaller portions, but they both use a different approach to achieve this. Iterative development methods deliver small portions of functionality which are fully finished, like puzzle pieces in a puzzle. Incremental software development methods deliver the complete product, but in a minimally viable fashion, and then improve upon that increment. The core of both methodologies is the same: they rely heavily on the introduction of shorter development cycles.

In 2001 a group of software developers created the Agile Manifesto (Beck, Needle, Sutherland, & Schwaber, n.d.), a set of guidelines that provides the main directives for the Agile approach. Agile development methods are simply development methods which are based on Iterative and Incremental practices. The guidelines from the Agile Manifesto summarize the mindset of Iterative Development methods excellently:

I. Individuals and interactions over processes and tools.

II. Working software over comprehensive documentation.

III. Customer collaboration over contract negotiation.

IV. Responding to change over following a plan.

In Agile development methodologies the states of software development as described earlier are still being used, only now they do not cover the entire product, but increments of functionality of the product. For instance, only a minimally viable login screen of a mobile application is specified, designed, built and tested, instead of the complete application. Later on this login screen may be enriched with, for example, a ‘forgot password’ option. These incremental iterations allow for quick feedback cycles with the stakeholders, and thus changes to requirements can be easily made. It also guarantees that at least a portion of the software (that portion with the highest priority/value) is always finished. An iteration in Scrum typically lasts between 1 week and 1 month (Schwaber, 2009).

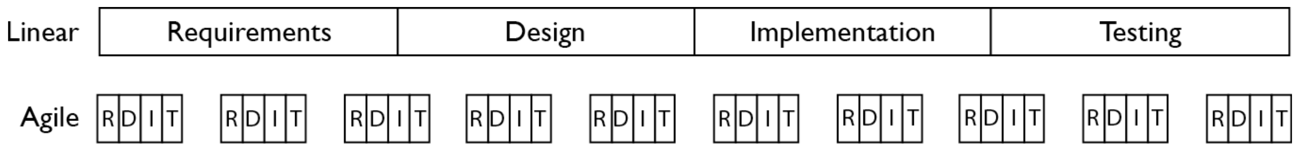


Figure 4. A comparison of Linear and Agile development techniques.

Mapping Agile software development on the failure categories as defined by Lyytinen and Hirschheim: Due to the quick feedback cycles Correspondence Failure can be mitigated by changing the requirements based on the feedback that is received at the end of the iteration. Process Failure is resolved due to the fact that at the end of every iteration some business value is always delivered. And thus when the time or budget runs out there will still be a Minimally Viable Product that should be ready for release. Interaction Failure again can be mitigated by involving the stakeholders at the end of every iteration. Their feedback is taken into account during the following iterations, which should result in a product that is continually adjusted to their requirements. Expectation Failure is yet again moderated by involving all stakeholders continuously, not only at the requirements, design and testing state.

Scrum clearly complies with the Agile Manifesto guidelines. Its focus on small, multi-disciplinary teams and transparency and face-to-face contact adheres to the '*Individuals and interactions over processes and tools*' guideline. The second guideline, '*Working software over comprehensive documentation*', is met due to the fact that at the end of every Sprint at least some functionality is delivered. This Minimally Viable Product approach to development ensures that at the end of a development process there will at least be a releasable product delivered that holds business value. Working software is not delivered only at the end of the development process, it is delivered continuously. Both *'Customer Collaboration over contract negotiation'* and '*Responding to change over following a plan*' are made possible through iterations/quick feedback cycles.

Scrum can be easily combined with other Agile practices, such as Extreme Programming and Test Driven Development. One of the major benefits of short cycle development practices is that it becomes increasingly simple to gather and analyze metrics on the progress of the development team and the project they are working. This study relies heavily on the metrics that are gathered every Sprint.

The Scrum framework is an Agile management framework that prescribes specific roles, artifacts and terminology. An iteration in Scrum is called a Sprint, these Sprints typically last from a week up to a month (Schwaber, 2009). Important to note is that every sprint should have the same duration, this 'heartbeat' allows for increased predictability, for instance the productivity of a software development team can be analyzed and compared when the team works in iterations of a set time duration. All meetings and activities in Scrum are time-boxed, they also have a set duration. To illustrate, the Sprint Planning meeting is scheduled to take up eight hours for a one month Sprint. The duration decreases progressively with the Sprint duration, a Sprint Planning meeting for a two-week Sprint is four hours. When the timebox expires and not all work is completed, the resulting work is picked up in the following Sprint, there are no extensions to the Sprint.

3.3. Scrum Roles

Scrum places a lot of emphasis on teamwork and the team members' independency from involvement of higher management. At the core of Scrum is the Scrum team, this is the team that is responsible for delivering the software at the end of every iteration. Scrum teams consist of an interdisciplinary group of people, meaning that the team members posses all the skills necessary to complete the work they committed to for a Sprint, ranging from system architects, to QA experts to UI designers. Although the composition of Scrum Teams can change over time depending on the work that needs to be done, although it must be noted that team stability is often mentioned as a pre-requisite for highly effective teams (Schwaber, 2009). Scrum defines four roles:

I. The Scrum Team

II. The Development Team

III. The Scrum Master

IV. The Product Owner

The Development Team
The Scrum Team consists of the Development team and a Product Owner. Literature suggests that a Scrum team is ideally comprised of 7 people, plus or minus two (ref). When a team is too large, coordination between the team members becomes too difficult. When a team is too small, it is hard to compose a team that has all the necessary skills and the planning the team does will become less accurate.

The Scrum Master
The Scrum Master ensures that the team can work without hindrance and that possible roadblocks for the team are dealt with accordingly. For example the Scrum Master makes sure everyone attends the daily meetings so that everyone is kept up to date on the latest developments. The Scrum Master also takes part in the actual development process during a Sprint and is therefore also a member of the Development Team.

The Product Owner
The Product Owner is not a part of the Development Team. He or she is the one who manages what the team works on during a Sprint, comparable to a Project Manager in a typical software development process. The Product Owner is the Scrum team member who is responsible for prioritizing/changing and adding items to the Product Backlog, based on feedback from the customer and other stakeholders. He or she is the link between the *business* and *development*. The Product Owner is ultimately the person who decides whether or not a User Story can be considered as complete or done.

These roles are shown in the meta-model below (Figure 5).

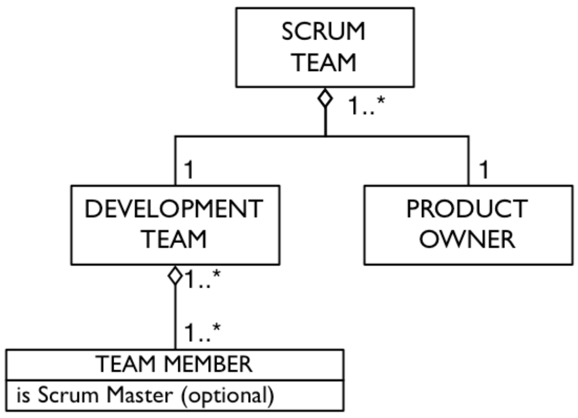


Figure 5. Scrum Roles meta-model.

3.4. Scrum Processes

Within a Sprint there are several specific meetings and processes which are repeated on a regular basis (see Figure 6 for an Activity Diagram of the Scrum processes):

I. The Sprint Planning Meeting

II. The Grooming Meeting

III. The Daily Standup

IV. The Sprint Review

V. The Retrospective Meeting

The Sprint Planning
The Sprint Planning meeting is divided into two parts, during the first part of this meeting the development team, in cooperation with the Product Owner, decides on what it is going to do in the following Sprint. This is based on the priority of the backlog items and other constraining factors such as work capacity and

dependencies between backlog items. When the participants decide to take up an item in the Sprint it is moved from the Product Backlog to the Sprint Backlog.

During the second part of the Sprint Planning meeting the team breaks up the backlog items that have been committed to and transferred to Sprint Backlog into more detailed and smaller pieces of work called Tasks. The team member responsible for the Task estimates the size of the Task in hours, usually every Task takes between 4 and 16 hours. The decomposition from backlog items (which are estimated on a relative scale in Story Points) to Tasks ensure that the progress of the team during a Sprint can be tracked more carefully and the requirements for the Scrum team are more clear.

The Sprint Planning meeting's timebox is set at 8 hours for a one month Sprint. This timebox is decreased proportionally as the Sprint duration is decreased; a two-week Sprint would have a four-hour Sprint Planning meeting.

The Grooming Meeting
To ensure that the Sprint Planning meeting is finished within its timebox Scrum teams often hold Grooming meetings. Although not officially part of the Scrum framework, Grooming meetings have become a standard component of a Sprint and most experts, including co-founder Ken Schwaber, agree on its usefulness (ref). The Grooming meeting is used to crystallize items on the Backlog; descriptions, requirements and acceptance criteria that are not clear to the development team can be discussed and clarified so that the Sprint Planning meeting is efficient as possible.

Normally the Product Owner is the only who manages the Product Backlog, the Grooming meeting is the preferred moment to involve the entire Scrum team in this process.

The Daily Scrum or Standup
Every day the team gets together for the Daily Scrum or Daily Standup, during this meeting every team member answers three questions: What did I do between this Standup and the previous one? What am I going to do before the next Standup? What's keeping me from completing my work (also known as Impediments in Scrum)? The goal of this meeting is that all team members are continuously aware of the status of the Sprint and what their team members are doing. The Daily Scrum has a timebox of 15 minutes.

The Sprint Review or Demo
At the end of the Sprint the completed work is demonstrated to the Product Owner and other stakeholders during the Sprint Review or Demo. This is simply a summary of all items that have been finished in the Sprint, along with a demonstration of those particular items. This is also the meeting during which the stakeholders can give their input on the Product Backlog. A User Story is considered 'Done' when the Product Owner accepts it, usually that is when the Story complies fully with the Acceptance Criteria that were written down for the User Story and the Definition of Done (DoD), which is a list with general requirements for the User Stories (for example, the software component needs to be unit tested). These artifacts will be explained in more detail below. The Sprint Review has a four-hour timebox for a one-month sprint and is decreased proportionally.

The Retrospective
After the Demo the Scrum Team holds a Retrospective, a meeting during which the team members reflect on the past Sprint and look for things to improve on in the following sprint. Three items are discussed during the Retrospective: What went well during this sprint? What didn't go well during this Sprint? What should we do better in the coming Sprint? It is this inspect-adapt element that is very characteristic to Scrum. After the Retrospective the entire process simply starts over again with a new Sprint. The Retrospective has a timebox of four hours which is decreased proportionally as the Sprint duration is decreased.

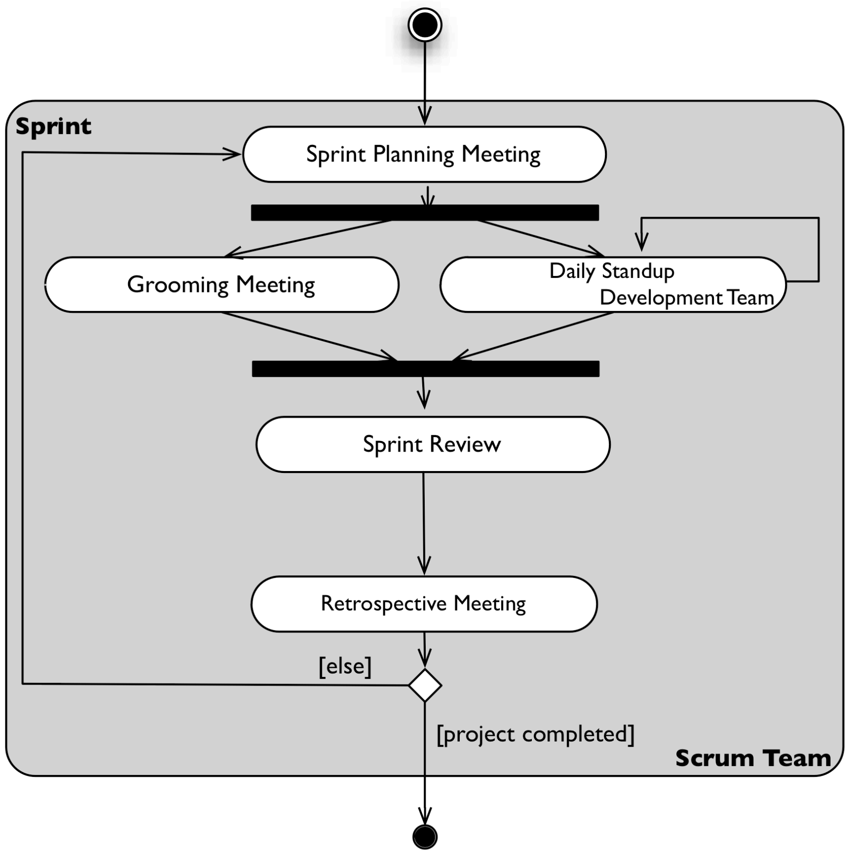


Figure 6. Activity Diagram highlighting the Scrum processes.

3.5. Scrum Artifacts

As with the processes mentioned in the previous paragraphs Scrum uses a set number of (hierarchical) artifacts, the most prominent of which is the hierarchy that divides the software components the development teams work on in different levels of granularity. This hierarchy of artifacts consists of:

I. Releases

II. Sprints

III. Backlog Items

IV. Tasks

Where Releases represent the largest level of granularity and Tasks represent the smallest level of granularity.

Release
A release occurs when the combination of work that was achieved in a number of Sprints is taken into production and accessible to the customers. There is no clear prescription for how many Sprints should make up one Release. In practice this number will vary for every organization, when the benefits of taking the newly added features into production outweigh the cost of the actual deployment (Schwaber & Beedle, 2002). As such a Release has a Start Date and an End Date (which depends on the amount of Sprints making up the Release).

Sprint
A Sprint is an iteration of work during which an increment of product functionality is delivered. As mentioned above such an iteration typically takes between one week up to a month. As such the Sprint has

a Start Date and an End Date. The product functionality that is committed to in a Sprint is gathered in the Sprint backlog and is usually briefly summarized in the 'Sprint Goal'.

Backlog Items
The increment of product functionality which is delivered during a Sprint is divided into multiple Backlog Items. During the Sprint Planning meeting these backlog items are moved from the Product backlog to the Sprint backlog. A backlog item can take a variety of forms, one commonly used is the User Story, another is a Defect or Bug. A backlog item typically has a number of properties depending on its type, for instance support materials like UI mock-ups. A Backlog Item has one or more Acceptance Criteria, requirements the Product Owner compiles which decide whether a Backlog Item can be considered 'done' at the end of a Sprint.

A Backlog is an ordered list of items, the priority of these items is context-dependent, risks are taken into account, the business value, dependencies on other items, customer requests et cetera. A Backlog Item's size is estimated in a relative value called Story Points. The teams decides what amount of Story Points to assign to a Backlog Item during the Sprint Planning.

Task
A Backlog Item is broken down into more concrete 'pieces of work' called Tasks. Contrary to Backlog Items, Tasks are estimated in hours, the average size of a Task lies between 4 and 16 hours. Every time a team member works on a Task he or she updates the remaining amount of hours for that particular Task. This allows teams to accurately calculate the amount of work remaining within a Sprint.

Displayed in Figure 7 is a meta-model of the Release Entity hierarchy, which is a summarization of the entities and their properties as mentioned above.

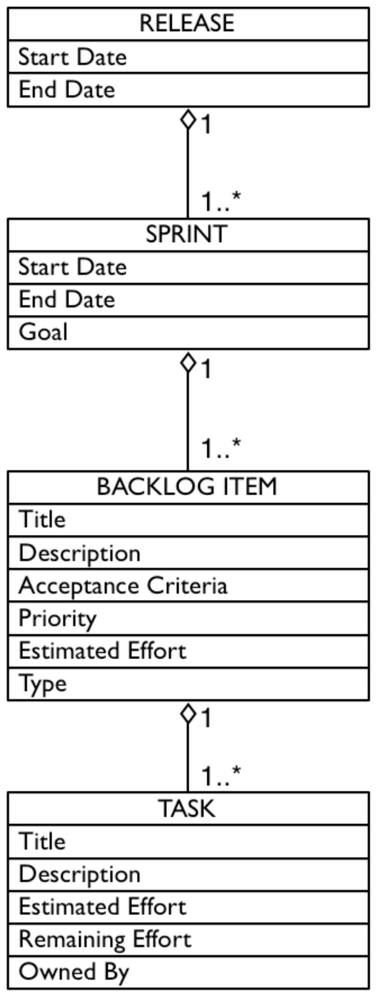


Figure 7. Meta-model of the Scrum Release hierarchy.

Besides the hierarchy of artifacts, other notable artifacts include:

I. Product Backlog

II. Sprint Backlog

III. Burndown Chart

IV. Scrumboard

V. Definition of Done

VI. Definition of Ready

VII. Velocity

VIII. Acceptance Criteria

IX. Epics

X. Impediments

Product Backlog

The key artifact of the Scrum framework is the Product Backlog. The Product Backlog is an ordered list that contains partitions of the software product, called Backlog Items, usually written in a particular format, known as User Stories. The Product Owner is the one who decides which of these items on the list have the highest priority (in coordination with customers and other stakeholders).

Sprint Backlog
The Sprint Backlog contains all the work that the Scrum Team commits to for a single iteration. The Sprint Backlog is The size of every item in the Backlog is estimated by the team. This size, or effort, usually measured in Story Points, is estimated by the team as a whole, every team member gives his or her estimation of the size of the Backlog item (commonly using the Fibonacci range). The estimated effort of a Backlog Item is accepted when consensus on the size of the item has been reached. This is why it is important that a Scrum team is of sufficient size, as estimations become more accurate when more team members factor in their estimations.

Burndown Chart
Backlog Items are broken down further into Tasks, chunks of work of about 4 to 16 hours, to increase traceability of the work during a Sprint. Two artifacts are most commonly used to trace the progress of the Sprint, the Burndown Chart is a graph with the remaining amount of work on the vertical axis (in Story Points or hours), the days of a Sprint are shown on the horizontal axis. When the team members 'burn' work, i.e. when they finish a particular Task, the amount of work remaining is reduced. The amount of work left is compared with the ideal amount of work remaining:

*[the total amount of work]-[the total amount of work] / [amount of working days] * [the current Sprint day]*

An example of a Burndown Chart is displayed in Figure 8.

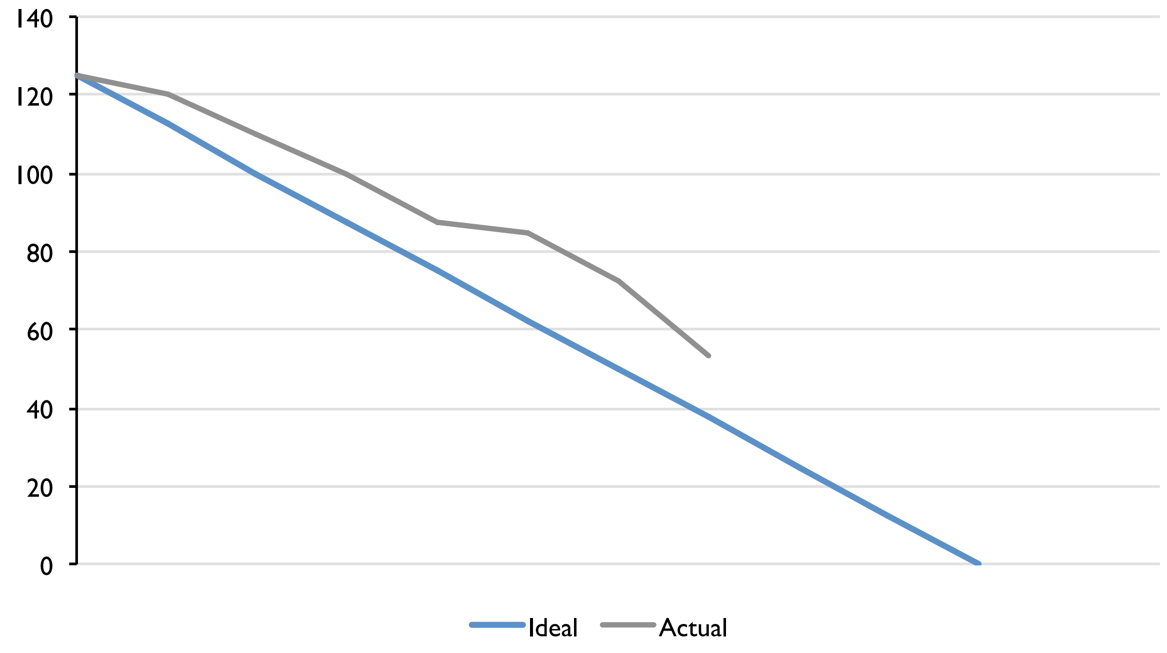


Figure 8. Burndown Chart example.

Scrumboard
A *Scrumboard, Taskboard* or *Scrum Wall* is the other method which is often used to visually illustrate the progress of a Sprint. The Scrumboard is usually just a piece of paper on a wall with sticky notes representing the Tasks of the Backlog Items in various states of progress (i.e. *to do*, *doing* and *done*). Every time a team member starts working on a Task he or she moves the Task to the appropriate state. The high priority Backlog Items are listed at the top of the Scrumboard thus the Tasks move around the board in a fan-like pattern.

Definition of Done
The Definition of Done is a set of general guidelines that apply to all backlog items to determine when they can be considered 'done'. For instance, an item that could belong on a Definition of Done is: 'all written code must have a unit test coverage of at least 80%'.

Definition of Ready
Only User Stories which are 'ready' are placed on the Sprint Backlog. This 'ready' state is usually determined by the Definition of Ready (DoR), which is a list of requirements a backlog item should adhere to (for example, when working on the UI a mock-up should be included).

Velocity
After a number of Sprints the Velocity (the total amount of Story Points a team can deliver within a Sprint) will become clear, enabling the team and the Product Owner to plan which items they can deliver in every Sprint.

Epics
Epics are User Stories which are too large to pick up in a single Sprint.

Impediments
An impediment is something that prevents a Scrum team member from finishing his work. An impediment can take a variety of form, be it lack to proper infrastructure, or unclear specifications, or time constraints. The Daily Standup is used by team members to announce their impediments, it is up to the Scrum Master, along with the rest of the team, to resolve these blocking issues so that work can be completed as required.

3.6. Additional Requirements for Scrum Tools

The following paragraphs describe the attempt to find additional requirements for Scrum tools through scientific studies on such digital aids, rather than by describing the Scrum framework earlier in this chapter. Very early on during the literature review it became clear that tool support for distributed Agile software development teams is a field that has seen virtually no scientific coverage so far. The review was started with a very narrow search scope, which was gradually broadened to extend the amount of search results retrieved for the search query. Initially the focus was solely on studies of tools, thus specific search queries were formed. Furthermore the queries were only aimed at the titles of the papers (the operator *allintitle:* was used to achieve this). Due to the lack of search results this focus was quickly extended to studies on distributed agile development where tooling was also discussed not just in the title, but anywhere else in the paper as well. See the table below for a impression of the search query, it's specificity, the resulting papers and the relevant papers.

Search Query	*Specificity*	*No. Of Results*	*Relevant Papers*
Allintitle: Scrum tool requirement	Very high	-	-
Allintitle: Scrum tool	Very High	8	<5
"distributed Scrum tool" and similar	Very high	Up to 10	-
"distributed Scrum" tool support	High	Up to 200	<10
Scrum tool	Low	>10.000	<30

Table 1. Literature search query results.

Despite broadening the search query to allow for a greater pool of studies to choose from, the original expectation with respect to the amount of scientific literature on the subject proved to be met; the amount of relevant papers was poor at best, and thus the scope was broadened to include non-scientific sources.

3.6.1. Defining a Scrum Project Management Tool

A pattern quickly emerged in the papers that were found: The resulting set of papers and studies mostly consisted out of case studies revolving around the design and roll-out of Scrum tools that addressed a particular component of the Scrum process. Examples of these components are, for example, a planning poker[1] tool (Morgan & Maurer, 2006; Petersen & Wiil, 2008; Pinna, Mauri, Lorrai, Marchesi, & Serra, 2003; Wang & Maurer, 2008), a Scrumboard tool (Engum, Racheva, & Daneva, 2009), a management reporting tool (Møller, Nyboe, Jørgensen, & Broe, 2009), and so on. No paper attempted to define the entire Scrum process and map it to a set of requirements for a tool. In fact, most of the studies found aren't triggered by a set of pre-defined requirements at all, rather they seem to describe studies that were initiated on an ad-hoc basis. We presume that this is caused by the fact that collocated teams require less tool support than distributed teams do. The fact that the Agile/Scrum principles advise against using tools most likely doesn't contribute to the development of a tool ecosystem either.

Although interesting, these types of studies into specific components only illustrates a limited scope for the complete set of requirements for what the researcher perceives as an Agile tool, a software product that encompasses all procedures and artifacts of the Scrum process. There appears to be a discrepancy in the general perception of a Scrum tool versus which functionalities we would consider such a tool to include. Which is why, from here on, the distinction between Scrum tools and Scrum Project Management Tools (SPM tools from here one) is made. An existing definition for an SPM tool, one that describes which characteristics such a tool should have, could not be found.

This lack of definitions for Agile tools immediately poses a problem. The term 'Scrum tool' is used as an umbrella concept in literature. There is a blurry dividing line that separates different clusters of tools which are used by organization at different maturity levels in their Agile adoption and other contextual factors (i.e. organizational size, collocated or distributed teams). Ranging from the simple online scrumboard tools, to Agile suites that support all Scrum processes like the case tool, to fully fledged Application Lifecycle Management tools (such as Microsoft's Team Foundation Server). This research is trying to establish the requirements a SPM tool should adhere to, however, where to we draw the line with respects to the functionalities of the tool? At which point in time do we cross the boundary from a Scrum project management tool into the realm of Application Lifecycle Management tools? Should coding and repository functionalities be included? Or is that beyond the scope of such a tool? And what about testing?

We therefore pose that a SPM tool should support its users in all processes of Scrum, it takes a more holistic approach, whereas a Scrum tool can be considered a software based solution that automizes/supports a particular element of Scrum. Application Lifecycle Management tools extend this functionality greatly by not only focussing on project management, but also on other aspects of software development, such as build and configuration management, deployment and testing and quality assurance. This leaves us with the question which elements of Scrum should be present in a Scrum project management tool. For now we take a purist view for the scope of a SPM tool. From the *Theoretical Background - Scrum* section of this paper we distinguish the following roles, processes and artifacts of Scrum:

Within the SPM tool a distinction can be made between three different roles: The Scrum Master, Team Member and Product Owner. Furthermore the tool should provide support for the five main meetings in Scrum: The Sprint Planning Meeting, Sprint, Daily Scrum Meeting, Review and Retrospective. Finally the Scrum product management tool should allow its users to create store and maintain the four major artifacts in Scrum: the Product Backlog, Sprint Backlog, Sprint Burndown Chart and Release Burndown Chart. Further interviews with Scrum experts, both members from Scrum teams as well as certified Scrum trainers should help in further narrowing down the definition of a SPM tool.

3.6.2. General findings of SPM tool usage

The general tenure towards SPM tool usage appears to be a cautious or negative one. The literature suggests that the trigger to use a tool tends to come from higher level management, such as project managers overseeing multiple Scrum teams, as managers look for ways to create transparency and uniformity in Scrum projects (Kurpicz, 2011). In 2010 research organization Forrester performed an evaluation of vendors of what they name Agile Development Management (ADM) tools (West & Grant, 2010). From their respondents they found that tools become a necessity when organizations start scaling Agile, when Scrum moves beyond the initial pilot team(s) to become the standardized way of developing software. Working with whiteboards and post-its works for single teams, but when Agile is adopted by more then one team tools, or rather automation, quickly become a must. Tools can bring a level of transparency,

1 Planning Poker is a technique that is used to estimate the relative workload needed to complete a User Story.

the ability to see the bigger picture, while this is rather difficult using traditional tooling, such as whiteboards and Excel sheets, see also Kivaisi (2010).

One study conducted by Azizyan, Magarian and Kajko-Matsson (2011), sponsored by Swedish telecommunications giant Ericsson, was specifically initiated to make visible how many organizations use Scrum tools and the reasoning behind the tool they chose to employ. The organization needed an independent and objective study to base their Scrum tool decision-making process on, not a study that was initiated by a Agile tool vendor. Although the research was sponsored by a commercial organization, Ericsson's goal was to map the Agile tooling software ecosystem to help pick the right software for their own teams, which assures its objectivity. The study was conducted using an online survey that was eventually filled in by 121 respondents from 120 companies coming from 35 different countries. It must be noted that the research does not focus on Agile tool support for distributed teams, but rather on Agile tools in general.

Out of the 121 respondents, 27% claimed to be working in a distributed team. The respondents represented a variety of different roles within their respective organizations, from developer to managing director. Interestingly, and confirming the text in the introduction regarding Scrum as the most prevalent Agile method, 54% of the respondents reported using Scrum as their Agile methodology, while 32% reported using a combination of Scrum and XP[2]. A survey conducted by SPM tool vendor VersionOne in 2011 confirms Scrum's dominance, 52% of the respondents reported using Scrum, 14% reported using a Scrum/XP hybrid (VersionOne, 2012). Continuing with Azizyan's results, 74% of the respondents claimed to be using a digital tool (ranging from Excel sheets to dedicated software suites) to support their Agile process. 61% of all collocated teams reported using physical tools (e.g. a physical Scrumboard), while for distributed teams this percentage was below 30%. Similarly the results showed distributed teams were more likely to use a software solution as compared to collocated teams (close to 90% for distributed teams versus 65% for collocated teams). 31% of all collocated teams reported to be using both physical and digital tools.

Azizyan et al. provide some insight into the process behind deciding upon which Agile tool to use within an organization (Azizyan, Magarian, & Kajko-Mattsson, 2011). The interesting conclusion from this study is that none of the tools that were investigated were found suitable for the company that commissioned the research. According to the researchers this is due to a number of reasons: although a large number of tools and their features were researched, they found it to be impossible to illicit all the requirements for the tool from the stakeholders involved at the case company. In retrospect, finding a tool everyone from all levels in an organization likes from the start might be a bit delusional. Besides the relative subjective aspect of stakeholders 'not liking' the tool, they also found that none of the tools tested were able to support the case company in all of their processes. This a more interesting conclusion; Agile implementations come in a wide variety of shapes and sizes, and a large number of organizations use hybrid forms of Agile implementations (VersionOne, 2012). Which might also explain the large number of 'home-grown' or in-house developed Agile tools being used throughout the industry (22% according to VersionOne (2012)), as no tool will support these hybrid forms out of the box.

This type of conclusion is typical for the field of Agile software development. Although the development process is relatively uniform and guided by fairly strict guidelines, there appear to be two distorting factors that contribute to the type of findings Azizyan et al. uncovered. On the one hand many organizations use a hybrid form of Agile, in Scrum terminology also called 'zombie Scrum', or 'Scrum-but', where not all of the artifacts and processes are implemented (without modification). Usually this is caused by the fact that Agile methodologies ask for a major switch in the way people within organizations perform their work. This is of course a great cause for resistance, and so, inevitably, remnants of 'the past way of doing things' tend to stick around for a while. This sounds even more counter-productive when you take into account that Scrum works better when it is executed according to its guidelines (Sutherland, 2010). On the other hand the rest of the organization (management, operations) might not be well-adapted to the Agile processes, processes throughout the organization which have existed for decades are ill-suited to fit with Scrum's iterative base. This of course leads to different requirements for the tool support from different levels within the organization.

2 EXtreme Programming is an Agile software development methodology that is focusses specifically on the programming aspect of software development.

3.6.3. Requirements for Agile tools

As mentioned earlier it was found that studies and surveys that have been conducted to investigate the requirements for Agile/Scrum tools were mostly sponsored by specific software vendors such as (Behrens, 2006; TargetProcess inc., 2008; VersionOne, 2012), which is a reason to doubt their objectivity with respect to what functionality their users require.

Azizyan et al. in their survey not only asked which Agile tools participants were using, they also provided the participants with 5 pre-chosen characteristics of Scrum tools that they estimated provided the most value to the users of the tool (Azizyan, Magarian, & Kajko-Matsson, 2011). They also used an 'other' field where the participants could fill in other characteristics they felt were important when deciding which tool to use. In order of importance these characteristics are: *ease of use*, *customizability*, *price*, *availability of reports* and *integrations with other systems*. In the *Other* field the participants noted some more functionalities and features they value highly, in order of importance (and grouped): *The ability to generate reports*, *the integration with existing systems*, *virtual Scrumboard features*, *interface improvements*, *project status tracking*, *ease of use*, *flexibility*, *budget tracking*, *requirements management*, *comprehensibility* and *planning elements*. It must be noted that these characteristics are very high level, this study is more interested in the requirements for functionalities of the tool, workflows, and the translation from a project management process (Scrum) to a digital support tool.

Using our definition for an SPM tool the requirements defined for the Scrum tools from the literature study can be incorporated and combined to form a set of requirements for a SPM tool.

3.6.4. Functional Requirements

From a holistic perspective the findings from the literature review are inconclusive. Most studies underline only particular components of the Scrum process. Using interviews, direct observation and a technical and functional analysis of the tool, the extent to which the case tool complies with these requirements will be analyzed. The general functional requirements that were derived from the literature are that the tool should support:

I. Releases, Sprints, User Stories and Tasks (Pinna, Mauri, Lorrai, Marchesi, & Serra, 2003).

II. Product Backlogs, Release Backlogs and Sprint Backlogs (Azizyan et al., 2011).

III. Sprint Planning meetings, Daily Standup meetings, Grooming meetings, Demo or Review Meetings and Retrospective meeting (Azizyan et al., 2011).

IV. Identifications and creation of interdependencies between User Stories and relationships between bugs and User Stories and Tasks (Azizyan et al., 2011; Engum, Racheva, & Daneva, 2009).

V. Team management, i.e. by allowing the user to invite team members, assign roles et cetera (Azizyan et al., 2011).

VI. Exchanging and storing project information (specification files, visual mock-ups et cetera) for longer durations of time (Kurpicz, 2011).

VII. Transparency into the status of the team/project, the progress of work, for instance through a Burndown chart (Azizyan et al., 2011; Kurpicz, 2011).

3.6.5. Non-Functional Requirements

The general non-functional requirements that were derived from the literature are that the tool should support:

I. The ability to collaborate, i.e. to work on items together (Kurpicz, 2011; Morgan & Maurer, 2006).

II. The use of visual metaphors to assist the transition from physical process to a tool based process (Azizyan et al., 2011; Behrens, 2006; Petersen & Wiil, 2008; Wang & Maurer, 2008).

III. The need for 'agility' in operability, for example by allowing remote access (Azizyan et al., 2011; Pinna et al., 2003).

IV. Modularity and connectivity, the tool should support connection with other software development tools already in use by the development teams (Azizyan et al., 2011; Pinna et al., 2003).

V. Documentation and support allowing for an easier transition to a tool based workflow (Kurpicz, 2011).

3.7. Concluding

This chapter is concluded by answering the first sub-question of this study: *What is the status quo of distributed Scrum tooling practices in scientific literature, and which processes and entities of Scrum can we derive from literature?* The scientific coverage of tool support for Scrum teams, and distributed Scrum teams in particular is limited. Studies of usage of Scrum has naturally increased in the last decade as the popularity of the software development method has increased. Some studies focus specifically on the performance of distributed Scrum teams (such as Sutherland, Viktorov, Blount, & Puntikov (2007), Herbsleb & Mockus (2003) and Hole & Moe (2008)), however, none of these studies touch upon the subject of tool support for these teams in an extensive manner.

Broadening the scope to Scrum tools in general, dropping the focus on distributed Scrum, we see a fragmented landscape of definitions and types of tools. The term 'Scrum tool', which was originally the moniker used in this study, turned out to result in an overly broad spectrum of tools, from simple single-use Planning Poker tools, to fully-fledged Application Lifecycle Management tools that require substantial investments.

A set of functional and requirements mostly pertaining to the concepts that should be identified by a Scrum Project Management Tool was derived from the general Scrum literature. A complimentary list of additional requirements for Scrum tools in specific were derived from studies performed by others. The results of this section will be used in later chapters to analyze to which degree the case tool supports important Scrum concepts.

4. Tool Analysis

The following chapter starts with a basic overview of the case tool, in which form it is available to its users and how it is structured. We then take a more detailed look at the functionalities of the tool, and how these functionalities are decomposed into specific features, or *apps* as they are referred to by the case company. The concepts supported by the case tool are then compared with the Scrum concepts defined in the previous chapter to analyze which features are supported and which features are not.

4.1. TOOL

As noted earlier the case tool is a web-based tool, users access the tool by navigating their browsers to the application, where they can log in with their company e-mail address and a password. An on-site and offline version of the case tool, which can be installed on an organization's intranet, is planned, but currently unfinished.

The case tool has a relatively flat account hierarchy. A single person from an organization creates the initial account. This person automatically becomes the Account Owner, a user role that allows this person to perform specific administrative purposes, such as upgrading the license, removing users and deleting the company account. The Account Owner can create new project teams, each of which has a separate instance of the case tool with all its features and functionalities.

There is a single pool of users for every account, the maximum number of users in this pool is limited depending on the license the organization purchased. The users who are in this pool can be assigned to multiple teams. That means that when a user is removed from all of his or her teams, she will still be part of the account user pool. There is no limit to the amount of teams the Account Owner can create. There is, however, a 500 Megs file limit per user, although this limitation is not actively enforced at the moment.

The case tool is built on a SQL server, which stores the team's Scrum project data. It is this SQL server data which is used for the case tool usage analysis. A combination of frameworks and technologies such as ASP .NET, JQuery, WCF, Javascript and HTML is used to build the tool. The majority of the functionalities are built in-house, a number of third party plugins is used for functionalities such as graph and chart rendering.

4.1.1. The goal of the tool

The goal of the case tool is 'to help (distributed) Scrum teams deliver better software and improve themselves, by supporting all aspects of Scrum, with a special focus on human interactions'. With respect to the distribution of the team members it does not make a difference whether these team members are distributed over several countries, or whether they are located within the same building and simply on different floors.

4.1.2. Features

[The description of the features have been removed in this public version of the paper.]

4.1.3. Reflecting on the features

It is clear that the person responsible for designing the case tool placed a large emphasis on making the tool as straight-forward to navigate through as possible. Dividing the case tool's functionality in separate apps makes sense from a usability perspective as it allows users to easily navigate to the distinct features of the tool. What is remarkable is that the division of functionality into separate apps does not appear to be dictated by the Scrum framework. To illustrate, some Scrum concepts and processes have 'their own' app, such as the Backlog, which is a Scrum artifact, and the Standup, which is a Scrum process. Inquiry at the people responsible for the development of the apps made it clear that the other apps were created based on the offshoring experience the case company has accumulated over the years. Enabling Scrum teams to work efficiently together through a digital tool requires an additional layer of supporting processes and artifacts on top of the ones identified in the Scrum handbooks. We will reiterate on this notion later on in this chapter.

4.2. Functional Architecture

In order to illustrate how the above mentioned features work together, the components of the case tool and the relations between those components are visualized in a Function Architecture Diagram (FAD). A FAD is a Functional Architecture Model (FAM) which is used to represent the "...primary functionality of a software product, consisting of its main functions and supportive operations" (Brinkkemper & Pachidi, 2010). The main purpose of a FAD is to offer insight into the specific functionalities of the tool and the interactions between these functionalities. The approach as defined by Brinkkemper et al. in (Brinkkemper & Pachidi, 2010) was used to create the FAD for the case tool:

I. *Determine the scope*, the scope for this tool is relatively clear seeing as it is a web-based cloud tool, it doesn't operate within a suite of other dedicated software.

II. *Define request-feedback flows*, the request-feedback flows show how the different components interact with each other.

III. *Model the operational flow,* the operational flow models the processes that take place in the tool to derive output from the input of the tool.

IV. *Add control and monitoring modules*, when the main operational flow is modeled possible supportive modules are identified and added to the tool.

V. *Specify external to/from internal interactions,* during this final step we model the interactions between internal components and external components and processes.

Keeping these steps in mind, a FAD of the case tool was created, as can be seen in Figure 20 below. The requirements for the software project/product are taken as input and the iteration of working software is used as output. The outcome shows, unsurprisingly, that the case tool mainly consists out of support components, loosely connected parts of software that assist the teams and other stakeholders in various aspects of their working process with Scrum.

We define two main processes: *Backlog Management* and *Sprint Management*. These two processes or functions of the tool are mainly supported by the Backlog app, as is explained earlier in this chapter. Backlog management can be considered as the process during which requirements from the customers are translated into Backlog items and are prioritized and placed on the Product Backlog by the Product Owner. During the grooming meeting the Backlog is re-prioritized with the help of the Scrum team, while the Product Owner changes the Backlog on a continuous basis. During the Sprint planning a Sprint Backlog is created within the case tool and Backlog items are moved from the Product Backlog to the Sprint Backlog.

Manipulations to the Backlog are shown on the Dashboard of the application through Interactions. The application also calculates a number of Key Performance Indicators which can be used by the Product Owner and/or other stakeholders to manipulate the Backlog. For instance, when the Burndown metric shows that the remaining amount of work left in a Sprint exceeds the estimated amount of work, the Sprint Backlog can be reduced. The Standup functionality is used to record the Daily Standup sessions. When things are amiss in a Scrum team these issues tend to surface during the Daily standup, hence the Standup functionality provides a trigger to both Backlog management as well as Sprint management.

The Video functionality is strictly a display of a live IP camera stream and as such does not provide any explicit functionality to the tool. The same goes for the Team Management functionality, this allows the users to add and remove team members to the team and to change the roles of these team members.

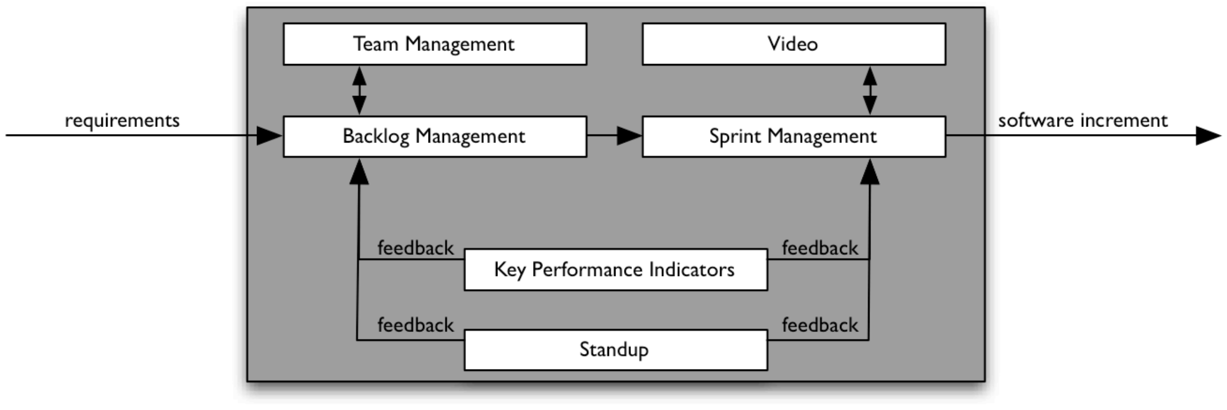


Figure 20.The Functional Architecture Diagram of the case tool.

4.3. Comparing TOOL with literature findings

In the following paragraphs the features offered by the case tool will be directly compared to the concepts found in the Scrum literature and the scientific sources as discussed in the third chapter of this study. For each of these concepts the degree to which they are supported by the case tool is evaluated and graded on a five-point scale (++ to --).

4.3.1. Scrum Roles

From the analysis of fundamental Scrum concepts in the previous chapter a list of roles used in Scrum projects was defined.

Concept	*Support*	*Notes*
The Scrum Team	+	All four basic Scrum roles are supported, the users can be assigned to these roles through TOOLs Team Management module. Despite this, the roles do not have specific right and/or abilities, everyone can edit the Backlog for example, and not just the Product Owner.
The Development Team	+	Idem.
The Scrum Master	+	Idem.
The Product Owner	+	Idem.

Table 2. Support of Scrum roles.

TOOL also recognizes these roles, and adds four additional ones: *the Account Owner*, *the Administrator*, *the Stakeholder* and *the Whomsoever User*. The Account Owner is the person who creates the initial organization account. The Account Owner is a role that was specifically created because using an electronic tool requires a 'superuser' with some administrative capabilities. The Account Owner is the person who can upgrade the license of the tool, remove users from the tool, et cetera. The Administrator is a user who can create and edit teams. Both the Stakeholder and the Whomsoever User roles are created specifically to designate people from outside the Scrum team who would still require to be invited to the tool, such as customers.

Apart from the Account Owner, there does not appear to be a real functional difference between the various roles in the case tool, the distinction of roles is purely of a visual nature. The fundamental responsibility of the Product Owner, for example, which is managing the Product and Sprint Backlogs is not supported by the case tool. Everyone in the team appears to have full read and write access rights to the Backlog functionality of the tool, even the people with the Stakeholder and Whomsoever User Roles. This is the main reason why the tool did not score the highest rating for the support.

The only real functional difference between these various roles in the tool is that it creates a hierarchy for user management, even though the Scrum roles weren't conceived to create a hierarchy within teams, they were made purely to designate who does what during the process. In the case tool the following hierarchy can be identified: Account Owner, Administrator, Product Owner, Scrum Master, Team Member, Stakeholder, Whomsoever User. There appears to be no real reason for the existence of this hierarchy other than the fact that users can not manipulate team member information from users of a higher role. Only the Administrator, Product Owner and Scrum Master are able to invite new users to the teams.

All in all the user role and management implementation feels incomplete, the Scrum roles are there in the case tool, but they serve no real purpose. There are of course some use cases where it makes sense to allow other people to have access rights to the Backlog, for instance when the Product Owner is unavailable due to illness. A Scrum team is perfectly capable of taking responsibility of the Backlog in these cases. However, allowing everyone in the team to manipulate the Backlog appears to be a business risk and does not adhere to the Scrum guidelines.

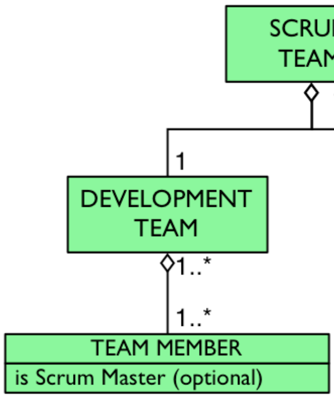


Figure 21. Scrum Roles supported by TOOL (in green).

4.3.2. Scrum Processes

From our analysis of fundamental Scrum processes in the previous chapter we defined a list of processes frequently used in Scrum projects (note that the importance of supporting these processes was also acknowledged by (Pinna et al., 2003) and (Azizyan et al., 2011)):

Concept	*Support*	*Notes*
The Sprint Planning Meeting	+/-	There is no explicit Sprint Planning functionality. The Backlog app can be used to support this process to a large degree however.
The Grooming Meeting	+/-	There is no explicit Grooming Meeting functionality. The Backlog app can be used to support this process to a large degree however.
The Daily Standup	+	The Daily Standup meeting is supported explicitly using the Standup app. Users can record and play back recorded Standup. Only one camera feed can be used.
The Sprint Review	-	There is no explicit Sprint Review or Demo functionality in the tool. TOOL does not have a feature that could partially support this process.
The Retrospective Meeting	--	There is no explicit Retrospective Meeting functionality in the tool. TOOL does not have a feature that could partially support this process.

Table 3. Support of Scrum processes.

The Sprint Planning Meeting, Grooming Meeting and Sprint Review are processes that are heavily involved with the Sprint, Release and Product Backlogs. Due to this the case tool's 'Backlog app' can be used to support these processes to some degree, but there is no explicit functionality that guides the users through these processes. Some actions have to take place offline, while others can be managed with the help of the case tool. In that respect it is a strange omission that raises the question whether the tool is intended to implicitly support the Scrum framework through a set of digital aids, by digitizing some of the artifacts that teams use in an offline setting, or whether the intention is to create a full suite of features by translating all Scrum concepts into a digital counterpart. Taking into account the goal of the tool, which is to create a Scrum project management tool with a focus on distributed teams, it is strange that some processes will have to take place offline, while others can be completed in the online environment.

The Retrospective Meeting is a very important process within Scrum, but is omitted in its entirety, whereas the other meetings could still be supported using some of the functionality of the case tool. The Retrospective is a major element of the 'inspect-adapt' adagio, one of the main foundations of the Scrum framework, which makes it lack of tool support even more peculiar. The concept of the Retrospective, summarizing at the end of the Sprint what went well, what didn't go well, and what should be improved during the next iteration, is a purely text-based one, which should make the retention of these meetings relatively simple.

Out of the total five Scrum sprint processes, only one is explicitly translated into a specific feature of the case tool, the Daily Standup. The '*Standups*' feature of the case tool allows the team members to record their Daily Standups, so they can be replayed, downloaded or e-mailed to another stakeholder at a later point in time. While the Standup is recorded the current Sprint's Burndown graph is displayed so the team is aware of the current state of the Sprint. Besides those features a number of statistics are shown, including the average duration of the Standup (which should take at maximum 15 minutes (Sutherland, 2010)), the average delay for the Standup to start, and who has missed Standups during the current Sprint. These statistics are interesting, because they are clearly meant to guide the Scrum team to follow the Scrum guidelines more strictly. As mentioned in the introduction to the Scrum framework, timeboxes and timezones are important concepts in Scrum, they are seen as necessary for teams to be able to reach their full potential (Schwaber & Sutherland, 2011). These Standup statistics appear to be the only initiative in the tool where the case tool actually tries to enforce a strict adherence to the Scrum guidelines. See Figure 22 for an overview of the Scrum processes and by which case tool features they are supported.

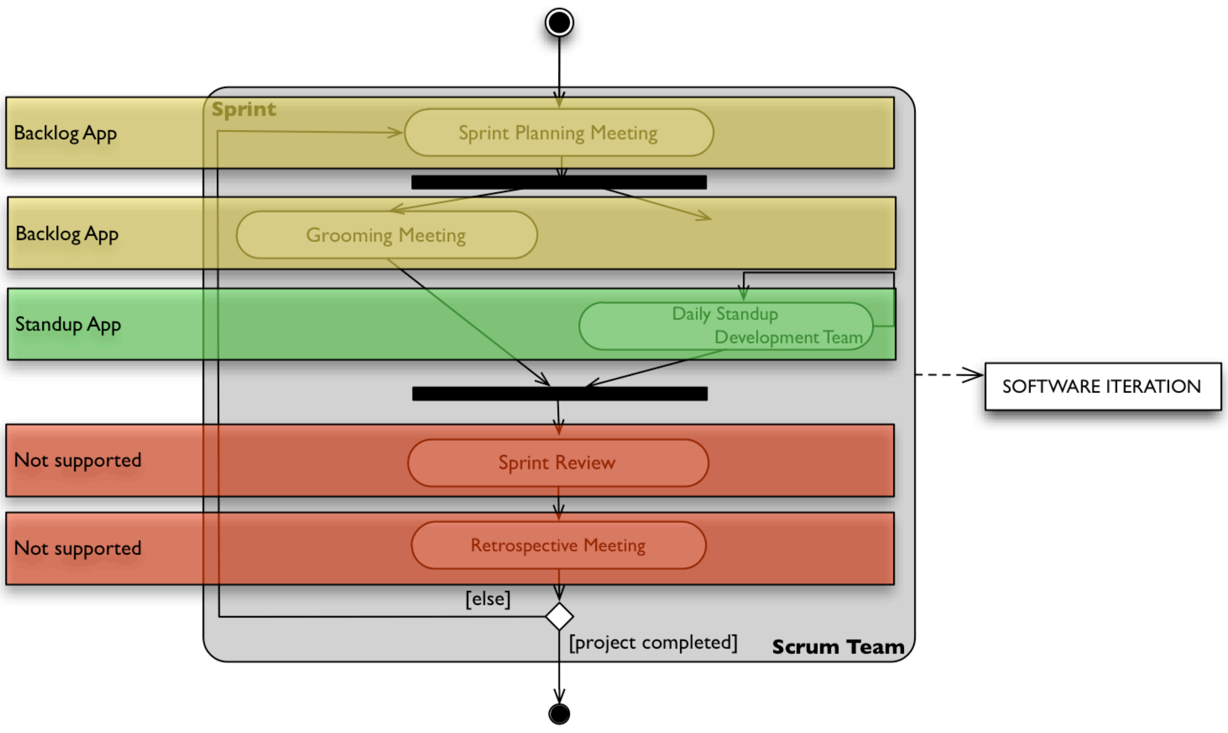


Figure 22. Scrum processes and by which case tool features they are supported.

4.3.3. Scrum Artifacts

From our analysis of fundamental Scrum concepts in the previous chapter we defined a list of artifacts used in Scrum projects. First we will cover the Release Entity hierarchy from this list:

Concept	*Support*	*Notes*
Releases	++	New Releases can be defined in the Backlog app.

Concept	*Support*	*Notes*
Sprints	++	New Sprints can be defined and matched to a Release in the Backlog app.
Backlog items	+	Backlog Items can be assigned to Sprints and Team Members, most properties identified in literature are implemented. Acceptance Criteria are not explicitly supported, although users could write these down in the *Description* field. Creating relationships between Backlog Items is not possible.
Tasks	++	Tasks can be assigned to Backlog Items and Team Members, and all properties identified in literature are implemented.

Table 4. Support of Scrum Backlog Artifacts.

Both Releases and Sprints are concepts which are supported by the tool. The case tool supports two different types of Backlog items, User Stories and Bugs. As mentioned in the introduction to Scrum chapter, a User Story is a commonly used format to describe a software component. A Backlog Item has a number of Acceptance Criteria, composed by the Product Owner, which are used to judge whether or not a Backlog item has been completed successfully. These Acceptance Criteria cannot be added to the case tool. Every Backlog Item and Task does have a Description variable however, where the Product Owner is able to record the Acceptance Criteria, a work-around which was observed in the project data of multiple teams.

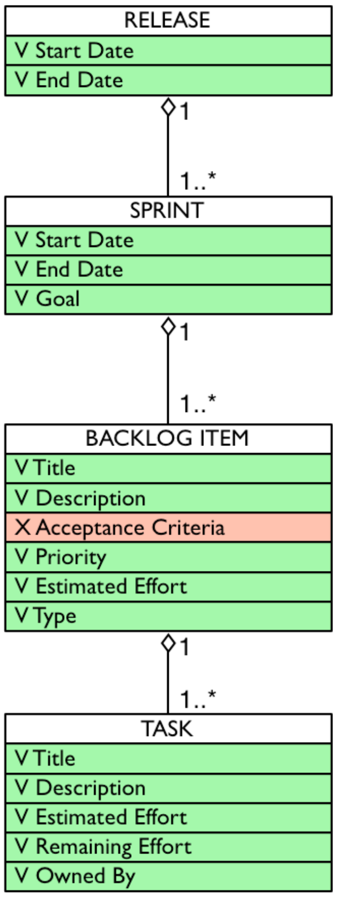


Figure 23. Scrum processes and by which case tool features they are supported.

Concept	*Support*	*Notes*
Product Backlog	+	Product Backlogs are implicitly supported in the Backlog.
Sprint Backlog	++	Sprint Backlogs are explicitly supported.
Release Backlog	++	Release Backlogs are explicitly supported.
Epics	--	Epics cannot be created in the case tool.
Impediments	-	TOOL does not have a feature that allows users to explicitly create Impediments. Some teams successfully use Backlog Items as a work-around.

Table 5. Support of remaining Scrum Backlog artifacts.

For every Release and Sprint TOOL allows its users to create Release Backlogs, Sprint Backlogs (and Product Backlogs). Both Backlog Items and Tasks can be added to these Backlogs. There appears to be no way to create relationships between User Stories other than the Release-Sprint-Backlog item-Task parent-child relationship. Both (Engum et al., 2009) and (Azizyan et al., 2011) acknowledged that this is an important characteristic of Scrum tools. There is no way to create dependencies between items, and as such bugs can not be linked to specific items, Impediments cannot be related to Backlog items, making the overall structure of the Product, Release and Sprint Backlogs less transparent.

Impediments, issues which block the Scrum team from successfully finishing their work, such as unclear work descriptions or lack of certain resources cannot be added to the tool. Some teams have been observed to use Backlog Items as a work-around, these teams give a Backlog Item a specific prefix to the title, such as 'IMPEDIMENT', so that it is clear to everyone why the Backlog Item is added to the Sprint. Impediments are defined during the Daily Standup, a Scrum meeting that *is* supported by the tool, which makes it more peculiar that these Impediments can not, after being defined, be registered in TOOL.

Concept	*Support*	*Notes*
Velocity	++	The Velocity of the Team is shown in the KPI app of the tool.
Burndown Chart	+/-	TOOL has a Burndown app, it however only shows Task Burndown charts, not Story Burndown charts.
Release Burnup Chart	--	A Release Burnup functionality is not included in the tool.

Table 6. Support of Scrum metrics.

The Velocity of a team is displayed in the KPI app of the tool, along with additional metrics not specifically mentioned in the Scrum guidelines. The application does have a Burndown chart, however it is solely a Task Burndown chart, it does not display the status of the Sprint on a Backlog Item level. As mentioned in the previous chapter, both of these Burndown charts are needed in order to get an accurate picture of the progress of the team (as noted by Kurpicz (2011) and Azizyan et al. (2011)). Whereas the Burndown chart displays the progress of a team over the course of a Sprint, the Release Burnup chart shows whether or not the Scrum team is on track to meet the requirements for an entire Release. TOOL does not have any functionality to support this artifact.

Concept	*Support*	*Notes*
Definition of Done	--	There is no functionality in the tool that allows users to save the Definition of Done.
Definition of Ready	--	There is no functionality in the tool that allows users to save the Definition of Ready.
Scrumboard	--	There is no explicit Scrumboard functionality in the tool. TOOL does not have a feature that partially supports this artifact.

Table 7. Support of other Scrum artifacts.

The Definition of Done and the Definition of Ready are two Scrum documents which are not supported by the tool.

The Scrumboard is missing from the case tool's list of features. A Scrumboard plays a major role in an offline based Scrum setting, because it allows the team members to easily see the current state of the Sprint, which Backlog items have been completed, which ones are still open, how many hours there are left to be burned and who is responsible for the items on the Scrumboard. The tool lacks such functionality completely. Arguably the users can use the Backlog feature to see the current state of the Backlog items, and the user can click on these Backlog items to see the state of the Tasks assigned to the Backlog items. This is a very unwieldy way of inspecting the progress of the Sprint, and most of all it doesn't give one the complete overview of the sprint within a single glance. Moreover, if distributed teams were to have separate physical Scrumboards they would soon run into synchronization issues.

4.3.4. Tool Characteristics from Scientific Literature

Most of the characteristics derived from scientific literature have already been discussed in the previous paragraphs, the ones which were not are described further here. The remaining functional requirements are listed below:

Concept	*Support*	*Notes*
Team management (Azizyan et al., 2011)	+/-	TOOL users can easily add or remove users through the Team app. Cross-team functionalities however are not present.
Exchanging and storing project information (Kurpicz, 2011uy)	-	Data on the project can be stored to some degree in Backlog Items and Tasks. This data is however not available from a centralized location.

Table 8. Support of functional requirements as identified in literature.

Azizyan et al. (2011) noted the importance of being able to manage the Scrum team. From changing the team composition, to being able to assign various roles to the members. Although these things are possible to some degree, the tool does not support any cross-team functionalities. Sharing a common Backlog or any other information between two teams is not possible and neither is it possible to assign a Task or Impediment to someone from a another team. Had the team management module been set up with a larger scope, the score would have been higher.

Exchanging and storing project information as noted by Kurpicz (2011) is only possible to a limited degree. Team members can assign additional information to Backlog Items and Tasks, and they can communicate with each other through the tool's Interactions Feed. This is however, where the data sharing functionalities end. There is no central repository where users can store and edit project specific data, such as the

Definition of Done, Definitions of Ready, or other important items, like project roadmaps. That means that it becomes increasingly troublesome to find and manipulate team related data. During the study it became clear that many teams use other solutions to this end, like shared drives, or cloud-based data solutions like Dropbox or Microsoft Sky Drive.

The remaining non-functional characteristics are summarized below:

Concept	*Support*	*Notes*
The ability to collaborate (Kurpicz, 2011; Morgan & Maurer, 2006)	+/-	Collaboration is possible in that team members can share responsibilities, and work together on a single Backlog. Team communication is encouraged through the Interaction Feed and the Video app. Working with multiple people on a single Backlog Item is not supported, and items cannot be shared across teams.
Usage of visual metaphors (Petersen & Wiil, 2008; Behrens, 2006; Wang & Maurer, 2008; Azizyan et al., 2011)	Not Suppor ted	There is no functionality in the case tool that lends itself for visual metaphors.
Operability agility (Azizyan et al., 2011; Pinna et al., 2003)	+	The tool is available through a SaaS license. There is no localized version available.
Modularity and connectivity (Azizyan et al., 2011; Pinna et al., 2003)	-	The tool can be connected to a Microsoft Team Foundation Server work item repository. Further connections are lacking.
Documentation and support (Kurpicz, 2011)	+/-	The tool has limited documentation available on its website, the documentation appears to be incomplete.

Table 9. Support of non-functional requirements as identified in literature.

The ability to collaborate as noted by Kurpicz (2011) and Morgan & Maurer (2006) is severely hindered by the lack of any cross-team functionality, as mentioned earlier. Multiple team members are also unable to work on a single item, making a true collaborative effort impossible. The lack of additional features like Backlog Item manipulation history also makes it more complex to collaborate on items.

The tool does not have any features that lend itself for the usage of visual metaphors, a characteristic that is seen as being important by Petersen & Wiil (2008), Behrens (2006), Wang & Maurer (2008) and Azizyan et al. (2011). Scrum has very little artifacts which are of a tangible nature that could be converted to a digital counterpart, the Burndown chart is present in very much the same form as it would be shown in physical form. The Taskboard would also lend itself perfectly for visual metaphors, but this functionality is not present. As far as text-based concepts go, the terms used in the tool largely agree with those used in Scrum literature.

As far as 'operability agility' as noted by Pinna et al. (2003) and Azizyan et al. (2011) is concerned, the tool is built on a Software-as-a-Service foundation, meaning that it can be accessed from every location, as long as the user has a PC with a network connection and a internet-browser. A localized version of the tool is missing from the portfolio and the compatibility of the tool with mobile devices is also poor, limiting the overall operability of TOOL.

With respect to Modularity and connectivity as noted by Pinna et al. (2003) and Azizyan et al. (2011), TOOL can be connected to Microsofts Team Foundation Server, which is a repository for work items that is used Microsoft Visual Studio. This it the only connection to third party software that the tool allows.

There is some documentation available on the website of the case tool. The information listed here appears to be lacking in that it covers not all features of the tool (such as the proper way to create Releases and Backlogs, or how to connect TOOL to Microsoft Team Foundation Server) and also appears to be outdated (the location of UI elements, for example, does not appear to match the location of these elements in the tool).

4.4. Concluding

To answer our sub-question *'how is the tool structured, which processes and entities does it support?'*, in short, in this chapter we have seen that the case tool only supports a limited set of the concepts we derived from Scrum literature, and the tool also seems to lack some of the tool specific features derived from scientific sources. Although the Scrum roles are also defined in the tool, the processes, and especially the artifacts, show only minimal support.

The (undesired) consequence of this is that the team members will have to resort to offline aids or other digital tools, as has been frequently observed during this study. The case tool's feature-set is an indication of an incomplete vision with respect to what TOOL should enable its users to do. The feature-set is simply incomplete. Elements such as the Standup app indicate that the goal of the tool is to translate the complete Scrum framework with all of its concepts into a digital format, making other tools and physical objects unnecessary, streamlining the workflow of distributed teams. However, the absence of some core aspects of the Scrum framework show that the case tool currently can not meet this goal.

Concept	*Processes*	*Artifacts*	*Functional Requirements*	*Total*
Support rated ++	0	6	0	6
Support rated +	1	2	1	4
Support rated +/-	2	1	3	6
Support rated -	1	1	2	4
Support rated –	1	5	1	7
Total	5	15	7	27

Table 10. The ratio of support for the identified concepts.

5. Analyzing Usage Data

This chapter describes the process of analyzing TOOLs database using the Goal-Question-Metric approach. What this means in practice is that certain pieces of data contained in the database are considered to be *metrics*. The data gives information about a particular element of the case tool. When combining these metrics, questions arise. For example, when having access to the team size metric and a metric that shows how successful Sprints were, the following question can be asked: '*which team size leads to the highest percentage of successful Sprints?*'. The resulting answer to this question might give insight into new features the tool could incorporate, or lacking or completely missing features. In this particular case, where we have a *team size metric* and a *Sprint successfulness metric*, the application could take a more agent-based role where it gives its users visual cues, or advice, as to what team size is most efficient.

The database used during this phase of the study is a clone of the database from the case tool that was created on the 15th of April 2013. The case tool has been in use since May of 2012, meaning that the copy of the database contains over 11 months of usage data. Some of this data was unusable, mostly due to the fact that a significant amount of organizations and individuals had used the tool for a very short duration of time as a trial. This data was discarded.

Below is a summarization of the remaining teams, the amount of Sprints they have logged in the database, the period during which they used the tool, and the amount of members the teams have.

Team	*Industry*	*Tool in use since*	*Sprints logged*	*Team size*
Team A	Telecom industry	08-2012 to 01-2013	13	4
Team B	Telecom industry	7-2012	22	11
Team C	Telecom industry	7-2012	22	3
Team D	Telecom industry	7-2012	27	11
Team E	Healthcare	8-2013 to 10-2012	8	4
Team F	Case tool development team	10-2012	13	2
Team G	Human Resource Management	12-2012	10	7
Team H	Aviation industry	2-2013	5	5

Table 11. The remaining teams, after cleaning up the database.

5.1. Contents of the Database

The data contained in the database mostly consists of records related to the iterations (Releases and Sprints), work items (Backlog Items and Tasks) and the Scrum Teams themselves. As the analysis progresses metrics from multiple tables are combined in order to create more interesting patterns.

Sprint length
The Sprint table contains all the Sprints currently logged in the case tool, their start-date and end-date and the corresponding Team ID. The most important metric that can be extracted from this data is the Sprint duration. See Figure 24 for a graphical depiction of the Sprint duration for the various teams. This metric raises three questions: All teams currently using the tool appear to be mostly using Sprints with a duration of two weeks (see Table 12), with an occasional Sprint duration of one week. In fact 23% of the Sprints had a duration of one week, whereas 77% had a duration of two weeks. In the guidebooks a Sprint length should

be 30 days or less, as long as the duration is consistent (Sutherland, 2010). So why are we only seeing Sprints with two different durations?

Team	*One-week Sprints*	*Two-week Sprints*	*Duration ratio*
Team A	1	12	(7,7% - 92,3%)
Team B	0	22	(0% - 100%)
Team C	1	22	(4,3% - 95,7%)
Team D	11	17	(39,3% - 60,7%)
Team E	8	0	(100% - 0%)
Team F	1	12	(7,7% - 92,3%)
Team G	3	7	(30% - 70%)
Team H	3	2	(60% - 40%)
Total	*28*	*94*	*(23% - 77%)*

Table 12. The Sprint duration per team.

Another thing of interest is that teams D, E, G and H each show a similar pattern (again, see Figure 24); the first few Sprints have an average duration of one week, after which the duration increases to two weeks. In fact, Team E has only done one week Sprints, why is this? A final pattern that can be discerned is the fluctuation in Sprint duration, as can be seen in the data from team A, C and F, where two week Sprints are followed by a single one week Sprint.

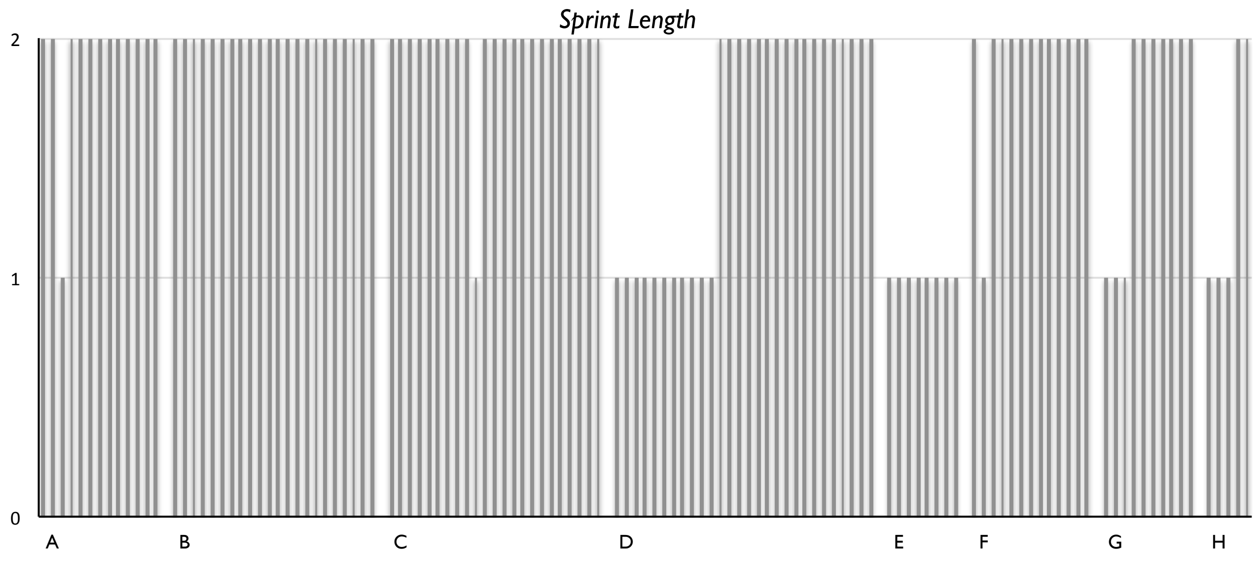


Figure 24. The Sprint length per team (in weeks).

After consulting with the Scrum teams it became apparent that the reason for the consistency in between teams - two week Sprints look to be the norm - is that, unless there are exceptional circumstances, all teams working at the case company start and end their Sprint at the same date. This is done to create a 'company heartbeat', it allows for easier management of the teams, from gathering team statistics and discussing those together on 'big sprint demo' where every team accounts for the amount of work the team was able to commit, to logistical and infrastructure issues. This pattern is then not a direct consequence of characteristics of the case tool, but rather of the case company. The fact that all Sprints appear to have the same duration makes it not possible to draw any conclusions on what Sprint duration might be more successful.

The second pattern, where we see that teams' initial sprints are shorter than their later sprints, is caused by the fact that new teams have shorter sprints in order to speed up the feedback loop with the sponsors of the project. During the first few weeks, especially when the client organization is not used to working with Scrum, Sprints are generally troublesome. In order to make sure that this phase goes as smoothly as possible, the cycle is kept short as short as possible, in order to allow more feedback, reflection and coaching. Once the customer and the team members are convinced of the Scrum team's skills and abilities the Sprint duration is lengthened. The other teams do not have these short initial Sprints, because the teams had been working as a Scrum team before they started using the case tool.

The other reason for teams to occasionally change their sprint duration is to respond to urgent issues, inquiry at Team F for example learned that when the web-based software product they were working on was under attack by hackers, the team changed their two-week sprint duration to a one-week sprint to implement some urgent security measures. Less notable examples for teams to switch sprint duration include cases where teams had to implement features that were deemed urgent by project management.

To summarize:

- The similarity in Sprint duration is the result of the case company's desire for a 'company heartbeat'.
- New teams tend to start with one-week Sprints to increase the feedback frequency with the client.
- In very rare cases teams deviate from their two-week Sprint pattern to be able to respond to urgent matters.

Backlog Items and Tasks
The goal of this research is to improve the effectiveness and efficiency of the teams using the case tool. In order to do so a way to measure the team's current performance must be found. The amount of Backlog Items and Tasks a team commits to per Sprint is a very basic metric that can be used to illustrate a team's performance. The data on these elements is combined with other metrics in order to create an increasingly accurate performance picture of the teams. The input from the users of the case tool is used in discussions on these emerging patterns during the analysis.

As can be seen in Figure 25 and Table 13 below, the amount of Backlog Items each team commits to in a single Sprint fluctuates significantly. Five out of eight teams display a positive trend, while the other teams show a negative trend towards the amount of Backlog items committed to in every Sprint.

Team	*Average Backlog Items*	*Standard Deviation*	*Trend*
Team A	11,4	4,5	-0,05
Team B	16,8	7	-0,36
Team C	5,7	2,8	-0,1
Team D	13	6,1	0,28
Team E	5,3	2,2	0,17
Team F	14,4	7,3	0,85
Team G	14,6	10,1	2,31
Team H	7	1,5	0,3

Team	*Average Backlog Items*	*Standard Deviation*	*Trend*
Average	*11*	5,2	-

Table 13. The teams' Backlog Item data.

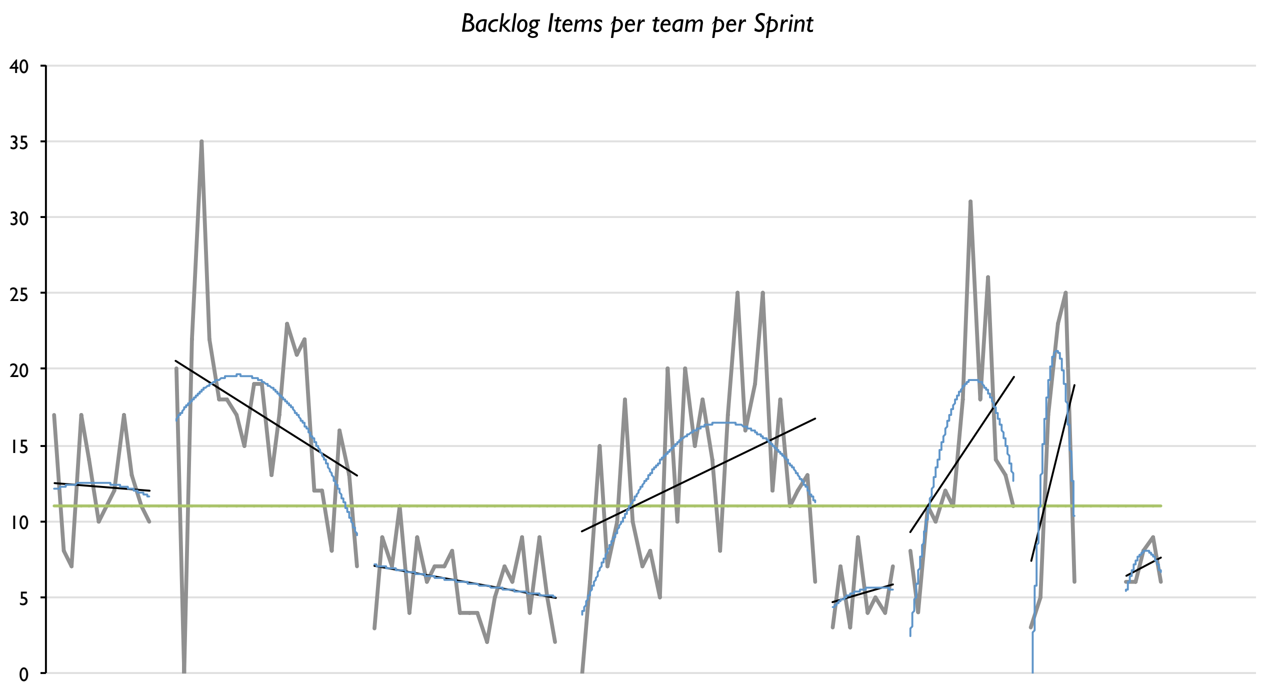


Figure 25. The amount of Backlog Items per team per Sprint (in grey), the linear trend and the polynomial trend.

This Backlog Item metric leads to a number of questions: Why is there so much fluctuation in between Sprints? Why do some teams show a negative trend, where a positive one is expected? How come the polynomial trend-line shows a parabola shape, with a decrease in the amount of Backlog Items in the later Sprints?

Since Backlog items have a relative sizing (in the case tool it is measured in Story Points), that makes measuring a team's increase or decrease in efficiency by measuring the total amount of Backlog items committed to per sprint an inaccurate method. This relative sizing also partly explains the somewhat arbitrary course of the graphs.

Inquiry at the users of the case tool learned that the teams sometimes move unfinished User Stories from one Sprint to another. This type of manipulation of the Backlog isn't recorded by the case tool, making the resulting graph inaccurate. If a team commits to five Backlog Items in Sprint A and then moves two unfinished Backlog items to Sprint B, the case tool will show that the team has only worked on 3 Backlog Items in Sprint A, which is of course not the case. These large dips in the graph thus can have a number of different root causes, from expected ones such as illness, to having the team work on a small amount of large User Stories as compared to a large number of small User Stories, to the teams moving unfinished User Stories to another Sprint.

To summarize:

- The fluctuation in the amount of Backlog Items per Sprint is partially caused by the relative sizing of Backlog Items.
- The fluctuation in the amount of Backlog Items per Sprint is partially caused by teams moving items between Sprints.

- The negative (linear and polynomial) trends are partially caused by teams working on fewer, larger items.

In order to make the above graph more accurate the uncertainty derived from the size of the Backlog Items needs to be removed. Instead of looking solely at the amount of Backlog Items picked up in a Sprint, Figure 26 below shows the total amount of Story Points a team commits to on a per-sprint basis. In this graph every team shows a positive trend. The obvious downside to comparing sprints on a Story Point basis is that estimating with relative sizes inadvertently leads to 'estimation creep', where teams give a Backlog Item a higher estimation to simulate higher productivity. This is static that should also be removed.

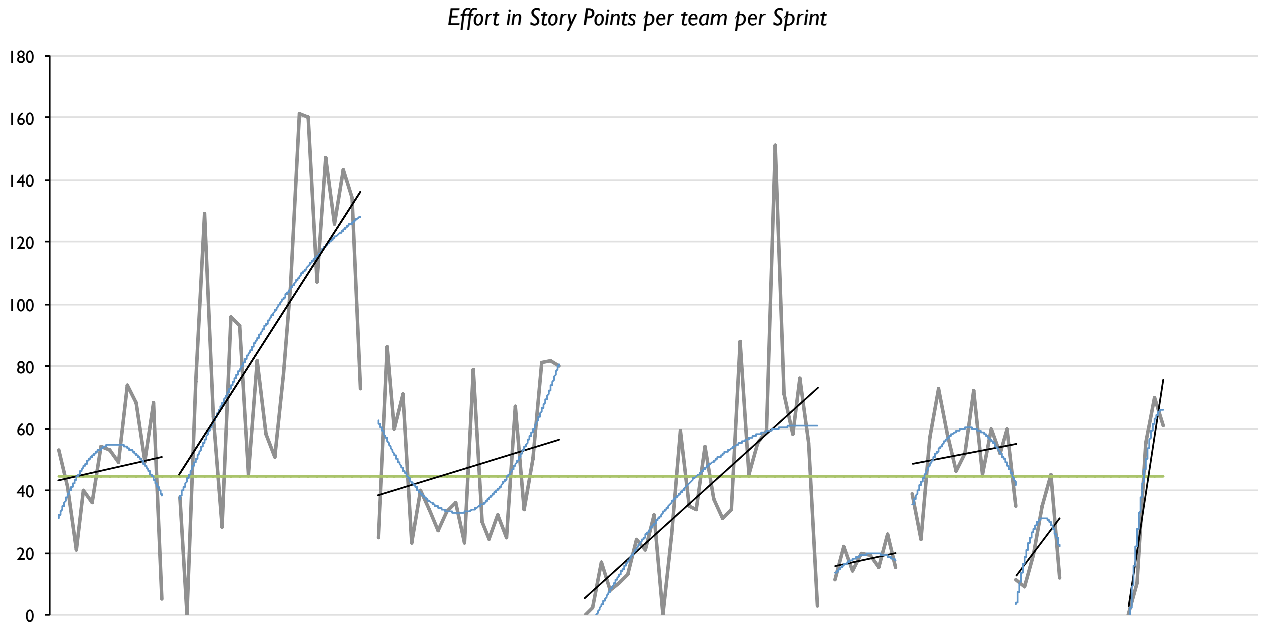


Figure 26. The estimated effort in Story Points per Sprint.

Team	*Average total Effort per Sprint*	*Standard Deviation*	*Trend*
Team A	47	19,1	0,63
Team B	90,6	44,7	4,34
Team C	45,3	24,8	0,85
Team D	39,2	32,7	2,51
Team E	17,8	4,9	0,6
Team F	51,8	14	0,53
Team G	23,8	15,7	3,67
Team H	39,2	34	18,2
Average	*44,3*	*24,3*	-

Table 14. The teams' Backlog Item data.

The graph still shows the same fluctuation in the lines as was observed in the previous paragraphs. Of note are the cases where the amount of Story Points has a value of 0. In retrospect these dips appear to be caused by teams moving several large unfinished Backlog Items to another Sprint leaving the Sprint without any Backlog Items.

As mentioned earlier in the introduction into Scrum, Backlog Items are broken down into Tasks, which means that Tasks are more concrete and specific descriptions of work. In TOOL Tasks are estimated in the amount of hours required for the task to be completed. In order to create a more accurate picture of the teams' progress Figure 27, which can be seen below, was created. This graph shows the amount of committed Task hours per team per Sprint.

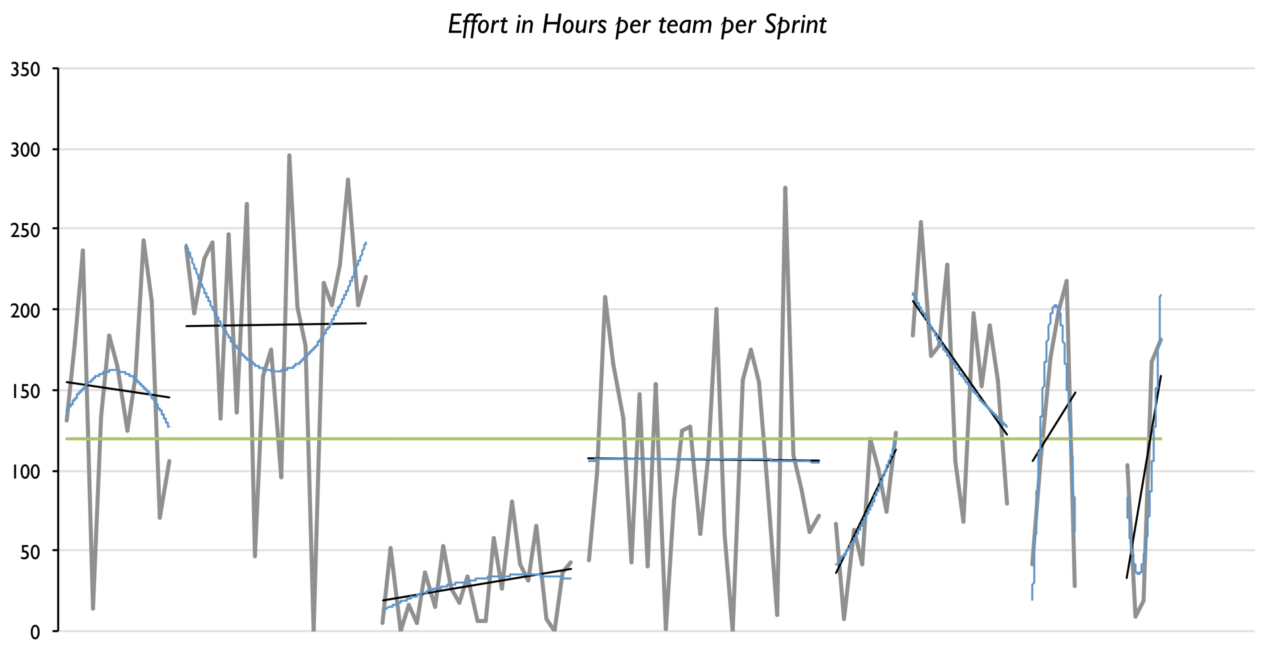


Figure 27. The estimated effort in hours per Sprint.

Team	*Average total Effort per Sprint*	*Standard Deviation*	*Trend*
Team A	150	64,4	-0,79
Team B	199,5	60,9	0,08
Team C	31,6	21,8	0,89
Team D	110,6	64	-0,05
Team E	74,5	39,6	10,95
Team F	163,6	56,1	-7,54
Team G	127	81	8,51
Team H	95,8	74,7	31,4
Average	*119,1*	*57,8*	-

Table 15. The teams' Task data.

The trends show a stark contrast to the trends in the previous graph, teams that appeared to be improving their productivity with Story Points now seem to a stable or even decreasing productivity. This raises the question whether teams were artificially raising their productivity through 'estimation creep', as mentioned earlier.

Figure 27 and Table 15 raise further questions: some teams appear to have very little or no Tasks at all in their Sprint Backlogs, what's causing this? Secondly, the amount of hours the teams estimate varies greatly in between Sprints, even more so than the amount of Backlog Items and the amount of Story Points. This is peculiar because there is a set number of man-hours in a week. The team members all work full-time, so in perfect conditions (no changes in team composition for example) the estimated amount of hours per Sprint should always be the same. So why isn't this the case in the graph? Another pattern of interest is the fact that there are Sprints with no tasks, while those Sprints did have Backlog Items, how is this possible? Team C also appears to be an outlier as the amount of hours per Sprint lies far below that of the other teams.

Before we answer these questions the data can be made more accurate still using the 'team size' metric established earlier, by dividing the amount of hours committed to in a sprint by the amount of team members. Two interesting patterns emerge (see Figure 28), the first is that the amount of hours a team commits to within a Sprint for most teams comes nowhere close to what we would expect from a two-week iteration (which would normally be around 80 hours). The second is that one team appears to commit to far more work than the other teams.

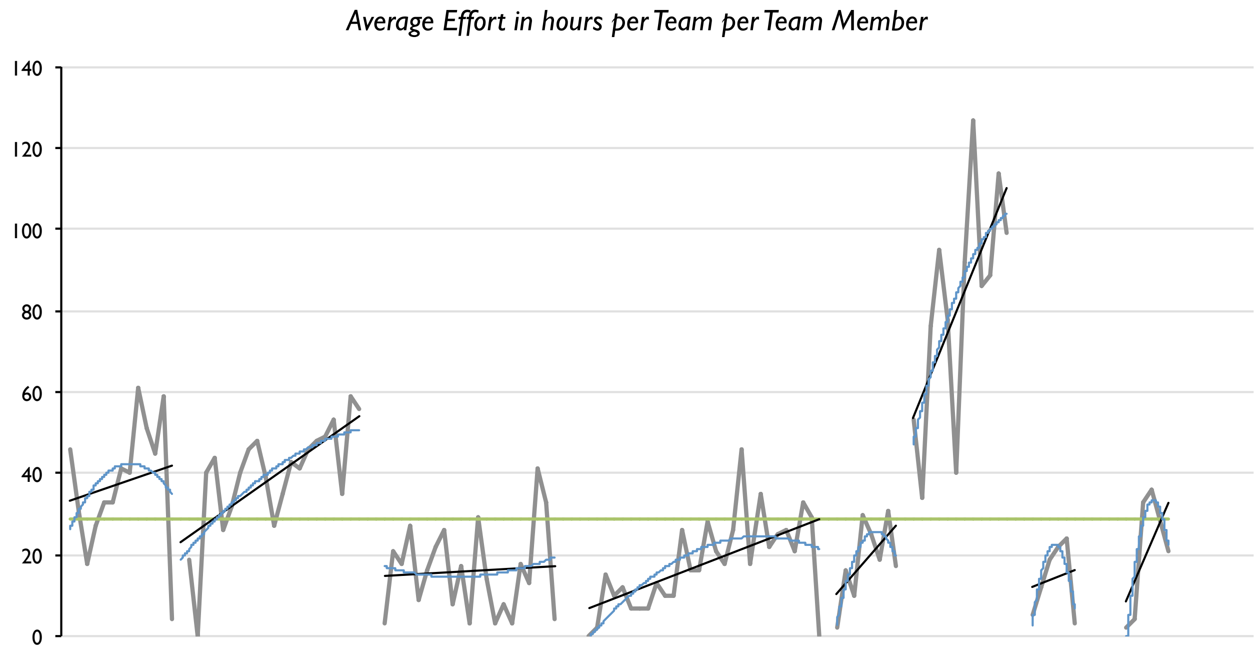


Figure 28. The estimated effort in hours per User per Sprint.

Team	*Average total Effort per Sprint per Developer*	*Standard Deviation*	*Trend*
Team A	37,6	15,9	0,72
Team B	39,8	12,2	1,48
Team C	16	10,9	0,12
Team D	18,5	10,7	0,81
Team E	18,8	9,9	2,40

Team	*Average total Effort per Sprint per Developer*	*Standard Deviation*	*Trend*
Team F	81,9	28	5,14
Team G	14,2	8,9	0,83
Team H	19,2	18,2	4,85
Average	*28,8*	*14,3*	-

Table 16. The teams' Task data.

Table 8 makes this data more transparent. The average work-week of one developer should be around 80 hours per week, only Team F appears to match this amount of hours. However, the graph and standard deviation shows that even for this team the amount fluctuates significantly.

Further metrics which could be used to remove even more static from the data could not be derived from the database, thus the data was presented to the users of the case tool to help explain the patterns we were seeing. Of course, the most important questions were: Why is the average effort in hours per week much lower than the average work week? Why is only one team performing close to the expected results?

After consulting with a number of case tool users from different teams we were able the root causes of these patterns were narrowed down. The team that consistently commits to more hours than the other teams appeared to be the case tool development team. Since this team is actually responsible for the development of the case tool they appeared to be using it with far more discipline than the other teams. Their estimation of the amount of hours they worked appeared to be relatively accurate. Obviously the other team members did work around 40 hours per week, and so this means that they did not log all the Tasks and all the hours they worked in the case tool. The tool also does not show how much work per team member is logged for a particular Sprint, implementing this kind of visual cues might increase the accuracy of the Sprint Planning of teams.

To summarize:

- The team that appeared to have the most accurate planning was the case tool development team.
- Being more involved with TOOL, this team used the tool with a higher level of discipline.
- The other teams appear to use the tool with lower levels of discipline, they do not log all of their work in the tool.
- The tool lacks features that help the teams with their Sprint Planning.

This usage pattern is the most interesting behavior that was extractable from the database of the case tool, as it suggests that different teams are using the case tool with varying levels of discipline. If this is the case, and that is what is being hypothesized, that would suggest that not every user of the tool seems to agree on the added value of using TOOL. This hints at a lack of either features, or other characteristics of the tool that contribute negatively to the way it is perceived by its users.

Further data is analyzed to find proof that confirms or denies the hypothesis that different teams use the case tool with varying levels of discipline. Once these metrics have been found questions can be formulated that will guide a discussion with the case tool users to look for the root causes of this behavior.

Description length
As mentioned earlier, every Backlog Item in the case tool has a dedicated field which is used to describe the Backlog Item; the title, the description, the estimated amount of effort, et cetera. We hypothesize that teams that use the case tool with lower dedication tend to not fill in the Description field or only write down a very minimal description.

In Figure 29 below the length of the descriptions of the Backlog Items are accumulated and divided by the amount of Backlog Items per Team per Sprint. This gives the average description length per Sprint for all teams (in number of characters). The results in between teams again varies greatly. Considering that the description length is in characters and not in words the average description is also very short (133 characters on average). The graph raises a number of questions: Why does Team C, which has a relatively consistent description length, score significantly lower than the other teams? Why is the difference between Sprints within a single team so large?

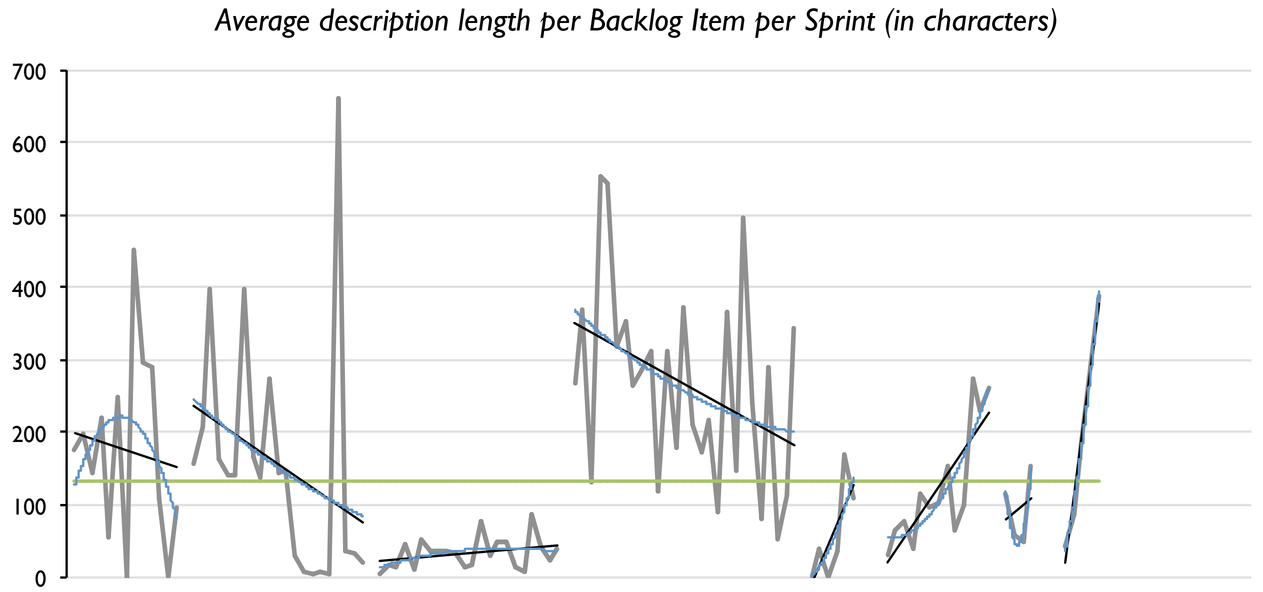


Figure 29. The average description length per Backlog Item per Sprint (in number of characters).

Team	*Average Description length per Backlog Item per Sprint*	*Standard Deviation*	*Linear Coefficient*
Team A	176,6	128,6	-3,90
Team B	156,5	164,8	-8,05
Team C	33,6	21,7	1,02
Team D	266,9	135,3	-6,49
Team E	44,1	62,9	27,51
Team F	124	81,6	17,22
Team G	63,2	61,9	9,7
Team H	199	142,3	89,3
Average	*133*	*99,9*	-

Table 17. The teams' Description data.

Taking a closer look at the individual descriptions, many of the large spikes in the graph can be attested to the fact that some users use the description text field to dump large pieces of code rather than describing what the Backlog Items does and what its acceptance criteria are. This means that the actual descriptions were even shorter than graph in Figure 29 would suggest.

So if the description field is not used extensively to describe work items, how do the teams make sure that what they are doing is according to the requirements set by the Product Owner? As it turns out and as expected, the descriptions are mostly present implicitly. Put simply, the team members know what to do. The team members discuss their Backlog Items with the Product Owner during the Sprint Planning, they agree to what the scope of these items should be and go from there. Further clarification and definition happens through channels outside of the case tool, for example through e-mail, Skype calls, or simply face-to-face conversations.

To summarize:

- The Description length is relatively short for all teams.
- Backlog Items aren't described properly due to the fact that the team members implicitly know what to do for a particular item.

Amount of Backlog Items finished in time
A final metric that could be extracted from the database and says something about the discipline of the usage of the tool is the amount of Backlog Items that weren't finished on time. Originally this metric was going to be used to judge whether or not longer descriptions and more descriptive titles led to a higher success rate in the Backlog Items and Tasks. In Scrum the Product Owner moves the state of a Backlog Item to done when that item has been finished as per the Acceptance Criteria and the Definition of Done.

Upon closer inspection of the Backlog Item data, large quantities of items appeared to be marked as 'done' past the end date of the Sprint. The grey lines in this graph show the total amount of Backlog Items for every Sprint per team, the red line marks the amount of Backlog Items from that Sprint that were marked as 'done' after the Sprint deadline had passed.

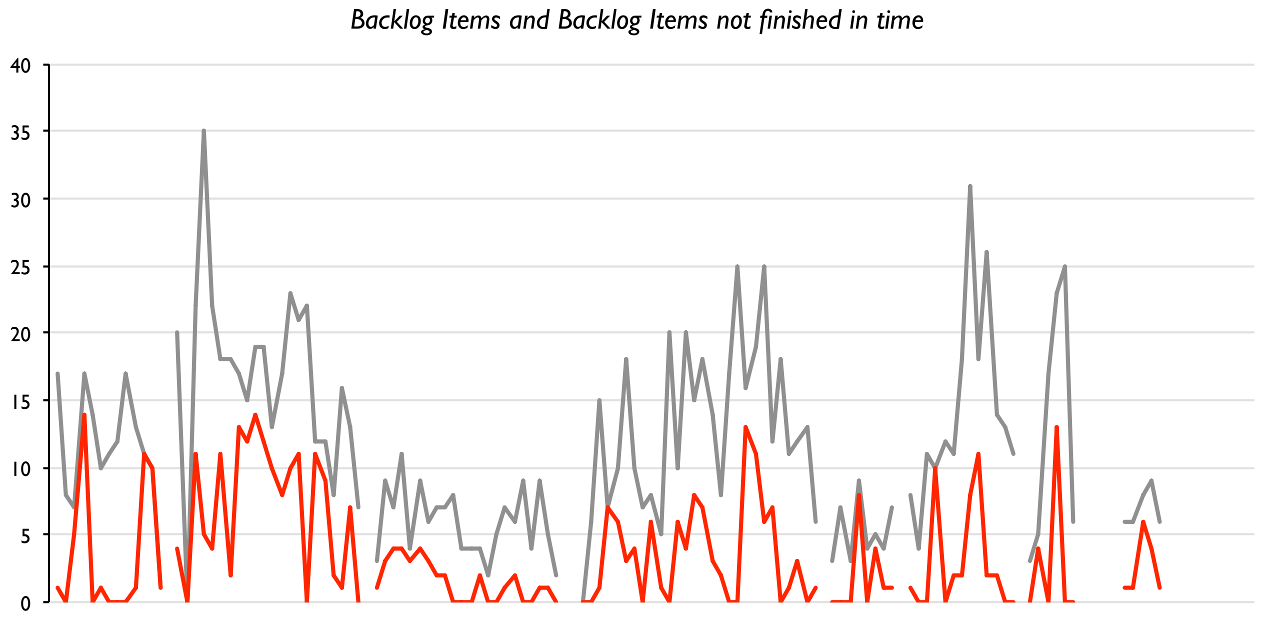


Figure 30. A comparison of the amount of finished (in grey) and unfinished (in red) Backlog Items.

Looking at the graph this would mean that a significant majority of all the Sprints performed by the teams working for the case company would be unsuccessful. To verify this the official Sprint results for all development teams working at the case company were analyzed. These documents are used during the communication with the clients of the case company to discuss how well the team and project are performing. These documents are validated by the Product Owners who work at the client companies.These reports showed that almost every Sprint by all the teams were considered to be successful. See Table 18 below for an excerpt from such a Sprint results report for Team A. The data clearly does not match with the graph for Team A that can be seen in Figure 30.

Estimated Points	*Delivered Points*	*Reliability*
40	40	100%
36	36	100%
41	41	100%
31	31	100%
43	43	100%
41	41	100%
41	41	100%
53	53	100%
41	41	100%
21	21	100%

Table 18.The estimated and delivered Story Points of Team B.

Apparently the teams are successful, but the data in the database does not reflect this. These findings were discussed with a number of team members and the Product Owners. As hypothesized it turns out that the Product Owners aren't very strict with keeping the tool in sync with the work of the development teams. Backlog Items are marked as done not at the end of a Sprint, or as soon as they are finished, but at a later point in time during the next Sprint. This distorts the data in the database to a critical degree. The same lack of discipline applies to the Tasks; not all Tasks which are actually worked on are created by the team members, and their status is not kept in sync with the actual progress of the Sprint.

Again this shows that the discipline of the usage of the tool is not at the level the case company might want it to be. According to the users the reason for this is that due to the small size of the teams everybody knows what the status of the Sprint is. Communication lines are short and direct and the Daily Standup contributes enormously to everyone always staying up to date on the status of the project.

To summarize:

- Not all work is converted to Backlog Items and/or Tasks.
- The state of Tasks are not kept in sync between the case tool and the actual progress of the Sprint.
- Product Owners fail to update the state of Backlog Items when they are done.
- Due to these issues, the data in the database cannot be used for a more intricate analysis.
- These patterns do show that perceived usefulness of the tool is low.

5.2. Observations

Over the course of a year the researcher has been present at the case company as a researcher in the field, taking up the role of a Proxy Product Owner within a distributed Scrum team. The goal of this element of the study was to supplement the findings of the database analysis, i.e. to find out through observation of the Scrum teams why the tool was so poorly used.

As mentioned earlier a Proxy Product Owner (PPO) takes care of the day-to-day operations of a Scrum team. The management at the case company defined the Releases and Sprints, the PPO's function is to ensure that these general directions are worked out. The advantage of taking up this embedded position is that the researcher had unhindered access to the tool, from the back-end infrastructure, to roadmaps containing future changes to the GUI. As such it became possible to get to know every aspect of TOOL. Next to that a large amount of time was spent talking to users of the case tool and prospects interested in purchasing a license for the tool. Due to these discussions it became clear what expectations Scrum professionals have of such a digitized tool and where the case tool failed or succeeded to meet these expectations. The experience and knowledge gained during this period of time has already been partially interwoven in the previous chapters.

The observations have been grouped into three related sets of causes: the characteristics of the case tool itself, the unique characteristics of the case company and the characteristics of the Scrum framework itself.

5.2.1. The case tool

Lack of features
There are some very common Scrum concepts which aren't supported by the tool. During the observations of the Scrum teams and conversations with the users this issue surfaced frequently. To give an example of such a shortcoming, support for one of the most important pillars of Scrum's inspect-and-adopt philosophy, the Retrospective Meeting, was completely absent. This led to teams writing down the proceedings of this meeting in a document and then storing that somewhere else. Another missing concept is the Taskboard or Scrumboard. The Scrumboard is one of the artifacts which would be ideal to digitize in a distributed setting, as it is hard to keep multiple Taskboards in sync when team members are working on items from different locations. This functionality too is absent from the tool.

These are pure Scrum concepts, but the tool also misses some additional features one would expect from a tool that is aimed at supporting distributed Scrum teams. One of the most frequently mentioned missing features is the ability to share data between teams. This feature did not show up in the database analysis due to the fact that all of the teams who currently work with the teams are from separate client organization and thus all work on separate projects. On a smaller scale it is also impossible to create relationships between Backlog Items; Bugs cannot be related to specific Backlog Items, dependencies cannot be created.

The consequence of lacking features, bar the fact that prospects will not invest in the tool, is that the teams of the client company, who are to some degree forced to use the tool, have to resort to other means to fill these feature-gaps. This is of course inefficient, which leads to frustration and the aforementioned lack of usage discipline.

Unfinished features
Some features of the tool are there to solve problems that are specific to the problems distributed Scrum teams face. For instance, the Standup app allows users to record their Standup so that it can be replayed at a later point in time. For distributed teams, where timezone differences sometimes make it hard to be able to attend certain meetings such as the Daily Standup, the ability to record these meetings can contribute greatly to keeping everyone up-to-date on the status of the team. However, in the implementation of this feature it is only possible to record one camera-feed. Since in a distributed team the team is always split among multiple locations, that would mean that these Standups do not contain all information given during a Standup.

Another example is the Video app, which allows users to set up live IP camera streams, so distributed teams can see who is sitting where, which colleagues are present, it allows for communication that increases the bonding between team members. The maximum amount of supported video streams is limited to two. These are not the only unfinished features, Impediments and Epics cannot be logged in the Backlog app, the Release Burndown cannot be set to User Points, only to Task hours, the Burndown can only be set for Sprints, not Releases et cetera. The features that are there do not seem to be fully finished, and the users agree with this notion.

Mismatch of focus and set expectations
The goal of the case tool is 'to help (distributed) Scrum teams deliver better software and improve themselves, by supporting all aspects of Scrum, with a special focus on human interactions'. However, what

was observed is that the tool is mostly being used by the Product Owners and other managers of the Scrum teams to check whether the teams' Key Performance Indicators meet the expectations. For the developers and other members of the Scrum team the tool appears to have little added value, the tool is mostly cumbersome to use and it does not tie in properly with existing tools, bar Microsoft Visual Studio.

Performance
Although the performance of the tool is not an element which was planned to be taken into account for this research, the performance of the tool is contributing negatively to its overall usage. Over the course of the study many performance related issues emerged. The browser-based nature of the case tool, as opposed to being a native application, appears to result in compatibility issues with different versions of web browsers. Numerous cases of features not working, or work being lost were reported. Besides these types of issues the browser based nature of the tool also also leads to long loading and processing times, for instance when creating work items and updating their status.

5.2.2. Contextual factors

Mismatch between case organization and tool
The lack of proper usage of TOOL within the case company can partly be attested to the fact that there isn't enough visibility for the tool. The internal processes within the case company make it almost unnecessary to properly use it. The performance data of the teams are gathered manually for example, and are not taken from the KPI app of the case tool itself. Some teams don't even use the KPI's identified in the case tool because they don't agree with the set that is available and the way they are calculated. If the tool was implemented properly into the workflow of the teams, the teams would have to start using it with more discipline, which would lead to more user feedback and eventually a better tool.

Team set-up
Another reason for the lack of usage, or lack of added value of the case tool to the daily processes of the Scrum teams, is the unique composition of the teams working for the case company. All of these teams are located in only two locations, location A being the Netherlands, and location B being India. The timezone difference between these two locations is relatively small (between 3,5 and 2,5 hours). This means that most communication can take place through synchronous communication, and there is relatively little need to extensively store data in the case tool. Therefore it would be very interesting to see whether teams that have a different geographic distribution would show the same usage patterns.

5.2.3. The Scrum Framework

Small teams diminish the need for extensive documentation
As mentioned earlier, the tight-knitted team set-up of the Scrum framework does not encourage, or seem to cause a need for extensive documentation of work items and elaborate tool support. All information that is needed to finish a Sprint is present with all the members of the team and when it isn't a team member is never far away. Due to this the need for a synchronous communication tool seems to be far greater than the need for a tool that is has a convoluted and asynchronous nature such as TOOL.

Transparency is present without tool support
Due to the fact that Scrum is based on short iterations and thanks to the many processes, meetings and artifacts incorporated in the Scrum, the framework is already leads to transparent processes and therefore one must question whether or not the extra effort that is required to create a little more transparency through a tool is worth it. The developers in general seem to disagree with this need, although people who from a management perspective do seem to agree with this need. Especially when a single person needs to manage multiple teams. Unfortunately such a team was not present in the case tool.

5.3. Conclusions

This section will serve to answer the sub-question: *What are the findings from the analysis of Scrum project data and how are these supported by observations of tool usage?* The analysis of the case tool was started with the underlying thought that a large quantity of data related to missing features could be identified, for example: if it turned out that longer descriptions lead to relatively more successful Backlog Items, the case tool could assist users in writing longer descriptions. If shorter Backlog Items, with smaller and more concrete Tasks would lead to more successful Sprints, the application could take a more agent-based set-up, encouraging the users to create consistent and easily manageable work items. Data about the team size would be able to tell us which team size turned out to be the most successful for Scrum teams. Patterns in Backlog Item descriptions could have hinted at additionally needed fields and properties for the concepts identified in the tool. The list goes on. During our MQG analysis however, it became clear quickly that the data in the database was not at all suitable to draw these kind of conclusions. The more explorative MQG approach, investigating the metrics first and seeing what kind of conclusions can be draw from them, in stead of going in with a set approach and trying to get the data to confirm our deny that approach, turned out to very useful in this case.

We might not be able to draw any conclusions about which kinds of Scrum team and work item configurations would be more successful, but the data did tell us something more important about the case tool on a much higher level; the apparent lack of proper usage of the tool. From a birds-eye view it might look like all the teams work properly with the tool, but the analysis of the finer details of the data revealed that this does not appear to be the case. Descriptions of User Stories are poorly used, the discipline with respect to keeping the status of Tasks up-to-date with the actual progress of the Sprint is lacking, due to this the metrics in the case tool are in accurate, all elements analyzed shows signs of lack of proper usage.

The reason for this behavior appears to be twofold: On the one hand the teams that are currently using the case tool are so small, that explicitly saving all aspects of their Sprints in a digital tool is time consuming and unnecessary. All knowledge with respect to the the current status of the case tool exists implicitly between the team members. Any uncertainties are resolved on the spot with face-to-face communication, or through other tools, such as Skype. The meetings, demo's, and other Scrum concepts ensure that everyone knows what to do and what the others are doing. Unlike large software development teams, not uncommon in waterfall based approaches, the communication lines in these Scrum teams are short, making the extensive documentation of all proceedings unnecessary. An interesting note here is that all teams using the case tool are fully independent, there is no need for these teams to work together. As such it would be interesting to find out whether multiple teams that work on the same project would show different data, as more information would need to be shared, which would possibly result in the need for more descriptive work items. Unfortunately this could not be analyzed with the current data-set.

The second major reason which can be deduced from the database analysis is that not all concepts and features of Scrum are available in the case tool, for instance Taskboard functionality or the ability to define Impediments. This means that the users of the case tool will have to perform some actions and store and manipulate documents outside of the case tool, while other actions have to be managed within the tool. This obviously leads to inefficiency and does not encourage usage of the tool. Some features that are present appear to be incomplete.

What distributed teams appear to need, and what the case tool partially fails to provide, is the means to solve the specific problems distributed teams face. A tool aimed at distributed teams should not focus to provide support for every Scrum process, rather it should aim to fill the gaps that are created when a team is being separated to multiple locations. That means, solving communication issues, ensuring that data can be saved in a persistent way that is accessible to all involved with the team, making sure that shared artifacts such as a Taskboard are easily synced. The case tool does this to some degree, for instance through its Standup app, however, this functionality is largely overshadowed by dedicated tools, such as Skype or Dropbox.

5.4. Development Focus Areas

Based on the Literature Review and the findings of this chapter, the following roadmap for the tool was created to serve as a suggestion for which elements of the case tool should be focussed on first, according to the researcher, in order to increase the usage of the tool. In no particular order:

- *Support Scrum processes which are especially useful to distributed teams.* The Sprint Review and Sprint Planning sessions are mostly coordinated through synchronous communication channels such as Skype. Therefore the need for added functionality to support these processes is comparatively low. The Retrospective Meeting is a good candidate for tool support, being able to store project related documents through the tool will significantly add to the perceived value of the tool.

- *Support Scrum artifacts which are especially useful to distributed teams*. Not all missing Scrum concepts should be implemented with the same level of priority. Some missing artifacts however would be especially beneficial to members of distributed teams. Most notable is the Taskboard, which is a physical artifact that is hard to keep in sync across multiple locations. Other candidates for quick inclusion into the tool are team specific documents, such as storing the DoD and DoR, and the ability to create Epics and Impediments.
- *Complete unfinished features.*
- *Address performance related issues.*
- *Enable cross-team functionalities.*
- *Integrate working with the tool into the daily workflow of the teams.*
- *Increase amount of metrics (such as a Release Burnup).*
- *Increase the connectivity with third party software suites to avoid duplication of effort for the Scrum teams.*

6. Implementing Taskboard and Retrospective functionalities

The following chapter describes the process of designing, implementing and evaluating some missing features into the case tool, it serves to answer the fourth sub-question: *Which new features and functionalities can be implemented within the course of the study and how are these features evaluated by the users?* Based on the comparison of the case tool with Scrum literature and from observations of seeing the teams work with the case tool, a number of missing features of the tool were identified. The main objective of implementing the features mentioned in this chapter is to increase the perceived value of the tool for its users.

We focussed on implementing two new features which were both frequently mentioned in the previous chapters, a Taskboard feature (a Scrum artifact) and a feature that allows the teams to record their Retrospectives (a Scrum process). Especially the lack of a Taskboard feature was considered a major issue for the distributed teams, as keeping multiple Taskboards in sync proved to be troublesome. These were two omissions in the case tool which led to activities having to take place outside of the case tool environment, which in turn significantly lowered the added value of the case tool. By implementing these features we aimed to increase the time the users spend inside the tool, decreasing the time users have to spend doing Scrum related activities through other tools and environments.

6.1. The Taskboard app

The Taskboard, or Scrumboard, is a surface, usually a whiteboard or an empty wall, that shows the Backlog Items and the Tasks of the current Sprint in such a way that one can immediately get an overview of the status of all items. Displayed below in Figure 31 is an example of a Taskboard, on the left hand side the User Stories are displayed, in the column after that the 'To Do' Tasks related to that particular Backlog Item are displayed, followed by the Tasks that are 'In Progress', and finally those that are 'Done'. Once a team member starts working on a Task he or she moves the Task from 'To Do' to 'In Progress' and so on.

As such the Taskboard is a very useful tool to get a quick overview of the status of all Backlog Items and related Tasks. Many organizations have variations on the model displayed in the figure below depending on their specific working processes, for the implementation of the Taskboard in the case tool it was decided to stick to a basic Scrumboard version with a 'Backlog Item', 'To Do', 'In Progress' and 'Done' status.

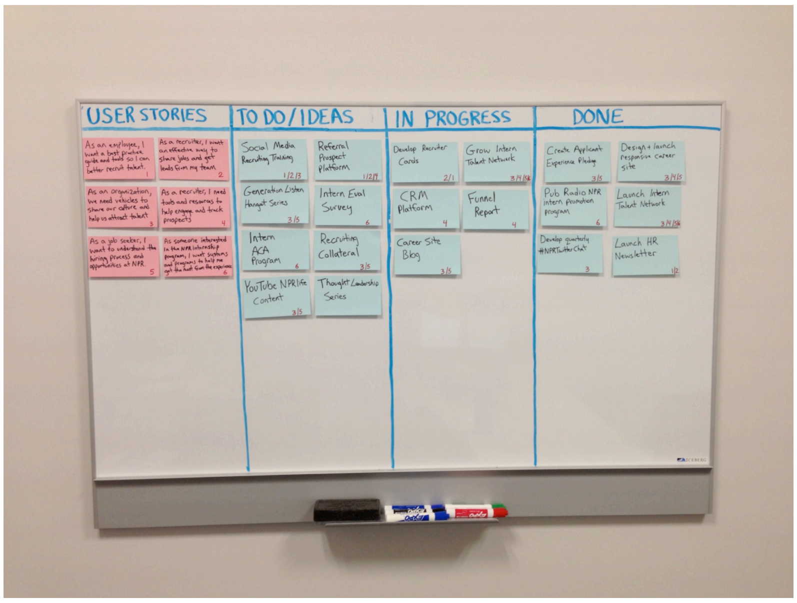


Figure 31. A physical Taskboard

As mentioned in our study of Scrum literature, interactivity and metaphors were deemed important for Scrum tools (Azizyan et al., 2011; Behrens, 2006; Petersen & Wiil, 2008; Wang & Maurer, 2008). In order to create a feature which would feel instantly familiar to the users of the case tool, the design of the Scrumboard functionality closely mimics that of a physical Scrumboard. Backlog Items and Tasks are displayed as post-its on a virtual board. Users can easily drag and drop these Tasks with their mouse cursors to change the status of the items. The updated status is automatically reflected in the Backlog functionality of the case tool, and of course the Taskboard stays in sync across all accounts, overcoming the need to have multiple Scrumboards.

We took advantage of the possibilities of having a digitized Taskboard by showing, among others, the total amount of hours of work remaining, the amount of open Tasks, a picture and the name of the team member who is currently working on the Task. Clicking on the description of the User Story will show a pop-up window with the full title and the description of the Backlog Item. The remaining hours for the tasks can easily be adjusted by double clicking on the 'Remaining field' on the post-its. The functionalities in the digital Taskboard therefore cover all those a physical counterpart would offer. After implementing this new feature of the case tool, its successfulness will be evaluated in the following chapter.

[Confidential data removed.]

6.2. The Retrospective app

The Retrospective Meeting is a very important process in the Scrum framework. The inspect-and-adapt foundation of Scrum emphasizes the constant need for reflection and self-improvement. It is exactly this what the Retrospective Meeting attempts to facilitate. During the course of the study the researcher has taken part in many of these meetings. The process of the meeting is simple, the Scrum team as whole gets together at the end of the Sprint to discuss how the Sprint progressed. Specifically, three questions are asked:

- What went well during this Sprint?
- What didn't go well this Sprint?
- What should we do better next time to ensure the Sprint goes better?

All team members weigh in their opinion as much as possible. The idea is to start a discussion with the goal of identifying areas, be it in the project, or the Scrum process itself, that needs improvement.

Before implementing the Retrospective app into the tool, the meetings usually took place through tools such as Skype video-chat, this allows for synchronous communication where all participants are able to see each other, which makes it easier to react to the discussion. Both the team members in the Netherlands, as well as the team members in India gathered behind a single PC with webcam, so that everyone was visible. Notes of the Retrospective sessions were saved in a separate document, or not saved at all.

The process of the meeting makes it very easy to translate it to a digital format. We decided not to focus on trying to replace the current tool the teams use as a primary means of conducting the meeting (mostly Skype), rather the focus was to support the meeting by allowing the users to store their Retrospective proceedings in a meaningful way. The design of the Retrospective app was kept as simple as possible (see figure 33 below); the screen is divided into three columns, one column for each question of the meeting. Every team member can add his items to each of the columns, this can happen during the Retrospective meeting, or it can happen afterwards. Of course, one of the benefits of storing Retrospectives in a digital tool is that the contents can be stored for later inspection. This way the users can easily select a previous Sprint to see whether or not they have improved what they set out to improve.

Users can enter and remove items in real-time, the app auto-refreshes so that contents of the screen are always up to date. After implementing this new feature of the case tool, its successfulness will be evaluated in the following chapter.

[Confidential data removed.]

6.3. Evaluation

The features were evaluated with multiple users over the course of several months. The users were given enough time to get accustomed to the features and how they worked. Usage of the features was not in any way encouraged by the researcher, the addition of the features, however, was of course mentioned to the users of the tool. The database of the case tool was also analyzed to gather some objective data on the usage of the new features.

The result of this evaluation is that, yes, the perceived usefulness increased of TOOL increased. All teams currently use the Taskboard as a primary means of updating the status of their Sprint. Being able to move Tasks around of a virtual board that closely mimics how a such a board would work in real life was seen as a positive addition of the tool. Especially considering the fact that previously the teams had to dig down deep into the Backlog app to perform the same task. Some teams noted that they used additional states compared to the ones in the app, such as 'In Testing' or 'Waiting for approval'. In a future update of the tool it might be beneficial to allow users to modify the tool to their own specific processes.

When analyzing the database for usage patterns no concrete improvements of the usage discipline were observed. User Stories were still marked as done after the Sprint was finished, and Tasks were not updated as hours were burned, but rather at the end of the day, or after other arbitrary periods of time. It seems that having to manually update Tasks and Backlog Items still formed a distraction from work. Some teams were a little more strict with the usage of the Taskboard functionality, what was found is that it mostly depended on how involved the Product Owner was with the team. An involved Product Owner knows who is doing what and what the status of the Sprint is, where a Product Owner that is less involved has to depend more on the tool and will see to that it is used properly.

The implementation of the Retrospective app was less successful. When checking the database to see the usage of the Retrospective app it became clear that only four out of eight teams currently using the tool were registering the proceedings of the Retrospective meeting with TOOL. It must be noted that these are partly different teams than the ones used in the database analysis. Inquiry at the teams learned that for most explicitly saving the proceedings of the meeting was deemed as being unnecessary.

To conclude: The addition of the new features is evaluated positively, and adding additional features mentioned in this study will surely help to create a better, more effective Scrum Project Management Tool. The question remains however, whether within the context of the case company there is a need for such a tool. Not all features are used properly, seemingly because the need for those features is not present within certain teams.

7. Conclusion

The goal of this chapter is to answer the main research question: '*How can the perceived usefulness of advanced tool support for distributed Scrum teams be improved by making improvements based on the analysis of performance data and the observation of distributed Scrum teams?'*. In this study we have evaluated the use of a Scrum Project Management tool from a number of perspectives. The tool was compared with Scrum literature, so the feature-set of the tool could be compared to the concepts described in literature.The tool was compared with scientific research, so we could compare the contributions and recommendations of others with experience in digitizing aspects of Scrum with the implementation of the case tool. Usage data was analyzed to give insight into the usage patterns of the users. Observations of the researcher, acting as a Proxy Product Owner in a distributed Scrum team, were used to interpret the usage patterns. Finally, two major new features were implemented with the goal to increase the usage of the tool.

The most important lesson to take away from this study is that, at least within the setting of the case company, implementing a Scrum Project Management Tool that tries to support every aspect of the Scrum framework, and needs to be used intensively on a daily basis, feels like an overkill. The needs and requirements for such a tool should come naturally from the users, however, the case tool does not seem to be built around this perception, it based its functionality on the concepts from the scrum literature rather than the needs of distributed teams.There are some very real and tangible negative consequences of distributing teams over various geographical locations, consequences which are well worth mediating. Communication becomes more complex, it is harder to keep everyone in sync on the status of all items, storing and manipulating data specific to the project becomes more important, cultural issues arise, et cetera.The case tool appears to focus on these issues only partially, with the result that to solve these kind of issues the users have to resort to other tools.

Communication is far richer and more efficient when it is done either face-to-face, or through a direct video communication tool, such as Skype.That means that the case tool isn't fit for a communication tool. There is an Interactions Feed where people can post short messages, but nobody uses that when the majority of communication takes place through other, more efficient tools. Other features where the application could have some benefit and tries to solve distributed team specific issues, such as being able to record a Standup are hindered by the fact that these features feel unfinished. In the Standup app for example users are only able to record their standup one-way, that means that a part of the Scrum team isn't record, largely devaluating the use of such a feature.

The nature of Scrum also does not lend itself easily for a conversion to a digital platform, simply because in most cases it feels unnecessary. Most information pertaining to the work that needs to be done in a Sprint is implicitly available among team members, where information is lacking and further comments are needed the users rely on their direct communication tools mentioned earlier.

The developers are the people who are mostly using TOOL, at the time of writing it is however used more as a support tool for management to get an insight into teams' key performance indicators, rather than as a tool that supports the entire Scrum team. Some of the teams encountered during the study used other tools for their work item repository, which are more broad in scope and features and tie in directly to the software suites developers use for programming. This is far more efficient approach than using the case tool, developers can stick to a single tool for programming.The major problem is that tools like TOOL only appear to be useful for management, as there is one thing Scrum lacks, and that is outputting tangible KPI data.

Looking at the larger picture, the case tool takes a middle way that does not seem to be feasible. Either a tool is created that encompasses all the features which currently cause its users to look for other applications (from development to communication), or it focusses on a smaller number of features that make sense for distributed teams that can be directly tied in to the tools that are already being used.

Summarizing, to answer the main research question, in the current setting of the case company, where single teams work on separate projects, there appears to be simply no need for a Scrum Project Management Tool. Communication becomes even more important for a distributed Scrum team, and as such we would recommend communicating through synchronous communication platforms instead of through an asynchronous tool such as TOOL. There is a wide variety of Scrum Project Management Tools available, and the successfulness of some of these prove that they do have merit, but we expect that this merit only becomes visible when these tools are used for more complex projects, with teams spread over more locations, and with multiple teams working on a single project.

7.1. Limitations and Recommendations

There are some obvious limitations in this research, most notable is that the case tool is a very specific tool, used within only one case company, and the generalizability of the results from this study is therefore questionable. There is only a limited number of teams using TOOL, and to complicate things further these teams are all 'in-house', they work for the case company and using the case tool was not a choice. It would have been interesting to see how other teams would have responded to the case tool.

All of the teams using the case tool are distributed between two locations, the Netherlands and India. The relative timezone difference between these two regions is small, therefore a lot of communication could take place between synchronously and the need for a tool is reduced further. Further research effort should be focussed on analyzing how distributed teams use the tool when they are located further apart. We hypothesize that this would increase the need for a tool that allows the team members to store information asynchronously.

Then there is the very nature of the case tool, TOOL. Can we generalize this tool to other available Scrum Project Management tools? Every tool that was encountered during this study appears to have its strengths and weaknesses, and its own approach to providing support to Scrum teams. So the answer to this question is most likely negative. It would be interesting to compare different Scrum and other Agile Project Management Tools with each other to map exactly which features of which tool overlap, and to evaluate with the users of those tools which features are valued mostly by Scrum team members.

Literature

Azizyan, G., Magarian, M. K., & Kajko-Matsson, M. (2011). Survey of Agile Tool Usage and Needs. Agile Conference (AGILE), 2011, 29–38. IEEE.

Basili, V. R., & Weiss, D. M. S. E. I. T. O. (n.d.). A Methodology for Collecting Valid Software Engineering Data. *Software Engineering, IEEE Transactions on*, (6).

Baskerville, R. L. (1999). Investigating information systems with action research. *Communications of the Association for Information Systems*, *2*(19).

Baskerville, R. L., & Wood-Harper, A. T. (1996). A critical perspective on action research as a method for information systems research. *Journal of Information Technology*, *11*(3), 235–246. Taylor & Francis.

Behrens, P. (2006, December 4). Agile Project Management (APM) tooling survey results. Retrieved April 4, 2013,

Blum, F. H. (1955). Action research--A scientific approach *Philosophy of Science*, *22*(1), 1–7. JSTOR.

Brinkkemper, S., & Pachidi, S. (2010). Functional architecture modeling for the software product industry. *Software Architecture*, 198–213. Springer Berlin Heidelberg: Springer.

Caldiera, V. R. B. G., & Rombach, H. D. (1994). The goal question metric approach. *Encyclopedia of software engineering*, *2*(1994), 528–532.

Engum, E. A., Racheva, Z., & Daneva, M. (2009). Sprint planning with a digital aid tool: lessons learnt. Software Engineering and Advanced Applications, 2009. SEAA'09. 35th Euromicro Conference on, 259–262. IEEE.

Gartner. (2012, May 21). Gartner Says Worldwide IT Outsourcing Market Grew 7.8 Percent in 2011. *gartnercom*. Retrieved August 29, 2012, from http://www.gartner.com/it/page.jsp?id=2021215

Hart, H., Boeije, H. R., & Hox, J. (2005). Onderzoeksmethoden. Boom onderwijs.

Herbsleb, J. D., & Mockus, A. (2003). An Empirical Study of Speed and Communication in Globally Distributed Software Development, 1–14.

Hole, S., & Moe, N. B. (2008). A case study of coordination in distributed agile software development. Software process improvement, 189–200. Springer.

Hult, M., & Lennung, S. (1980). Towards a definition of action research: a note and bibliography. *Journal of Management Studies*, (17), 241–250.

KPMG. (2005). Global IT Project Management Survey. How committed are you? KPMG.

KPMG. (2010). KPMG New Zealand Project Management Survey 2010. KPMG.

Kurpicz, M. (2011). Tool Support for Scrum. *Unpublished*.

Morgan, R., & Maurer, F. (2006). Maseplanner: A card-based distributed planning tool for agile teams. Global Software Engineering, 2006. ICGSE'06. International Conference on, 132–138. IEEE.

Oman, P., & Pfleeger, S. L. (1997). *Applying software metrics* (Vol. 46).

Petersen, R. R., & Wiil, U. K. (2008). Asap: a planning tool for agile software development. Proceedings of the nineteenth ACM conference on Hypertext and hypermedia, 27–32. ACM.

Pinna, S., Mauri, S., Lorrai, P., Marchesi, M., & Serra, N. (2003). XPSwiki: An agile tool supporting the planning game. Extreme Programming and Agile Processes in Software Engineering, 104–113. Springer.

Rubinstein, D. (2007). Standish Group Report: There's less development chaos today. *Software Development Times, 1*.

Schwaber, K., & Sutherland, J. (2011). The scrum guide. *Scrum Alliance*.

Susman, G. I., & Evered, R. D. (1978). An assessment of the scientific merits of action research. *Administrative science quarterly*, 582–603. JSTOR.

Sutherland, J. (2010). Scrum handbook. Boston: Scrum Training Institute Press.

Sutherland, J., Viktorov, A., Blount, J., & Puntikov, N. (2007). Distributed scrum: Agile project management with outsourced development teams. System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, 274a–274a. IEEE.

TargetProcess inc. (2008, April 3). Agile Tools *http://www.targetprocess.com*, The Good, the Bad and the Ugly. Retrieved April 4, 2013,

van Solingen, R., & van Berghout, E. (1999). *Goal Question Metric (GQM) Approach*. A Practical Guide for Quality Improvement of Software Development. New York: McGraw-Hill.

VersionOne. (2012). State of Agile Survey 2011. *http://wwwversiononecom/*, 1–12.

Wang, X., & Maurer, F. (2008). Tabletop AgilePlanner: A tabletop-based project planning tool for agile software development teams. Horizontal Interactive Human Computer Systems, 2008. TABLETOP 2008. 3rd IEEE International Workshop on, 121–128. IEEE.

West, D., & Grant, T. (2010). Agile Development: Mainstream Adoption Has Changed Agility. *Forrester*, 1–22.