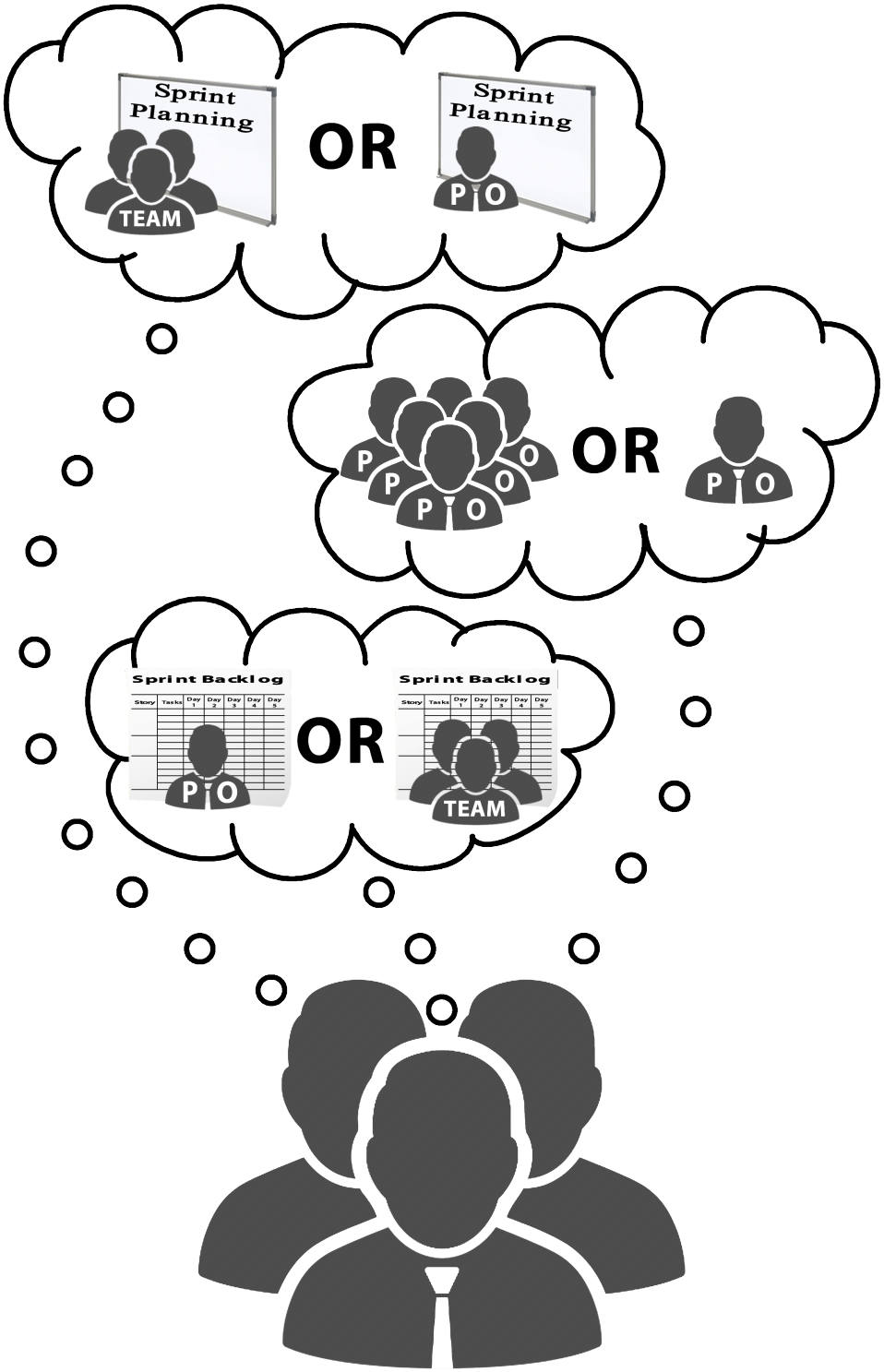
ScrumBut in Professional Software Development



Aalborg University – Department of Computer Science
Master's Thesis – Group IS1016F16 – Spring 2016

AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Selma Lagerløfs Vej 300
9220 Aalborg Ø
Phone (+45) 9940 9940
Fax (+45) 9940 9798
http://www.cs.aau.dk

Title:
ScrumBut in Professional Software Development

Subject:
Systems Development

Project period:
2016-02-01 – 2016-08-22

Project group:
IS1016F16

Participants:
Alexander Drægert
Dan Petersen

Supervisor:
John Stouby Persson

Printings:
4

Pages:
56

Appendices:
26

Total pages:
96

Abstract:

Agile methodologies are widely used in the software industry with Scrum being the most common framework. Scrum has few, well-defined practices, but many companies still deviate from the textbook version. Limited research has investigated the underlying explanation of why. Against this backdrop, we report an investigation of ScrumButs in professional software development based on a multimethod research approach analysing 17 empirical research papers on Scrum modifications and interviews with 9 Scrum practitioners from 7 software companies. ScrumButs were identified pertaining to all parts of the Scrum framework, and the analysis shows how particular ScrumButs may involve different forms of reasoning reflecting different forms of organisational culture. We discuss how these findings may nuance our assessments of ScrumButs beyond implicit value judgments of being inherently good or bad, and how the agility of the modified practices may be affected.

Summary

Software development is a complex endeavour with numerous variables determining whether or not a project is successful. To manage these variables a wide range of methodologies have been developed. Amongst these exists a set of methodologies and frameworks known as *agile*. In this study we focus on the most widely used agile framework *Scrum*, and specifically on the modification or omission of Scrum practices, known as *ScrumBut*.

ScrumBut is an established phenomenon in the Scrum community, however it is still being discussed whether this phenomenon is harmful or benign. Previous research of ScrumBut has yielded many reasons, narrated by software professionals, for prevalent instances of ScrumBut. However, limited research have investigated the underlying explanation of why ScrumBut emerges.

To investigate this we conduct a Grounded Theory study with data collected from 17 empirical research papers on Scrum modifications and interviews with 9 Scrum practitioners from 7 software companies.

ScrumButs are identified in all practices of the Scrum framework and presented along with the reasoning behind, as reported by Scrum practitioners and empirical research papers. Further analysis show that these reasonings can be associated with the competing values model of organisational cultures. The competing values model is a framework through which four different forms of organisational culture can be distinguished based on the core values pertaining each culture. To illustrate this association, we show how particular ScrumButs may involve different forms of reasoning reflecting different forms of organisational culture.

By relating these finding to existing literature we confirm the findings of those who investigate the different variations of ScrumBut by identifying similar variations. Additionally, existing literature focusing on tailoring of agile methodologies is extended by providing a better basis for understanding the conditions in which adaptations are made, through a presentation of the reasonings behind the adaptations of Scrum. Finally, the association between ScrumButs and organisational culture breaks with the current discussion of whether ScrumButs are harmful or benign, and moves it towards a more nuanced point of view, taking the organisation's core values into account.

A condensed and more focused representation of the findings and contributions of this study has been compiled into an article at the end of the report.

Preface

This report was written by two 10th semester software engineering students at Aalborg University. It combines the preliminary findings from 9th semester with a more thorough study of relevant literature and several interviews.

A big thank you goes out to everybody who allowed us to interview them – without you the project would not have been very interesting at all.

We, of course, also want to thank our supervisor, John S. Persson, for his help, guidance and not least patience throughout the project.

The main findings of the report can be found in the form of an article in Appendix F.

Aalborg, August 22, 2016

Alexander Drægert

Dan Skøtt Petersen

Contents

1 Introduction

Software development is a complex endeavour with numerous variables determining whether or not a project is successful. Tiwana and Keil [92] present six risk factors which impact the success rate of a project. Among those risk factors, the one with the highest impact factor is *Methodology Fit*. They argue that there is no "one size fits all methodology" and one methodology may fit one project, but not another. Thus it is important to be aware of what different methodologies entail in order to select the correct fit for one's project.

It has long been commonly agreed that methodologies can be categorised into two categories: heavyweight and lightweight. In heavyweight methodologies a sequential series of steps is followed resulting in a "big bang" delivery of all functionality at the same time. They measure success according to the conformance with a requirements specification which is compiled early in the project and stays relatively stable throughout the project. Thus, they have an anticipatory style of development attempting to anticipate and plan for risks and tasks [58, 64]. Some common heavyweight methodologies are: The Spiral Model, Rational Unified Process (RUP), Incremental Model, and Waterfall Model. Lightweight methodologies are conducted by following a iterative series of steps resulting in an incremental delivery of functionality over time. They measure success according to the business value delivered by relying on frequent customer interactions and rapid changes to the requirements. Thus they have an adaptive style of development attempting to adapt to the emerging risks and tasks [58, 64]. Some common lightweight methodologies are: Agile, Prototype Model, and Rapid Application Development (RAD). As of 2015 Hewlett-Packard Development Company [38, p. 1] found that *"[t]he vast majority of organisations [...] use Agile"* out of all methodologies, which makes it particularly interesting to study.

The Agile methodology first got its name, when Alistair Cockburn expressed dissatisfaction with methodologies being labelled "lightweight", at the meeting from which the "Agile Software Development Alliance" emerged [29]. This meeting was held to better define the common values of these "lightweight" (now agile) methodologies. In attendance was a large group of prominent people from the community (Beck et al. [6]), who in collaboration described the values and principles of Agile and thus created the *Manifesto for Agile Software Development* [6]. The set of values and principles which they introduced was later used by Conboy [17] to define what it means for a method or methodology to be *agile*. The definition, values,

and principles of agile software development are further explained in Section 2.1. Some of the most common agile frameworks and methodologies are, according to VersionOne [98]: Scrum, Extreme Programming (XP), Lean Software Development, Feature Driven Development (FDD), and Crystal. Leading the poll is Scrum with nearly 70%, making it the most widely followed agile methodology, and as such, research into Scrum is paramount.

The methodology Scrum was first defined in 1997 by Jeff Sutherland and Ken Schwaber in the paper "SCRUM Software Development Process" [77]. It is an agile methodology which focuses on managing complex product development projects. The motivation for Scrum is allowing the team to react to changes in requirements and other aspects during the project's execution [21]. When problems or changes occur, the change is inspected and the team adapts as needed. This learning process of "inspect and adapt" is a core element of Scrum in addition to "transparency" as encouraged by the components of the framework. The Scrum framework consists of three roles, five events, and three artefacts which are described in Section 2.2.1.

This limited amount of components makes Scrum appealing to adopt and in some cases customise. Looking at the numbers from VersionOne [98] one can extrapolate that 23% of Scrum followers extend the framework with additional methodologies or practices. This is what the community calls ScrumAnd [97] while changes to Scrum components are called ScrumBut [78]. Especially ScrumBut is an interesting phenomenon as it is often seen in professional software development [22, 45, 70, 82], while the community still argue whether they are "signs of dysfunction" [78] or "a natural part of any agile methodology" [19]. Both of these phenomenon are further explained in Section 2.2.2.

These changes to the methodology is in the research community known as *method adaptation*, *method configuration*, *method customisation*, *method engineering*, *method modification*, *method tailoring*, *situational method engineering*, etc. The difference between these terms are not clear and can be different depending on the author of a given paper. Ågerfalk and Fitzgerald [3], Baskerville and Stage [5] discovered that two distinct views on method tailoring exist. Aydin et al. [4] furthered this discovery by distinguishing these views as *static* and *dynamic* method tailoring. He also found that these views often used different terms when describing the phenomenon method tailoring.

Static method tailoring share some assumptions with the plan-driven methodology, as it assumes that the context is static and structured, disregarding any progress in the method during the project, as they configure or engineer the methods [53, 55]. In static method tailoring it is often method engineers, project management offices, and the like that tailor the methods [4]. This often leaves the developers as method users, who just use the methods as instructed [54, 55]. When describing static method tailoring the commonly used terms are: *Method configuration*, *method engineering*, *method modification*, etc.

Dynamic method tailoring share some assumptions with the agile methodology, as they assume that the context is ever-changing and the method has to change with it [28]. The focus of dynamic method tailoring is on *"how methods are enacted in practice"* Karlsson [52, p. 14], and as such it is often both method users, i.e.

developers, and method engineers that tailor the methods [4]. In dynamic method tailoring the commonly used terms are: *method adaptation*, *method customisation*, *situational method engineering*, etc.

Aydin et al. [4, p. 26] further state that *"the [dynamic] tailoring process often is ill-structured so it becomes difficult to document changes and their rationale"*. This may also answer as to why there is so limited empirical research into ScrumButs. To contribute to the body of research on adaptations of Scrum, this study will explore the emergence and reasoning of ScrumButs in professional software development. To guide the study, the following research questions are proposed:

I. *What ScrumButs are prevalent in professional software development?*
II. *How are ScrumButs reasoned in professional software development?*

To study these research questions we use a pluralist methodology as described by Mingers [65], as his research suggests that this will give a richer understanding of the results. We follow a Grounded Theory approach, described by Wolfswinkel et al. [101], to analyse existing literature supplemented with in-depth laddering interviews of practitioners, described by Schultze and Avital [76]. A more detailed explanation about the use of research methods in this study is described in Chapter 3.

The remainder of the report is structured as follows: In Chapter 2 some necessary information about Agile Software Development, Scrum, and ScrumButs is presented. In Chapter 3 we present the argumentation and execution of our research methodology. This is followed by Chapter 4 where we present our findings. In Chapter 5 we discuss the contribution and limitations of the study, as well as suggestions for future research. Lastly, in Chapter 6, we conclude the study.

Agile Software Development, Scrum & ScrumButs 2

This chapter outlines the research conducted in the area of agile software development. In addition, Scrum is described to provide a frame of reference for modifications to the framework. Finally, ScrumBut is explained while presenting research and disputes regarding the phenomenon.

2.1 Agile Software Development

The Agile Manifesto [6] consists of four values and twelve principles to support them. The values are:

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

Notice that the values to the right are still considered important, but not as important as the values to the left. Along with the four values, the manifesto formulates twelve principles to follow in agile software development, but as argued by Dingsøyr et al. [23], the principles do not constitute an actual definition of agility. In the years following the publication of the agile manifesto, several researchers started defining the term (e.g. [37, 62]). Focusing on information systems development (ISD), Conboy [17, p. 340] defines agility as *"the continued readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality, and simplicity), through its collective components and relationships with its environment."* This definition is widely used (e.g. [49, 71]) and while the focus is on ISD, it is applicable to software development in general; Dingsøyr et al. [23, p. 1214] call it *"by far the most comprehensive definition of software development agility."* In addition to the definition itself, Conboy [17] formulated a taxonomy (see Table 2.1) that is useful for determining whether or not a given method component can be considered agile.

Besides studying agility as a concept, other subjects are also being studied. Many study individual practices or methodologies (e.g. [43, 66, 84, 100]), and in a recent example Eloranta et al. [25] study Scrum anti-patterns. Others look at

1. To be agile, an ISD method component *must* contribute to one or more of the following:
   a) creation of change
   b) proaction in advance of change
   c) reaction to change
   d) learning from change
2. To be agile, an ISD method component *must* contribute to one or more of the following, and *must not* detract from any:
   a) perceived economy
   b) perceived quality
   c) perceived simplicity
3. To be agile, an ISD method component *must* be continually ready, i.e., minimal time and cost to prepare the component for use.

Table 2.1: Taxonomy of ISD Agility [17, p. 341].

the relationship between organisational agility and project agility (e.g. [9, 71]) or the impact of organisational culture on agile method use (e.g. [49, 86]). Iivari and Iivari [49] propose a number of hypotheses on the subject of the latter. Dingsøyr et al. [23] notes a growing interest in combining agile software development with principles of lean software development; a trend that continued after their study (see e.g. [73, 87]).

Another common subject, the one in focus in this project, is that of agile methodology tailoring. Some focus on tailoring specific methodologies (e.g. [10, 13, 19]), while others focus on how to tailor agile methodologies in general (e.g. [12, 18, 39]). An example of tailoring specific methodologies can be seen in Conboy and Fitzgerald's [19] study in which the objective is to *"assess how amenable XP is to tailoring, and to develop a set of recommendations for its improvement in this regard"* and *"investigate how developers are undertaking XP tailoring efforts and to develop a set of best practices for developers to follow"* [19, p. 2:2]. Hoda et al. [39] study tailoring of agile methodologies in general by relating adaptations from different sources [40, 41, 42] to the context of their independent projects, and they find that agile teams *"evolve their methods and practices to be as effective as possible within their projects' contexts"* [39, p. 86].

2.2 Scrum

Takeuchi and Nonaka's [88] paper "The New New Product Development Game" discusses the need for a new approach to product development. Using the rugby term *scrum*, they presented the philosophy of an approach using self-organising teams. They found those teams to perform better than traditional hierarchical teams. Their paper ultimately led to the development of a framework to support this philosophy, and sticking with the analogy from the original paper it was named *Scrum* [77, 81]. Since then several improvements have been made, and today Scrum is used by the majority of organisations practising agile software development [81, 98]. Many of

the organisations claiming to do Scrum, modify it to fit the context in which they are working. Schwaber [78] calls these changes *ScrumButs* and their significance will be discussed in Section 2.2.2, after the presentation of Scrum itself in Section 2.2.1. A previous version of these two sections can be found as Chapter 2 of our 9th semester project report [24].

2.2.1 What is Scrum?

Scrum (noun): *A framework within which people can address complex adaptive problems, while productively and creatively delivering products of the highest possible value.*

[80]

Several books and papers exist on the subject, providing a much more detailed account of the framework [7, 60, 74, 79]. To keep the discussion on an abstraction level simple enough to cover the entire framework, this study will focus on the description in "The Scrum Guide" [80]. Scrum, as described in "The Scrum Guide", consists of 3 roles, 5 events, and 3 artefacts. The following serves as an overview of the ideal use of Scrum according to said guide, which is chosen because it gives a relatively simple overview of the framework, and is written by the people who formalised the first version of Scrum.

Scrum Roles

The Scrum Team consists of a Product Owner, a Scrum Master, and the Development Team. It is self-organising, meaning it is responsible for directing its own work. It is cross-functional, meaning it has all the expertise needed in the team, without having to rely on, for example, external consultants.

Product Owner The Product Owner is responsible for managing the Product Backlog, including making sure Product Backlog items are clearly defined, understandable and ordered. The Product Owner has full control over the requirements, and nobody else is allowed to change what the Development Team works on.

Scrum Master The Scrum Master enforces proper Scrum practice and manages communication with people outside the Scrum Team. The Scrum Master also serves as facilitator of Scrum events, and is ultimately responsible for making sure the Development Team is able to do their job with as few impediments as possible.

Development Team The Development Team should have between three and nine developers, and it is their job to turn the Product Backlog into Increments of potentially shippable product, i.e., "Done" product. Aside from receiving guidance from the Scrum Master on how to properly do Scrum, the Development Team only answers to the Product Owner. There is no hierarchy on the team – everyone has the title *developer* regardless of their function.

Scrum Events

The 5 Scrum Events, shown in Figure 2.1, are used to reduce the need for ad-hoc meetings and to create regularity in the work flow. All Scrum Events are time-boxed to avoid unnecessary discussions.

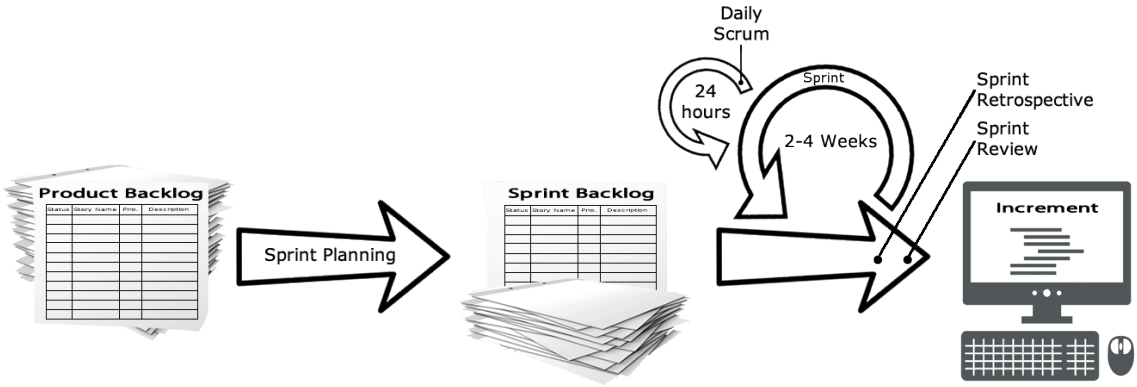


Figure 2.1: Overview of events and artefacts in Scrum.

Sprint A Sprint is an event with a duration of up to a month, in which a "Done" product increment is created. A new Sprint starts immediately after the previous Sprint ends. Changes to the Sprint Backlog during a Sprint should be avoided, but the Product Owner and Development Team are allowed to renegotiate the scope, without endangering the Sprint Goal or quality goals. The Product Owner may cancel the Sprint, should the Sprint Goal become obsolete.

Sprint Planning Sprint Planning is a collaborative effort where the Scrum Team works together in turning Product Backlog items into a Sprint Backlog, as well as creating a Sprint Goal. In a Sprint Planning meeting it is decided what can be delivered at the end of the Sprint and how to accomplish that goal. The meeting is time-boxed to a maximum of eight hours.

Daily Scrum In the Daily Scrum the Development Team members present what they did since the last meeting and plan what they will do the next 24 hours. Impediments that prevent meeting the Sprint Goal are presented as well and a possible solution is found either during or after the meeting. The meeting is time-boxed to a maximum of 15 minutes, and should be held the same time and place every day. The Daily Scrum can be followed by a more detailed discussion to adapt the rest of the Sprint.

Sprint Review The Sprint Review is a meeting between the Scrum Team and key stakeholders, held at the end of each Sprint. The Product Owner presents the progress made during the Sprint, and the Development Team goes over problems as well as what went well, followed by a demonstration of the new features. Afterwards, the Product Backlog is revised, and business related topics are discussed if needed. The Sprint Review should produce a revised Product Backlog, including probable items to include in the following Sprint. The Sprint Review is time-boxed to a maximum of 4 hours.

Sprint Retrospective The Sprint Retrospective takes place between the Sprint Review and the following Sprint. It is time-boxed to a maximum of 3 hours, and only members of the Scrum Team participate. The objective is to find out what went well during the sprint and what can be improved, and to create a plan for how to implement the improvements.

Scrum Artefacts

The Scrum Artefacts are used to create transparency in the development process. They serve as ways to manage what has to be done, work in progress, and the finished product. The Scrum Artefacts are shown in relation to the Scrum Events in Figure 2.1.

Product Backlog The Product Backlog is an ordered list of items the Product Owner determines might be needed in the finished product. Each item is estimated by the Development Team, but the Product Owner has full responsibility of the actual contents: description, prioritisation, estimation, and expected value. The Product Backlog should be refined regularly, but effort should not exceed 10% of the Scrum Team's time. If it is relevant for the project, several Scrum Teams may share one Product Backlog.

Sprint Backlog The Sprint Backlog is the result of the Sprint Planning event. It contains a description of the Sprint Goal and a list of items from the Product Backlog the Development Team find should be implemented to reach it. Only the Development Team is allowed to alter the Sprint Backlog during a Sprint, and during the Daily Scrum they asses it to decide how likely it is they will reach the Sprint Goal. The Sprint Backlog should constantly be updated to reflect progress, new tasks, finished or unnecessary items, etc.

Increment An Increment contains the items completed in the current and previous Sprints. The Increment only contains "Done" items, regardless of whether they will be released or not.

2.2.2 What is ScrumBut?

ScrumBut (noun): A modification or omission of a Scrum practice following the syntax (We use Scrum, but)(Reasoning)(Workaround).

[51, 78]

The term ScrumBut was coined in 2006 by Gunnerson, and is a prevalent occurrence when studying Scrum practices in professional software development, as seen in for example [36, 61, 70]. Research into this phenomenon often focuses on what instances are prevalent and their relation to Scrum [22, 25, 45]. So far there is a limited amount of research explaining why ScrumButs emerge.

An example of a ScrumBut is given by Schwaber [78] as *"(We use Scrum, but) (having a Daily Scrum every day is too much overhead,) (so we only have one per week.)"*

Schwaber [78] argues that ScrumButs are *"exposing dysfunctions that contributes to problems"* and that allowing ScrumButs *"retains the problem while modifying Scrum to make it invisible."* Others have different views on ScrumButs as for example Kniberg [59] who argues that *"anything that works for you is right, anything that doesn't is wrong"* and one should not be afraid of doing Scrum wrong if it works, a sentiment Jeffries [51] agrees with. Conboy and Fitzgerald [19, p. 2:4] further argue that *"anything labelled as agile should itself be flexible and amenable to tailoring."* The different views are seen throughout the professional software development industry, with supporters of both arguments [18]. Despite this controversy, limited studies into the practical effects of ScrumBut have been conducted. Heikkilä et al. [36, p. 94] have conducted a study on ScrumButs regarding user story management and sprint planning, concluding that changes to these *"cannot be said to be categorically harmful ScrumButs."*

In addition to ScrumBut, another term has been coined in regards to modification of Scrum. This is the term *ScrumAnd* which is used when *extending* Scrum with additional practices, for example implementing XP practices such as Pair Programming in collaboration with Scrum. Although differentiating between ScrumBut and ScrumAnd seems easy, not all cases are straight forward. As such Krishna and Basu [61] developed a set of principles to determine if a deviation from Scrum is a ScrumBut or a ScrumAnd. To easier use these principles to determinate which concept a given instance should be defined as, they developed three tables, shown here in Table 2.2, Table 2.3, and Table 2.4.

Open-Closed Principle

Open for extension, additional feature	Closed for modification in Scrum practices	OCP Result	Status
Yes/No	Yes	Yes	Scrum or ScrumAnd
Yes/No	No	No	ScrumBut

Table 2.2: Open-Closed Principle Table [61, p. 3].

Single Responsibility Principle

Single Responsibility	Multiple Responsibility	SRP Result	Status
Yes	No	Yes	Scrum
No	Yes	No	ScrumBut

Table 2.3: Single Responsibility Principle Table [61, p. 3].

Applying OCP and SRP

OCP	SRP	Status
Not Applicable	Not Applicable	Scrum
No	No/Not Applicable	ScrumBut
Yes/Not Applicable	No	ScrumBut
Yes	Not Applicable	ScrumAnd
No/Not Applicable	Yes	Scrum
Yes	Yes	ScrumAnd

Table 2.4: Application Table [61, p. 3].

To illustrate the use of these tables, the concept of the instance "we use Scrum, but/and we use a product owner committee instead of a single product owner" can be determined. First we look at Table 2.2 to see whether Scrum allows for modification of this practice. The Scrum Guide [80, p. 5] says *"the Product Owner is one person, not a committee,"* thus it is not amenable for modification and the OCP Result is *No*. Secondly, we look at Table 2.3 to see if the role in question has more than a single responsibility as prescribed by The Scrum Guide [80]. In this case there are multiple role holders which divides the responsibility, and thus the answer to *Single Responsibility* will be *No*, prompting the SRP Result to also be *No*. Lastly, we look at Table 2.4 to determine if the statement is a ScrumBut, ScrumAnd or just regular Scrum. The results from Table 2.2 and Table 2.3 was *No* and *No*, respectively, resulting in the statement being defined as *ScrumBut*. In this study we focus only on ScrumButs. Scrum does not account for every part of the development of a product, and adding practices is therefore natural.

Research Methodology 3

The term *methodology* should not be confused with the term *method*. A *method* is, according an interpretation of [16, 47] by Mingers [65, pp. 241–242], a *"well defined sequence of operations that if carried out proficiently yield predictable results."* A *methodology* can be defined by one of three definitions:

1. The study of methods.
2. The combination of methods used in a given project.
3. A predefined combination of methods collected under a name.

In this study we will use to the second definition of *methodology*.

Before selecting or developing a methodology to follow in this study, we first looked at the questions we try to answer. These questions are open inquiries into what ScrumButs are prevalent and how these ScrumButs are reasoned in professional software development. Given these questions it is ideal to collect data from a professional software development environment. As there exists several methods to do this, we looked into recommendations for conducting studies. We found that Mingers [65] recommends combining multiple methods when developing our methodology. Using two or more methods based on different paradigms allows for a richer and more nuanced understanding of the problem at hand. By viewing the problems through different lenses, details that are overlooked from one point of view may be spotted from another, allowing the different views to support each other, thereby giving a richer understanding of the research results. Mingers [65] points out, that of the studies that do use different methods, many stay within the same paradigm and/or use methods that are traditionally linked closely together, mentioning, for example, interviews and case studies. The used methods and their results are not claimed to be bad, but a more nuanced conclusion could likely be gained by exceeding one paradigm.

Heeding Mingers' [65] recommendation we decided to collect data from both practice and academia. To collect this data we decided to develop a methodology based on Schultze and Avital's [76] *laddering* interview technique and Wolfswinkel et al.'s [101] approach to Grounded Theory. Tailoring these two methods can be done in numerous ways. We have listed and discussed some of these in Appendix A.1. After consideration and the recommendation from Adolph et al. [2], not to get caught

up in choosing a configuration, we decided to use the configuration illustrated in Figure 3.1, which uses both methods concurrently, while combining the results in one shared NVivo (data analysis) project. This configuration is the fourth discussed in Appendix A.1.

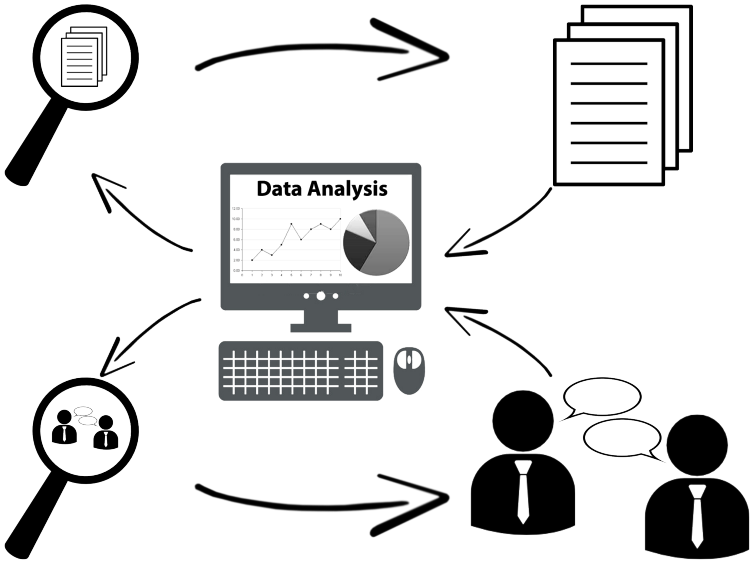


Figure 3.1: Overview of events and artefacts in Scrum.

In practice, the criteria for the literature search and potential interviewees are defined according to Wolfswinkel et al.'s [101] and Schultze and Avital's [76] instructions. The data collection is performed and the resulting data is collected and analysed in NVivo. This process continues until a saturation point is reached, i.e. until no significant new knowledge is obtained.

3.1 Methods

In this section we introduce the rationale for selecting the two methods in our methodology, as well as theories, principles and instructions pertaining to these methods.

3.1.1 Grounded Theory

Grounded Theory is so named because the method is used to generate a theory which is *grounded* in the data [32]. This theory is a set of integrated conceptual hypotheses that explain a substantive area [31]. The concepts and categories that integrate these hypotheses are used to account for patterns of behaviour which are relevant to the substantive area [30]. The unique thing about Grounded Theory is that it first collects data and then systematically develops a substantive theory directly from the data. In contrast, other logico-deductive research methods develop a theory without relying on data, and then systematically collect data to substantiate or test the validity of this theory [32].

The research questions for this study are open and are not pre-formulated theories needing validation. Instead they are inquiries about the understanding, realisation, and reasoning of a process. This kind of inquiry is highly suited for Grounded Theory as, according to Schreiber and Stern [75, p. 13], *"the research questions that begs to be answered through Grounded Theory is: What is going on here?"* They further argue that *"Grounded Theory best analyses processes and identifies complex and hidden processes"* which is exactly what this study does. Although Grounded Theory has a long history of usage in empirical studies of social phenomenons and social studies, the method only started being used in software engineering research in recent years. Adolph et al. [2, p. 488] states that *"Grounded Theory is an excellent method for studying software engineering and generating theories,"* confirming it not to be immature for this field of study. They further state that *"Grounded Theory is useful for research in areas that have not been previously studied or where a new perspective might be beneficial"* [2, p. 491] which further warrants the use of Grounded Theory in this study.

When investigating Grounded Theory one quickly discovers that numerous versions of the method exist. The most common versions are *Glaserian Grounded Theory* [30, 32], *Straussian Grounded Theory* [20, 85], and *Constructivist Grounded Theory* [14, 15]. It is debated in the Grounded Theory community which method(s) should be called "real" Grounded Theory and which one is the "best" [11, 57]. In software engineering there is a tendency to use Straussian Grounded Theory, and Adolph et al. [2] argue that this might be because this approach is the most procedural. They do, however, give the following recommendation: *"Do not get caught up in the debate over which method is the "best" Grounded Theory method. The best method is the one that has the most resource and support availability. Nonetheless, be clear which method you are employing in your study"* [2, p. 509]. Following this advice, we have chosen to follow Wolfswinkel et al.'s [101] approach to Straussian Grounded Theory, due to its comprehensive guide to conducting literature studies.

Wolfswinkel et al.'s [101] Grounded Theory consists of five steps which can be seen in Table 3.1 with the prefix "GT". The table also shows the steps included in the interview part, which are explained in Section 3.1.2. It is imperative to know that despite the representation being staged, the process is iterative. As, for example, the data collection (step 1-3) is performed numerous times until a saturation point has been reached. This point is reached when collecting more data does not reveal any new categories in the analysis and only supports existing ones. The analysis is iterative as well, as each new category leads to a re-evaluation of all existing higher-order categories, thus influencing the theory.

Categories are created during the analysis step which is divided into three substeps: open coding, axial coding, and selective coding. Wolfswinkel et al. [101] suggest that before starting to analyse the data, to first mark the areas containing useful data. This focuses the analysis effort on the useful parts of the data and prevents the researchers to be distracted or waste time on information not pertinent to the study.

After preparations are made the open coding begins. This step involves identifying *"a set of categories or a bird's eye image of the study's findings"* [101]. This

abstraction enables us to build a set of concepts derived from numerous different sources, without adhering to each specific source. In this study we conceptualised the ScrumButs and their respective justifications into categories as open codes.

Grounded Theory

GT1. Define
- a. Define inclusion/exclusion criteria
- b. Define research field
- c. Define sources
- d. Define search terms

GT2. Search
- a. Perform search

GT3. Refine sample
- a. Perform sample refinement

Interview

I1. Define
- a. Identify relevant interviewees

I2. Contact
- a. Contact relevant interviewees

I3. Prepare
- a. Create/update interview guide

I4. Perform
- a. Perform interview

GT4. Analyse
- a. Open coding
- b. Axial coding
- c. Selective coding

GT5. Present
- a. Represent and structure content
- b. Structure article/report

Table 3.1: Step-wise representation of our methodology.

The next step is axial coding with the goal of identifying *"the interrelations between categories and their sub-categories"* [101]. These axial codes or higher-order categories as they also are named, represents the themes or patterns in the findings in the data. To find these existing axial codes are continuously compared and restructured.

The last step is selective coding where we *"integrate and refine higher-order categories"* by *"identifying and developing relations between the higher-order categories"* [101]. It is in this step we theorise about the higher-order categories to answer our research questions.

3.1.2 Laddering Interview

An interview makes it possible to *"get a first-person account of the participant's social reality"* [76, p. 2]. In an interview, views are exchanged between the researcher and a respondent, but according to Grunert and Grunert [33, p. 212] the outcome should *"be a result more of the respondent's cognitive structures and processes than of the researcher's cognitive structures and processes."* To accomplish this, Grunert and Grunert [33] calls for open methods (e.g. unstructured interviews), that allow the respondent to answer with *natural speech*.

Conducting interviews is a good way to generate data for this study, as it allows for gaining detailed insight into the respondent's reality. For interviewing single individuals Myers and Newman [67] describe two types of interviews: structured and unstructured or semi-structured. The structured interview has a strict script, and leaves no room for improvisation. In an unstructured or semi-structured interview there is little to no script, and improvisation is needed. For this study, the semi-structured interview is ideal. Coupled with the Grounded Theory approach, a semi-structured interview can help us make sure all important aspects of Scrum are covered while still allowing the respondents to express their reality as they experience it.

Schultze and Avital [76] present three types of interviews: appreciative, laddering, and photo-diary. A more detailed discussion of the approaches can be found in Appendix A.2. The approach found to be most appropriate for this study was the laddering interview, as it allows for a more comprehensive understanding of the interviewee's reality. Schultze and Avital [76] describe a version, that includes use of the Repertory Grid method, however, as we are not interested in a comparison of interviewee's different views, we focus only on the second part of the approach, namely the laddering process. In the laddering process the respondent is encouraged to elaborate their answers, by being asked to explain their previous answer. In the context of this study, loosely based on the example given by Schultze and Avital [76, p. 9], a respondent may state that they have no Scrum Master, and the interviewer will ask why that is the case: "why do you not have a Scrum Master?" The respondent could answer "we prefer to have a project manager instead", and the interviewer continues "why do you prefer to have a project manager?" This process continues as long as meaningful answers can be given.

3.2 Data Collection

In this section we present the criteria, decisions, and definitions which we base our data collection on.

3.2.1 Literature Study

This study is a continuation of a pilot study conducted and presented in Drægert and Petersen [24]. The first step in a literature study, when following Wolfswinkel et al.'s [101] approach to Grounded Theory, is to define the search parameters. Hence the following paragraphs will describe this study's search parameters.

Inclusion/Exclusion Criteria The primary objective of these criteria is to filter out papers not relevant to this study while including papers which are. Therefore criteria such as outlet reputation, paper citations, or the quality of theory presented in a paper is not regarded, as the focus is solely on empirical data pertaining to ScrumButs.

Inclusion criteria

- Papers published in or after 2001, due to the first book published about Scrum by Schwaber and Beedle [79].
- Papers stating the use of an empirical research approach.
- Papers presenting data of ScrumButs from a professional software development environment.
- Papers presenting circumstances leading to the experienced ScrumButs.
- Papers written in English.

Exclusion criteria

- Papers comparing frameworks, practices, methods and/or methodologies.
- Papers with data collected from educational institutions or other non-professional software development environments.
- Papers focusing on tools, models, or metrics.
- Papers solely proposing and/or testing newly developed frameworks.

Define Research Field We focus on the field of Computer Science, as it is here we can obtain observations about the professional software development industry. Although papers concerning Scrum is also to be found in other research fields, e.g. management, we have discovered that the papers regarding Scrum in software development settings are listed in the research field of Computer Science as well as other research fields. This observation lets us disregard other research fields without losing relevant data, as the papers we exclude will focus on the theory of Scrum and not the usage in a professional software development setting.

Define Sources We have selected the four largest software engineering digital libraries to query. These digital libraries are all well regarded in the field and contain only papers that have been peer-reviewed to ensure quality [1, 26, 46, 91]. The four digital libraries are:

- Association for Computing Machinery (ACM) Digital Library
- Institute of Electrical and Electronics Engineers (IEEE) Xplore
- Scopus
- Web of Science

Define Search Terms The search terms are developed as an aggregation of "Scrum" and research field terms synonymously with "tailoring". These terms will be used to search titles, abstracts, and keyword for matches. The search terms are:

- Scrum AND { adapt*, customi*, modif*, tailor*
- ScrumBut

Search Results Performing the search with the specified definitions results in 388 papers[1]. As per Wolfswinkel et al.'s [101] instructions, these 388 papers were refined until only 17 papers, meeting the specified criteria, remained. The instructions for this refinement consist of five steps, which are shown in Figure 3.2.

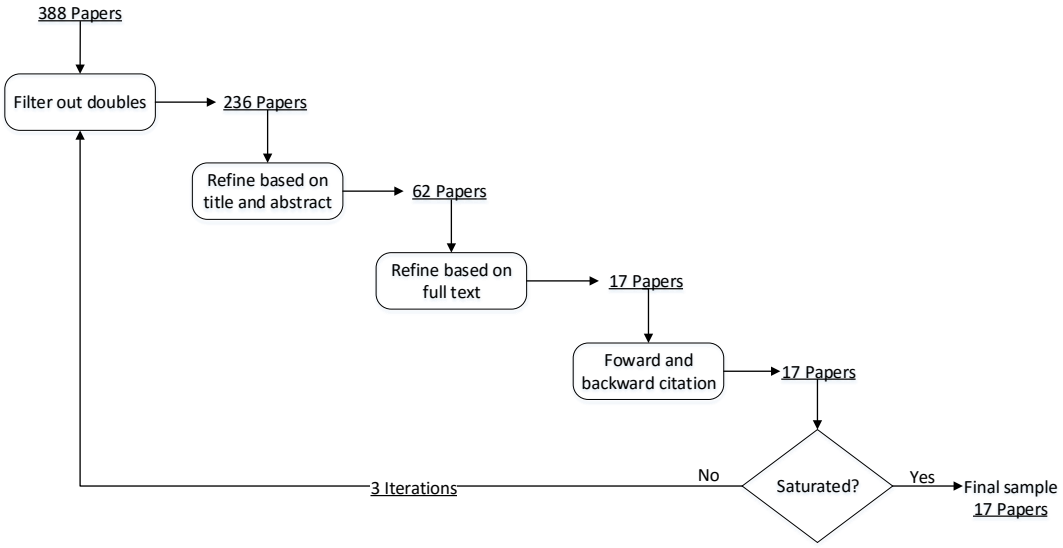


Figure 3.2: Literature search refinement process.

As seen in the figure, the process restarted 3 times as a result of the analysis not reaching a saturation point. When a new iteration of literature search started, the criteria and parameters were broadened. To make the refinement process easier, all papers from each digital library were stored in the research tool *Zotero*[2]. The final list of refined papers can be found in Appendix C.

3.2.2 Interview Study

Where meta-data for a literature study is available and searchable, meta-data for interviews is more difficult to obtain. In other words, it is hard to make sure the interviewees fulfil the required criteria before the interview. As with the literature study, the first step in finding the right candidates is to define the criteria. On a general level we are interested in candidates who can help us understand ScrumBut

[1]The exact queries can be found in Appendix B
[2]A screenshot and description of Zotero can be found in Appendix E.1.

and its emergence in practice. In this section we cover which criteria a candidate need to fulfil, present the final candidates, and how the interviews were conducted.

On an individual level, the ideal candidate:

- is a member of the Scrum Team,
- has thorough knowledge of how the team does Scrum, and
- knows if, which and why adaptations have been made.

Overall, the candidates should consist of people with different roles on the Scrum Team, different amounts of experience, and from multiple companies. To get an idea of how adaptations are perceived from the developers' point of view compared to that of the Scrum Master/Product Owner, having both a developer and Scrum Master/Product Owner from the same team is ideal.

Table 3.2 shows the final list of interviewees. The interviews are listed in chronological order, except for Interview X which is moved to the bottom as it is not used directly in the findings. Interviews 2 and 3 are listed as the same organisation, but the interviewees come from different departments, which according to the interviewee from Interview 3 do not have any direct interaction. Interview 2 and X each have two rows in the table: one per interviewee. Team sizes include Scrum Master, Product Owner, and developers. In cases where a developer is also Product Owner or Scrum Master, they are only counted once. Scrum experience indicates the experience of the interviewee, not the team as a whole. The rightmost column shows the reference used in the remainder of the report.

Interview	Scrum Role	Org.	Team Size	Scrum Exp.	Interview Duration	Ref.
1	Scrum Master	α	6-7	3 years	1:32	[I1]
2A	Developer	β	7	13 years	1:38	[I2A]
2B	Project Manger	β	7	8 years	1:38	[I2B]
3	Developer	β	10	2 years	1:14	[I3]
4	Developer	γ	10	12 years	1:05	[I4]
5	Scrum Master	δ	4	<1 year	1:08	[I5]
6	Product Owner	ϵ	7-10	10 years	1:09	[I6]
7	Scrum Master	γ	10	11 years	1:18	[I7]
8	Developer & Scrum Master	θ	3	1 year	1:05	[I8]
9	Product Owner	ω	5-7	10 years	0:55	[I9]
XA	Scrum Coach				2:42	[XA]
XB	Scrum Coach				2:42	[XB]

Table 3.2: Table of interviews.

Besides the 9 software development professionals, we interviewed two highly experienced Scrum coaches (Interview X). This interview was focused on Scrum, so as to strengthen and broaden our understanding of Scrum. During the interview the coaches were kind enough to relate some of their experiences and understanding of ScrumBut, which we used to develop talking points for further interviews.

Before the first interview, an interview guide was put together, detailing a list of questions about the background of the team and interviewee, the important topics to focus on, and some probes in case the conversation should stop. In addition it describes the roles of all participants. After each of the first two or three interviews the interview guide was updated to reflect our new experience, but it is worth noting it was used as a means of keeping the interview on track – not as a manuscript. The final interview guide is shown in Appendix D (Danish).

All interviews were recorded for later analysis, and all followed a similar pattern. First we asked questions about the interviewee, the team, and the company. Then we would allow the interviewee to describe how they do Scrum. When ScrumButs were mentioned, they were noted to be explained later. If the description neglected any Scrum practice, this was also noted for later explanation. After the description, we would start questioning the mentioned ScrumButs and neglected Scrum practices. These questions were repeated, as described in Section 3.1.2, until satisfactory answers were given for all inquires where this was possible. Then we would start a line of questioning regarding ScrumButs discovered in previously analysed interviews or papers, in cases where they had not already been mentioned. Finally, the interview was concluded by allowing the interviewee(s) to ask us questions.

Findings 4

In this chapter, we explore the different ScrumButs and present the patterns discovered when coding the data. Although the coding was done in an iterative manner, as detailed in Chapter 3, the findings presented here represent the final iteration of coding.

The chapter is organised into two sections: In Section 4.1 an overview of the identified ScrumButs is presented along with examples of the immediate reasoning behind them, i.e. the reasons given to explain the ScrumBut. In Section 4.2 the reasonings are related to an organisational culture based on the Competing Values Model to illustrate the association between organisational culture and the emergence of ScrumBut.

When an interview is referenced, it is done as shown in Table 3.2, e.g. the first interview will be referenced as [I1].

4.1 Reasoning of ScrumButs

ScrumButs were identified in all parts of Scrum, and in this section the ScrumButs are presented to provide an overview of the changes made. As in Section 2.2, the ScrumButs are presented in three groups: Roles, Events, and Artefacts.

The ScrumButs are presented in Tables 4.1-4.6. The tables are structured as follows: In the left column a snippet from the Scrum Guide shows how a given part of Scrum should be done, and in the middle column the corresponding ScrumBut is presented along with references to all identified instances. The right column shows the reasoning behind each ScrumBut.

4.1.1 Scrum Roles

The Scrum Team consists of only three different roles – Product Owner, Scrum Master, and Developer – and the Development Team should have between three and nine members [80]. Table 4.1 shows the identified ScrumButs related to the Product Owner and Scrum Master, and Table 4.2 shows those related to the Development Team.

Roles (part 1)		
Scrum [80]	**ScrumBut**	**Reasoning**
"The Product Owner is one person, not a committee."	Having a Product Owner Committee [I1, I8, 8, 25, 22, 35, 93]	People with different stakes in the project need authority [I1]. Decision makers do not want to give up mandate [I8]. The product has several branches [8, 93]. Multiple client relations [35].
The Product Owner is acountable for: "Ordering the items in the Product Backlog."	Product Owner Not Accountable for Ordering of Product Backlog Items [I1]	Wish to keep board of directors accountable for direction of product [I1].
"The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master."	No Product Owner [I1, I2A, I2B, I6, 22, 25, 95]	Product Owner tasks handled by legacy roles [I2B]. Internal tasks can be handled by team lead [I6]. Redundant because of separate product/project management office [I1, 22]. Requirements received directly from client [I2A, 22].
"The Scrum Team consists of a Product Owner, the Development Team, and a Scrum Master."	No Scrum Master [I2A, I8, 22, 27, 50, 95]	It is easier to engage the team without a Scrum Master [27]. Replaced by a Project Manager [50]. Extra resources for development gives more value [22]. The team does not use enough events to justify having a Scrum Master [I8].
"The Scrum Master helps those outside the Scrum Team understand which of their interactions with the Scrum Team are helpful and which aren't."	Scrum Master Not Protecting the Team [I1, 25]	

Table 4.1: ScrumButs related to the Product Owner and Scrum Master.

The Scrum Master and Product Owner are both very central aspects of Scrum, yet as seen in Table 4.1, several examples have been identified of a Scrum Team having *No Scrum Master* or *No Product Owner*. The reasons for not having a Scrum Master include giving the responsibility to another role [50] and not perceiving the role to add enough value [I8, 22]. In one of the cases, where the role is not perceived

to add enough value, their explicit goal is to *"avoid reducing the overall capacity by assigning a developer as full-time Scrum Master"* [22, p. 46]. The reasons for not having a Product Owner all relate to either having other roles taking over the Product Owner's responsibilities [I1, I2B, I6, 22] or simply getting requirements directly from the client [I2A, 22], while *Having a Product Owner Committee* is grounded in not being willing to let one person have the final word [I1, I8, 8, 35, 93]. One interviewee reasoned: *"I think, in our case, it is because neither of us want to give up our authority as we all have an idea of how to run a company–and in which direction we want to take it"* [I8, translated from Danish]. The fact that these roles are so central to Scrum may help explain why so many cases are identified, as more people will have an understanding of how the roles are supposed to be used, increasing the chance of being aware of ScrumButs relating to them. In addition, the roles may clash with roles already existing in the company from before Scrum was used (e.g. [I2B, 50]).

Table 4.1 only shows few identified cases of the *Scrum Master Not Protecting the Team*, which is surprising because that can be a very difficult task to accomplish. Additionally, the few identified cases provide no explanation. It is possible the lack of instances of this ScrumBut is caused by lack of focus in interviews as well as papers – while important, it is arguably not a very visible part of Scrum, and it can easily be forgotten. Another possibility is that Scrum Masters are simply (perceived to be) better at protecting their teams than expected.

Scrum Master Not Protecting the Team as well as *Team Not Self-organising* both have no associated reasonings. The lack of explanation further strengthens the argument that the ScrumButs are not related to a central part of Scrum and may have been forgotten, as the teams using them have not even considered the reason why. The lack of explanation can also be linked to the fact that they, along with *Product Owner Not Accountable for Ordering of Product Backlog Items*, only have one or two identified cases. ScrumButs with few identified cases are included because they help show a more complete picture, and the purpose of these findings is not to show statistical evidence of how often a given ScrumBut appears.

Table 4.2 shows the ScrumButs related to the Development Team. *Having Titles on Development Team* has been identified in several cases, and the explanations are all found in the interviews conducted for this study, although cases were identified in papers as well. Notice that all reasons relate to things outside the power of the team itself; e.g. to ease personnel management [I1] or because it is required by the organisation [I6].

Roles (part 2)		
Scrum [80]	**ScrumBut**	**Reasoning**
The Development team should have between three and nine members.	Exceeding Team Size Boundaries [I1, I8, 22, 45]	The team works on a project that does not require many resources [I1]. The company is very new [I8]. The team was large before Scrum was introduced [22].
"Scrum recognizes no titles for Development Team members other than Developer."	Having Titles on Development Team [I1, I4, I6, 44, 95]	It makes personnel management easier [I1]. Architects are highly skilled/experienced and in high demand [I4]. Dictated by organisation and part of people's identity in the company [I6]. It makes it easier for people to move to different departments [I6].
"Development Teams are cross-functional."	Development Team Not Cross-functional [I4, 25]	Development some times depends on legacy components [I4]. There is a separate testing team [25].
"Scrum Teams are self-organising."	Team Not Self-organising [25]	

Table 4.2: ScrumButs related to the Development Team.

Several instances were identified of *Exceeding Team Size Boundaries*, which indicates the team is either too small or too big. The reasonings show no particular pattern, and it should be noted that the teams in question only barely exceeded the suggested limits of a Development Team size of three to nine developers. The smallest teams reported having two developers (e.g. [I1, I8]) while the largest teams reported having around ten members (e.g. [22, 45]).

4.1.2 Scrum Events

The Scrum Events *"create regularity"* and *"minimize the need for meetings not defined in Scrum"* [80, p. 7]. The Scrum Events play a central role in Scrum, and as Tables 4.3 and 4.4 show that many different ScrumButs were identified, relating to them.

Events (part 1)

Scrum [80]	**ScrumBut**	**Reasoning**
Daily Scrums should take place every day.	Daily Scrums Cancelled or Postponed [I2B, I8, I9, 22, 27, 45, 63, 70, 93, 94, 99]	Only done when Scrum Master needs updates [I2B]. Frequent meetings with externals [I9]. Working closely together makes meetings superfluous [I8, 27, 94]. It is difficult for the team members to meet daily [45, 70, 99]. Some times a team member could not meet on time [63, 93]. Progress is not sufficient to warrant meeting [70]. The team is very small [45].
Sprint Retrospectives should be held at the end of every Sprint.	Retrospectives Cancelled or Postponed [I1, I2B, I4, I5, I6, I8, I9, 22, 27, 35, 45, 70, 95]	People, especially Product Owner, cannot attend [I4, 27]. Problems are solved as they appear [I8, I9, 27]. Not enough to talk about every Sprint [I1, I6, 45]. Time pressure dictated from outside the team (e.g. management) [70]. Team is too large [45]. Lack of feedback [45]. Issues handled through other meetings [45].
Sprint Reviews should be held at the end of every Sprint.	Sprint Review Cancelled or Postponed [27, 83]	A stong client relationship with frequent contact makes Sprint Reviews unnecessary [27]. A "Weekly Meeting" makes Sprint Reviews unnecessary [83].
Sprint Planning should be held at the beginning of every Sprint.	Sprint Planning Cancelled or Postponed [I2B, I4, I9, 83, 95]	No estimations, so planning is done on demand [83]. Sprints are replaced with Flow [I2B, I9]. People, especially Product Owner, cannot attend [I4].

Table 4.3: Cancelling or postponing Scrum Events.

All Scrum Events have one type of ScrumBut in common, namely being postponed or cancelled; these ScrumButs are presented in Table 4.3. Daily Scrums are

more often postponed than cancelled completely. In several cases it is simply impractical for all to meet daily [I9, 45, 70, 99]; for example one interviewee stated *"We constantly visit our clients, and we would rather prioritise having three days instead of five"* [I9, translated from Danish], thereby opening up days for visiting clients without disturbing the meetings.

In some cases team members have problems being there on time [63, 93], and instead of having the meeting without full attendance, it is postponed or arranged in a way, so all members can attend. In a few cases the Daily Scrum is used as a way to share task based progress with the team or management than a way for assessing the progress towards the Sprint Goal, and the meeting is instead only held when there is sufficient progress to justify a meeting [I2B, 70]. In some instances, the meeting is not conducted at all, because the teams perceive their members to work so closely together that the meeting is superfluous [I8, 27, 94]. It is especially noteworthy that the reasons given for cancelling or postponing both Daily Scrums and Sprint Retrospectives are different in nature.

Sprint Retrospectives are cancelled or postponed for some of the same reasons as Daily Scrums: Team members have trouble attending [I4, 27] or there is not enough to talk about after only one Sprint [I1, I6, 45]. In other cases the problems are simply resolved as they appear, removing the need for a formal meeting [I8, I9, 27]. In one case it is mentioned that the Sprint Retrospective is incorporated in other meetings because *"the Scrum model was working effectively and any issues or changes could be adequately handled through the other meetings"* [45, p. 113].

Only two cases of *Sprint Review Cancelled or Postponed* were identified, and the event is either replaced by another meeting [83] or considered unnecessary because of a strong client relationship [27]. As seen in Table 4.4, several cases of having *Sprint Review Without Stakeholders* have been identified, so while the meeting is seen as important enough to conduct, it can be difficult to include stakeholders. Four different reasons are given, but all indicate that participating in the meeting is not perceived to be worth the time of external stakeholders, and instead the meeting is done with only the Scrum Team–sometimes with a team member acting as stand-in for the client (e.g. [I4]).

Most events are time-boxed to take a maximum of 4 hours, with the exception of the Daily Scrum, which is time-boxed to take a maximum of only 15 minutes; the bottom four ScrumButs in Table 4.4 show examples of those time-boxes being exceeded. The reason for exceeding the time-box of the Daily Scrum is mostly that discussions are allowed during the meeting, instead of waiting until after the meeting [I2B, I9, 22, 45, 63, 96].

One case explains why the discussions are tolerated during the meeting: The discussions are important, but the team does not have a lot of time, so by allowing the discussions to take place during the meeting rather than afterwards, the maximum duration can be used as an incentive to stay on track–even if it means it is some times exceeded [I9]. The other events are, except for a few cases, done within the set time-box. In fact, many teams finish the meetings much faster than the prescribed maximum duration, and many report having planning meetings lasting less than two hours, compared to the limit of eight hours (e.g. [22, 70]).

Events (part 2)		
Scrum [80]	**ScrumBut**	**Reasoning**
"The heart of Scrum is a Sprint."	Replacing Sprints with Flow [I2B, I9, 27] Not Using Sprints [25]	Project lengths are very short [I2B]. It is necessary to be very responsive to clients' needs [I9, 27].
"The Sprint Retrospective occurs after the Sprint Review."	Merging Events [I6, 22, 45, 70]	Separating the meetings feels unnecessary [I6]. Separate meetings take too much time [70]. No real Sprint Retrospective planned, so its agenda is added to the Sprint Review [70].
The whole Development Team should participate in one meeting.	Multiple Daily Scrums [45]	The Development Team is too big to include everybody in one meeting [45].
"Attendees include the Scrum Team and key stakeholders."	Sprint Review Without Stakeholders [I4, 22, 45, 63, 70]	Stakeholders do not have time to participate [I4]. Internal reviews by other teams [45]. Product Owner or other acts as stand-in [I4, 45]. Meeting too long and detailed [63].
Daily Scrum is an event for the Development Team.	Some Developers Not Attending Daily Scrum [63]	A combination of delay and unfocused discussion [63].
"The Daily Scrum is a 15-minute time-boxed event."	Extending Duration of Daily Scrum [I2B, I9, 22, 45, 50, 63, 96]	Detailed discussions allowed during meeting [I2B, I9, 22, 45, 63, 96]. Daily Scrum not held daily [22].
"Once a Sprint begins, its duration is fixed and cannot be shortened or lengthened."	Extending Duration of Sprint During the Sprint [25, 99]	Use of new technologies without prior training [99].
"Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint."	Extending Duration of Sprint Planning [93]	Estimating tasks is complicated by inability to agree on required complexity [93].
"This is a four-hour time-boxed meeting for one-month Sprints."	Extending Duration of Sprint Review [22]	

Table 4.4: Remaining ScrumButs related to the Scrum Events.

In most cases the Sprint duration is static and time-boxed to between two and four weeks. In those cases, a common way to deal with unexpected high priority tasks is to allocate a buffer – a duration of time with no preallocated tasks (e.g. [35]). Two cases mention another approach: The duration of the Sprint can be extended after the Sprint started [25, 99]. Vilain and Martins [99] report using a new technology, and instead of allocating a set amount of time for getting acquainted with the technology, they will extend the Sprint if it causes delays. They argue that this approach allows actual product development to start sooner. In addition it serves as a way to keep the development team focused on the tasks at hand: *"The idea behind that decision was to prevent a newly created sprint from causing team members to lose their focus on the issues related to learning the new technology, and also to save the team from unnecessary frustration resulting from a significant number of sprints not being completed"* [99, p. 599].

4.1.3 Scrum Artefacts

The Scrum Artefacts are *''specifically designed to maximize transparency of key information"*, and they serve as an important role in providing *"opportunities for inspection and adaptation"* [80, p. 12]. Table 4.5 shows the ScrumButs where management is interfering with the Scrum Team or where the Product Backlog is influenced. Table 4.6 shows the remaining ScrumButs related to the Scrum Artefacts.

The first thing worth noticing in Table 4.5 is that several ScrumButs have the reasoning that the team is interrupted by requests or requirements from management that cannot be ignored. As with *Having Titles on Development Team* (see Table 4.2) these ScrumButs have reasons originating from outside the team itself: In some cases the teams are required to do bug fixes and ad-hoc tasks even if they interfere with the original Sprint Goal [I2B, 89]. In other words, management obstructs the Development Team's plan to reach the Sprint Goal. In other cases, the Development Team does not even make the Sprint Backlog themselves [I3, 22]. In the one case backed with a reason an old hierarchical approach already had a set practice in place [22]. Lastly there are examples where the Development Team does not estimate Product Backlog items, but instead it is done by people deemed to have the necessary domain knowledge [I2A, 99]. One interviewee stated that *"it's not really necessary for the whole team to do planning poker; it's sufficient to include the ones with the appropriate domain knowledge"* and when asked why, replied *"it's my perception that it's deemed to take too much time"* [I2A, translated from Danish].

The remaining ScrumButs in Table 4.5 affect the transparency of the Product Backlog negatively. It is important for the Product Backlog to be transparent, as it shows upcoming work needed on the product. One ScrumBut has simply been dubbed *Product Backlog Not Transparent*, which covers cases that do not warrant their own category, but still significantly reduce the transparency of the Product Backlog to a point where it may be an impediment to the work-flow of the team; three cases fit that description [8, 70, 95]. In one of these cases, the lack of transparency comes from the fact that the Product Backlog is split into several parts, each maintained by one Product Owner, and as a direct consequence of *Having a Product Owner Committee* [8]. In two interviews, it was stated that irrelevant

Artefacts (part 1)

Scrum [80]	**ScrumBut**	**Reasoning**
"Only the Development Team can change its Sprint Backlog during a Sprint."	Management Changing Sprint Backlog During Sprint [I2B, 89]	Management requires ad-hoc tasks and bug fixes to take priority [I2B, 89].
"The Development Team is responsible for all estimates."	Development Team Not Making Estimations [I2A, 25, 99]	It is sufficient to include those with domain knowledge [I2A, 99]. Estimated product cost needed in advance [25]. It takes too much time [I2A].
The Sprint Backlog should be selected by the Development Team	Management Making Sprint Backlog [I3, 22]	Remains from old hierarchical development process [22].
The Product Backog should be "visible, transparent, and clear to all."	Product Backlog Not Transparent [8, 70, 95]	Several Product Owners each maintain their own (partial) backlog [8].
"The Product Backlog is an ordered list of everything that might be needed in the product."	Irrelevant Product Backlog Items Not Removed [I1, I6]	The people responsible do not know if a given item might be needed [I1]. It takes too much time to constantly refine, and the most important tasks are in the top [I6].
"The Product Backlog is an ordered list."	Product Backlog Not Ordered [25, 70]	It takes to much time and effort [70, 25]. No Product Owner [25]. Product Owner does not possess the required knowledge [25].
The Product Backlog is a central artefact.	No Product Backlog [25]	
"Product Backlog items have the attributes of a description, order, estimate and value."	No Estimations [25]	

Table 4.5: ScrumButs related to interfering management and the Product Backlog.

Product Backlog items are not regularly removed [I1, I6]. In one case, where it was argued that removing irrelevant items would take too much time, it was stated that the Product Backlog was very long, but as the important tasks are always on top, it is sufficient to remove irrelevant items a few times a year [I6]. In the other case the people responsible–there is no Product Owner in that particular case–do not have sufficient domain knowledge to know if a given item is needed or not [I1]. The ScrumButs related to the Product Backlog also cause other ScrumButs, e.g. Sprint Planning being done on demand due to a lack of estimations [83].

Artefacts (part 2)		
Scrum [80]	**ScrumBut**	**Reasoning**
Product Backlog Items selected for the Sprint Backlog should be doable within a Sprint.	Stories Longer than Sprint Duration [I5, 94, 96]	Student developers only work part time [I5]. Some items require a lot of research / experimentation [96].
"Any one product or system should have a definition of "Done" that is a standard for any work done on it."	Not Defining Done [I5, I8, 22, 70, 94]	Done criteria are defined on a backlog item level [I5, 70]. Development has not started yet, but is planned to be included later [I8].
"Development Teams deliver an Increment of product functionality every Sprint."	No Increment [I1, 25, 94]	Separate test site resulted in test-ready features rather than potentially releasable features after a sprint [94]. Sprints alternate between new development and testing, so a new Increment is produced every other Sprint [25]. New features are released immediately [I1].

Table 4.6: Remaining ScrumButs related to the Scrum Artefaccts.

An interesting observation is, that there are several ScrumButs related to the Product Backlog, but as shown in Table 4.6, and aside from interference by management, only two ScumButs influence the Sprint Backlog: *Not Defining Done* and *Stories Longer than Sprint Duration*. *Not Defining Done* was identified in several interviews and papers, and besides one case where the team had not started developing the product [I8], the reason given was that describing the *Done* criteria on a backlog item level is sufficient [I5, 70]. *Stories Longer than Sprint Duration* was identified in fewer cases, and both reasons are centred around team and product rather than process and management: either to account for having part time team members [I5] or to allow for experimentation which can be difficult to estimate [96].

Having *No Increment* (after every Sprint) can be associated to either a need for fast delivery [I1] or a separation of development and testing [25, 94], but in some interviews the interviewee seemed to understand *Increment* as a synonym for *release* (e.g. [I6]). In the example of Interview 6, the interviewee then went on to explain that they do indeed end each Sprint with something potentially releasable: *"I think all teams aim for–after each iteration–having something that could be released"* [I6].

4.2 Organisational Culture and ScrumButs

During analysis a pattern emerged in the data, and it became clear that an association can be made between the emergence of ScrumButs and organisational culture.

Relating a reason for using a ScrumBut to a culture does not prevent the company or team in question from having qualities from the other cultures.

4.2.1 Competing Values Model

Culture can be understood as *"a symbolic system consisting of learned, shared, patterned sets of meanings guiding the actions of cultural members"* [49, p. 511]. The Competing Values Model (CVM) can be used to distinguish four different cultures based on the dimensions *Internal Focus vs. External Focus* and *Stability vs. Change*, namely *Developmental, Consensual, Rational and Hierarchical* [49]. Figure 4.1 shows the dimensions and the corresponding cultures.

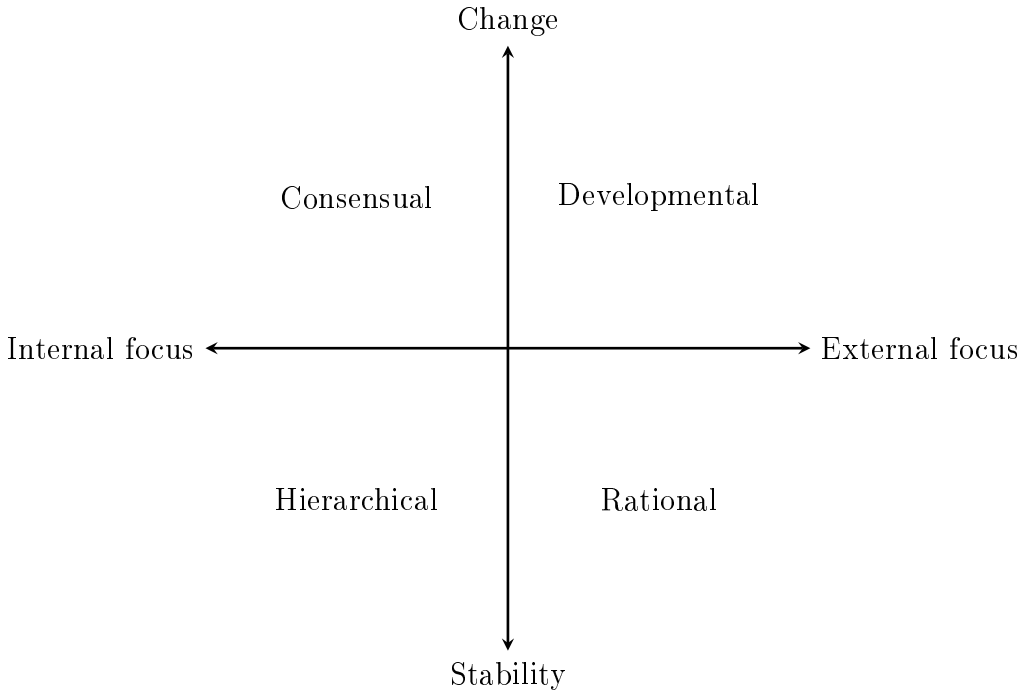


Figure 4.1: The dimensions of the Competing Values Model [49].

Being placed in one culture does not prevent a you from having values from the other cultures. Ngwenyama and Nielsen [69] present an overview of key values, based on Quinn and McGrath [72]. The values, shown in Table 4.7, will be used to associate a ScrumBut's reasonings with a culture, illustrating the association between organisational culture and the emergence of ScrumButs.

4.2.2 Categorising the Data

Each ScrumBut is associated with a culture based on evidence found in the paper or interview. For each ScrumBut, if a reason is given, the reason is analysed in relation to the values in Table 4.7. If the reason itself is not enough to make an association, the search is expanded to the surrounding area in the interview or paper, to look for evidence explaining the root cause of the reason. In some cases there is not sufficient evidence to make an association, in which case it is not placed. To illustrate how

Aspect	Developmental	Consensual	Rational	Hierarchical
Organisational orientation	Flexibility, adaptability, and readiness	Cohesion and morale	Productivity and efficiency	Stability and control
Organisational objectives	Growth and development	Group maintenance	Pursuit of objectives	Execution of regulations
Organisational structure	Complex tasks; Collaborative work groups	Complex tasks; Collaborative work groups	Complex tasks; Responsibilities based on expertise	Routine tasks and technology; Formal rules and policies
Base of Power	Values	Ability to cultivate relationships	Competence	Knowledge of organisational rules and procedures
Decision making	Organic, intuitive	Participatory, deliberative	Goal-centred, systematic and analytical	Top-down pronouncements
Leadership style	Idealistic, risk oriented, empowering	Team builder, concerned, supportive	Rational achiever, goal oriented	Dominance, conservative, cautious
Compliance	Commitment to values	Commitment to process	Contractual agreement	Monitoring and control
Evaluation of members	Intensity of effort	Quality of relationships	Level of productivity	Adherence to rules
Orientation to change	Change is embraced as part of growth	Open to change	Open to goal driven change	Resistant (orientated to maintaining the status quo)

Table 4.7: Overview of competing values in organisational culture [69, 72].

the associations are made, an example is given for ScrumBut reasonings from each of the four cultures.

4.2.2.1 Developmental

The following is a reason for *Extending Duration of Sprint During the Sprint*:

The sprints can have their time length increased by 25% in case of error in the estimate due to the lack of experience with new technologies.

[99, p. 599]

The reasoning indicates an orientation towards *flexibility and adaptability*, agreeing with the developmental culture.

4.2.2.2 Consensual

In this example the ScrumBut *Daily Scrums Cancelled or Postponed* is reasoned with working closely together:

At some point, the problem presented at the Daily Scrum was "there is no point in doing a Daily Scrum" and, after reflecting briefly on that, the team decided that if they kept the discipline of changing pairs often and asking for help whenever they needed it, the meeting could be abolished.

[27, p. 105]

The decision to remove the event is made by the entire team, fitting with the *participatory* decision making, which lies in the consensual culture as shown in Table 4.7.

4.2.2.3 Rational

One of the many reasons for having *No Scrum Master* is given below:

In all companies without a dedicated Scrum Master, the costs seem to play a major role. They avoid reducing the overall capacity by assigning a developer as full-time Scrum Master.

[22, p. 46]

The wish to *avoid reducing the overall capacity* indicates an orientation towards *productivity* which agrees with values of the rational culture, as seen in Table 4.7.

4.2.2.4 Hierarchical

The last example is from an interview, and the ScrumBut is *Having Titles on Development Team*. The explanation for that is:

[To make personnel management easier] we have a role called team lead. Some of the team leads are Scrum Masters while others have another role we use, namely Business Architect.

[I1] (translated from Danish)

This reasoning indicates an orientation towards *stability and control*, as well as *top-down* decision making: The team lead role is put on the team by management, because it makes it easier to manage the personnel. As seen in Table 4.7, that places the reasoning in the hierarchical culture.

4.2.3 ScrumButs Associated with Cultures

In Tables 4.8-4.10 each ScrumBut is assigned to appropriate cultures based on its underlying reasons, revealing which ScrumButs appear in which cultures.

4.2.3.1 Scrum Roles

Table 4.8 shows the ScrumButs related to roles associated with their respective cultures. *Scrum Master Not Protecting the Team* and *Team Not Self-organising* are not included, as no reasonings were given.

	Roles			
ScrumBut	**Dev.**	**Con.**	**Rat.**	**Hie.**
Having a Product Owner Committee	[I8]		[35, 93]	[I1, 8]
Product Owner Not Accountable for Ordering of Product Backlog Items				[I1]
No Product Owner			[I6]	[I1, I2A, I2B, 22]
No Scrum Master		[I8, 27]	[22, 50]	
Exceeding Team Size Boundaries	[I8]			[I1, 22]
Having Titles on Development Team			[I4]	[I1, I6]
Development Team Not Cross-functional				[I4]

Table 4.8: Cultures of ScrumButs related to the Scrum Roles.

Not having a Product Owner at all only happens in rational and hierarchical cultures; generally because they use traditional product management instead. Similarly *Having Titles on Development Team* also only happens in rational and hierarchical cultures, as a way of keeping track of personnel and knowing who to go to when expertise and experience is needed.

The reasons for having *No Scrum Master* are associated with consensual and rational cultures, which is interesting as the cultures are diametrically opposed (see Figure 4.1). Looking at the underlying reasons, it does make sense to place them in those two cultures: The cases placed in the consensual culture sees the Scrum Master as a hindrance to the team's engagement, while the cases placed in the rational culture sees resources spent on a Scrum Master as better invested in a Developer. *No Scrum Master* is also the only ScrumBut related to the Scrum Roles that does not have any reasonings associated to the hierarchical culture. As it will become apparent in the remainder of this chapter, most reasonings can be associated to these two cultures, with only few associated to the developmental and consensual cultures. Examples show that the same sources use both developmental and consensual arguments ([I8]) or rational and hierarchical arguments ([I4, I6, 22]).

4.2.3.2 Scrum Events

The ScrumButs related to the Scrum Events are shown in Table 4.9, and more ScrumButs have reasons grounded in developmental and consensual cultures com-

pared to the ScrumButs in Table 4.8. In spite of that it is clear that the reasoning behind most of the identified ScrumButs can be associated with the values of the rational and hierarchical cultures. That notion is especially true if *Daily Scrums Cancelled or Postponed* and *Retrospectives Cancelled or Postponed* are ignored when looking at the full picture, which is reasonable as they appear to happen regardless of cultural alignment, and as such cannot be regarded as a trait specific to a given culture.

Events				
ScrumBut	**Dev.**	**Con.**	**Rat.**	**Hie.**
Daily Scrums Cancelled or Postponed	[I8, 63]	[27, 94]	[I9, 93, 99]	[I2B, 45, 70]
Retrospectives Cancelled or Postponed	[I8]	[I6, I9, 27]	[45]	[I4, 45, 70]
Sprint Review Cancelled or Postponed	[27]			
Sprint Planning Cancelled or Postponed			[I2B, I9]	[I4]
Replacing Sprints with Flow		[27]	[I9]	[I2B]
Merging Events	[I6]		[70]	
Multiple Daily Scrums				[45]
Sprint Review Without Stakeholders			[63]	[I4, 45]
Some Developers Not Attending Daily Scrum			[63]	
Extending Duration of Daily Scrum		[96]	[I2B, I9, 63]	[45]
Extending Duration of Sprint During Sprint	[99]			
Extending Duration of Sprint Planning			[93]	

Table 4.9: Cultures of ScrumButs related to Events.

The fact that Daily Scrums and Retrospectives are cancelled or postponed across cultures, shows that sometimes reasonings based on different values can result in similar ScrumButs. In both Table 4.8 and Table 4.9 several ScrumButs have only one example of a reasoning associated with the culture. It does not provide much insight about the particular ScrumBut, but in combination with other ScrumButs it helps

paint a picture of the culture as a whole. ScrumButs with reasonings placed in the rational culture match the culture's organisational orientation towards productivity and efficiency, as most of those ScrumButs can be attributed to a wish for an increase in productivity. For example, one interviewee explained why they did not use Sprint Planning: *"The client tells us [about new requirements] with short notice, and [Sprint Planning] does not work well for that; [the plan] collapses as soon as the Sprint starts. We realised that quite early on, so we replaced it with an approach closer to ScrumBan."* [I9, translated from Danish]. The example shows rational values, both in terms of the *pursuit of objectives* and *openness to goal driven change* (see Table 4.7). In another example, the Sprint Planning time-box is extended: *"We wrestle with some of the more challenging features that impact the core data model. We explore different design ideas in an attempt to maintain a balance between getting something done for this release versus creating a more lasting solution."* [93, p. 156]. Again, an example of rational culture, with *goal-centred, systematic and analytical* decision making.

4.2.3.3 Scrum Artefacts

The ScrumButs related to the Scrum Artefacts are shown in Table 4.10, and as before, most are still placed in the rational and hierarchical cultures.

	Artefacts			
ScrumBut	**Dev.**	**Con.**	**Rat.**	**Hie.**
Management Changing Sprint Backlog During Sprint				[I2B, 89]
Development Team Not Making Estimations			[I2A, 99]	[25]
Management Making Sprint Backlog				[22]
Product Backlog Not Transparent				[8]
Irrelevant Procuct Backlog Items Not Removed			[I6]	[I1]
Procuct Backlog Not Ordered			[70]	
Stories Longer than Sprint Duration	[96]	[I5]		
Not Defining Done	[I5, I8]		[70]	
No Increment			[I1]	[94]

Table 4.10: Cultures of ScrumButs related to Artefacts.

One thing to note is the fact that all ScrumButs related to management doing the tasks of the Development Team are placed in cultures oriented towards stability;

instead of using the Scrum Artefacts as tools for providing transparency they are used as project management tools. That is especially apparent in *Product Backlog Not Transparent* where several Product Owners each maintain a partial Product Backlog (see Table 4.5), and only collect the higher priority backlog items before a Sprint Planning meeting [8].

Only few ScrumButs related to the Scrum Artefacts have reasons placed within the cultures oriented towards change, and the ones that do, *Stories Longer than Sprint Duration* and *Not Defining Done*, make sense in a setting where decision making is organic and intuitive and the organisation is oriented towards flexibility. The one case in the consensual culture is placed based on the supportive leadership style, allowing the team members working part time to participate even though limited work time some times make tasks exceed the Sprint duration. The lack of reasonings placed in developmental and consensual cultures seems to indicated that when changes are made to the Scrum Artefacts they are mostly made to adapt them to an environment that values stability.

5 Discussion

In this chapter we will discuss our findings in terms of contribution to the field of research in agile methodologies, limitations of the study, and finally we will suggest ideas for future research.

5.1 Contribution

The contribution of this study is twofold: firstly it extends current knowledge of why ScrumButs emerge in practice, and secondly i shows the existence of a relationship between organisational culture and the reasoning behind ScrumButs.

5.1.1 Scrum Anti-patterns

In June, 2016 Eloranta et al. [25] published a paper called "Exploring ScrumBut – An empirical study of Scrum anti-patterns" in the highly ranked software engineering journal "Information and Software Technology". As the name suggests, the paper is similar to this study, with a focus on the identification of ScrumButs. Others have studied the different use of Scrum practices (e.g. [22]), but Eloranta et al. [25] provide the first comprehensive attempt at mapping the different ScrumButs found in practice. Instead of grouping the ScrumButs by culture, the paper describes anti-patterns including consequences and suggestions to move away from each anti-pattern. The study is based on 18 interviews in 11 different Finnish companies, and a total of 14 anti-patterns were identified. It is worth noting that explicit examples of ScrumButs from Eloranta et al. [25] were included in the findings of this project, but the anti-patterns themselves were not.

In most of the cases presented by Eloranta et al. [25] we have supporting evidence of the emergence of a given ScrumBut, often including reasoning behind it. One example is their anti-pattern *Business as usual (No Sprint Retrospective)*, which corresponds to our *Retrospective Cancelled or Postponed*. Eloranta et al. [25] present a context in which this happens, namely in situations where agile development and Scrum is not sufficiently understood, or when there is no Scrum Master or the Scrum Master role is not used correctly. Our study does not link *Retrospective Cancelled or Postponed* to the same context, but instead provides additional context though the associated rationales. For example we identified the ScrumBut in cases where central people had trouble attending the event [I4, 27] or external time pressure dictated

it [70]. Mapping these reasonings to organisational cultures provides additional insight into the context in which a given ScrumBut may be present, which helps in understanding why ScrumButs emerge in software development. For some of the anti-patterns presented by Eloranta et al. [25] the Scrum Guide offers no specific recommendation. As this study focuses exclusively on ScrumButs that contradict the Scrum Guide, we have not recorded anything to support those anti-patterns.

In addition to interviews from a different social culture, our study includes a literature study of a wide range of papers that mention adaptations of Scrum along with reasons for the adaptations. This study also presents a number of more rare ScrumButs (e.g., *No Estimations*) that have no associated reasoning, but that help in painting a more complete picture of the ScrumButs that may be encountered. With our findings we confirm most of the findings of Eloranta et al. [25], as well as extending them with additional context and ScrumButs. Some findings from related studies are not confirmed, but no contradictions were found.

A similar study, i.e., a study focussing on tailoring of agile software development, was made by Conboy and Fitzgerald [19] who studied tailoring of XP instead of Scrum, but with a more general topic of finding out how to make agile methodologies more amenable to tailoring. The study is made up of two phases; first 20 researchers were interviewed to lay a foundation, or framework, for the areas to study, and later 16 practitioners were interviewed based on the framework. The findings are a number of constructs, identified in the first phase of the study, along with a reflection of the construct in practice and a recommendation for researchers to improve the situation. For example, relating to the construct*explicit statement of method boundaries*, it was found that most teams had problems identifying the boundaries of XP (i.e., where it should and should not be applied) in spite of considerable effort. Because of this the recommendation for researchers was to study *"levels of project success across different potentially problematic software development environments"* [19, p. 2:24]. Our findings extend this and other research in tailoring of agile software development methodologies by collecting and presenting the reasoning behind adaptations of Scrum, thereby providing a better basis for understanding under which conditions tailoring is done.

5.1.2 Organisational Culture and Agile Methodology

In the previous section concrete changes to Scrum were discussed, related to the first part of the findings, presented inSection 4.1. In this section the focus is on the findings presented in Section 4.2, where associations between organisational culture and ScrumButs are made through the competing values model of organisational culture. In a recent study, Persson et al. [71] hint that the agility of an organisation as a whole is linked directly to the agility of individual projects. The connection between organisational culture and agile methodology use was hypothesised by Iivari and Iivari [49] in 2011. The hypotheses were made based on an earlier study of traditional software development methodologies ([48]) coupled with their understanding of agile development. The focus in their paper is not on Scrum, but agile methodologies in general. As the focus in this report is on Scrum, we will comment on the hypotheses from the point of view of Scrum only. Two of the hypotheses are

presented in Table 5.1; note that *method* is used in the sense of *methodology*, i.e., "a predefined combination of methods collected under a name", as described in Chapter 1. The hypotheses are aimed towards quantitative studies, but we are still able to provide some insight.

In Table 5.1 hypotheses H63b suggest that teams with a hierarchical culture orientation perceive traditional methodologies to better support their values than agile methodologies. H67b suggests that teams with a developmental culture perceive agile methodologies to better support their values than traditional methodologies.

H63b The hierarchical culture orientation has a negative impact on IT managers' and software developers' beliefs in agile method support for the values of the hierarchical culture when compared with traditional methods.
H67b The developmental culture orientation has a positive impact on IT developers' and software developers' beliefs in agile method support for the values of the developmental culture when compared with traditional methods.

Table 5.1: Hypotheses about the hierarchical culture orientation [49].

The fact that a big part of the identified ScrumButs are placed in the hierarchical culture shows there might be a mismatch between the hierarchical values and those perceived to be supported by Scrum. That mismatch goes well in hand with the fact that the hierarchical culture is resistant to change. In Table 2.1 Conboy's [17] proposed taxonomy of ISD agaility prescribes three requirements that must be fulfilled for a practice to be considered agile. Looking at the different ScrumButs placed in the hierarchical culture, there is a tendency to make changes that causes the Scrum practice to lose some of its agility. This goes well in hand with the hierarchical culture's values on decision making, *top-down pronouncements*, and *dominating, conservative* leadership style shown in Table 4.7. An example emphasising those values is found in Table 4.10 where the ScrumBut *Management Changing Sprint Backlog During Sprint* moves Scrum towards a more traditional approach, where the team is not self-organising. The ScrumBut can be formulated as follows:

> *(We use Scrum, but) (ad-hoc tasks and unplanned bug fixes take priority,) (so management may make changes to the Sprint Backlog of an ongoing Sprint.)*
>
> [I2B, 89]

The workaround, *management may make changes to the Sprint Backlog of an ongoing Sprint* can be held against the taxonomy, and while it goes against the prescriptions of the Scrum Guide, the first requirement is satisfied as there is both a *creation of change* and a *reaction to change*. The second requirement is, however, not satisfied as *perceived simplicity* is negatively impacted, with *perceived economy* and *perceived quality* are both in a more grey area. This example shows a change of a Scrum component that negatively impacts agility, and while it neither proves nor

disproves the first hypothesis in Table 5.1, it does agree with it. On the other hand there are not many ScrumButs that can be linked to the developmental culture, which is consistent with the similarities between the agile values and developmental culture. For example, agile values *responding to change over following a plan*, where in developmental culture, *change is embraced as part of growth.* An example, however, of a ScrumBut linked to the developmental culture is the merging of events, found in Table 4.9. In this specific example, the ScrumBut can be formulated as follows:

(We use Scrum, but) (we find it unnecessary to conduct two individual meetings,) (so our Sprint Planning and Sprint Review have been merged.)

[I6]

Looking again at the taxonomy of ISD agility we see that this modification does not take away the agility of the original method component; the merged meeting still contributes to at least *reaction to change*, satisfying the first requirement. The meetings already contribute to *perceived economy*, and by making the change *perceived simplicity* arguably improves, while retaining *perceived quality.* The third requirement is satisfied before and after the merge as well, as the events are planned to take place at the same time every Sprint.

By focussing on the most common agile methodology, Scrum, our findings extend existing research by contributing to the understanding of the connection between use of agile methodologies and organisational culture. The connection between organisational culture and ScrumBut can nuance the debate of whether ScrumButs are good or bad. The nuance lies in moving the attribution of a ScrumBut's nature, from an inherent nature, to a nature dependent on its values and the values of its environment. An example of this can be demonstrated by the ScrumBut of *Development Team Not Making Estimations*: Instead of assuming its inherent nature as bad, looking at its reasoning and the values behind it, in association with the values of organisational cultures, shows its nature as more nuanced. It is shown in Table 4.10 that one of the reasonings behind this ScrumBut, *estimated product cost needed in advance* [25], is associated to the value-set of the hierarchical culture. Given this association it can be argued that this ScrumBut is benign, or even beneficial, in a company with strong hierarchical values, as the change is made to better support the organisation's values. Thus, the nature of a ScrumBut might not be as simple as the binary judgment of being inherently good or bad.

5.2 Limitations

The relationship between organisational culture and method use, as discussed by Iivari and Iivari [49], is complex and views on the relationship range from it being causal (one causes changes in the other) to emergent (the relationship is constantly evolving). We believe that the relationship is emergent, but this study only finds how organisational culture affects the emergence of ScrumButs. The limitation does not invalidate the findings, but further research is necessary to learn about how the emergence of ScrumButs affect organisational culture. Organisational culture was

found to influence the emergence of ScrumButs, but it may not be the only source of influence. As no other sources of influence are accounted for in the study, the findings may have been affected. Other sources of influence could be project budget, experience of developers, etc. These factors may influence the organisational culture directly as well, complicating the situation further.

The interviewees selected for the study were all situated within a relatively small area, as a result of which their social cultures may have been similar enough to influence the results. The limitation is partially neutralised by also studying literature with interviews from many different social cultures. The data obtained from the literature, however, has the limitation that the explanations are sometimes not very thorough. Some studies have no explanations at all, and as they provide little value for the study, general Scrum case studies were filtered in the search criteria, despite these likely containing various examples of ScrumButs. This decision limited our data, but as we sampled some of these, we found that only very few were aware of the ScrumButs and even fewer presented a reasoning for them (e.g. [68, 90]). Despite filtering out these papers, the study reached a saturation point, minimising the impact these papers could have presented.

The final limitation pertains to the Grounded Theory process itself: the act of coding the data, although attempted to be done in an objective manner, becomes subjective when decisions have to be made as to which parts of the data are important. To minimise this problem, and focus the analysis around similar topics, the coding process was frequently discussed during the analysis process.

5.3 Future Research

In this study we researched the practices and reasonings behind ScrumButs. Grounded in the data we found a connection between organisational culture and the emergence of ScrumButs. We have two suggestions for possible future research:

First, we suggest conducting a study to quantify the connection between organisational culture and ScrumBut reasoning. Such a study could improve the understanding of how organisational culture affects the adaptation of Scrum and provide new insights for the discussion of whether ScrumButs are inherently bad or not.

Secondly, we propose an expansion of this study, but with a focus on the evolution of Scrum and ScrumButs over time in the same company, shedding light on how organisational culture influences the emergence of ScrumButs, and on how ScrumButs influence organisational culture. In addition, it would be interesting to hold the ScrumButs found in such a study up to the taxonomy of ISD agility (Table 2.1) to determine if organisational culture influences the agility of ScrumButs, as discussed in Section 5.1.2.

Conclusion 6

The purpose of this study was to explore adaptations of Scrum in the form of ScrumButs, i.e., changes to the existing practices of the framework. The empirical research on the subject of ScrumButs is limited, and the question of the underlying reasoning is unexplored in research literature. In the study, the following two questions were pursued:

I. *What ScrumButs are prevalent in professional software development?*
II. *How are ScrumButs reasoned in professional software development?*

We collected 17 papers through a thorough literature search, using the four largest software engineering digital libraries of peer-reviewed papers, and by interviewing 9 representatives from 7 local software development companies using semi-structured interviews with a laddering approach. This data was then analysed through several iterations of open and axial coding until a selective code was identified and built upon.

ScrumButs were identified in all Scrum practices and their reasons ranged from precipitate, e.g. not ordering the Product Backlog because it takes too much time and effort [25, 70], to highly deliberate arguments, e.g. extending a Sprint after it starts to ensure there is enough time to experiment with new technologies [99]. We showed that our findings are consistent with the Scrum anti-patterns presented by Eloranta et al. [25], and that we contribute to research in agile methodology tailoring by extending the knowledge of which ScrumButs emerge in professional software development. For most ScrumButs one or more reasons were given to explain why the practice was changed. Individually, the reasons give little insight other than for the instance it explains, but after several iterations of analysis a pattern emerged. By holding each reason up against the values in the Competing Values Model, it was found that most reasons can be linked to values of an organisational culture. The connection between organisational culture and ScrumBut reasonings can help nuance the discussion of whether ScrumButs should be considered good or bad, by focusing on the underlying values of the reasoning and relating them to the organisation's culture.

We discussed the findings, and showed that the study connects ScrumBut reasonings to organisational culture in a way that fits the hypotheses proposed by Iivari and Iivari [49]. We also presented an example of how a ScrumBut linked to the hierarchical culture lost its agility when held up against the taxonomy of ISD agility [17], while another ScrumBut linked to the developmental culture did not. More research is needed to show if that trend can be found in other ScrumButs.

Bibliography

[1] ACM (2015). PEER-REVIEW SYSTEM. `http://www.acm.org/publications/submissions`. [Online; Accessed 2016-05-02].

[2] Adolph, S., W. Hall, and P. Kruchten (2011). Using Grounded Theory to Study the Experience of Software Development. *Empirical Software Engineering 16*(4), 487–513.

[3] Ågerfalk, P. J. and B. Fitzgerald (2006). Exploring the Concept of Method Rationale: A Conceptual Tool for Method Tailoring. In *Advanced Topics in Database Research*, Volume 5, pp. 63–78. Idea Group Publishing.

[4] Aydin, M. N., F. Harmsen, K. van Slooten, and R. A. Stagwee (2005). On the Adaptation of an Agile Information Systems Development Method. *Journal of Database Management (JDM) 16*(4), 25–40.

[5] Baskerville, R. and J. Stage (2001). Accommodating Emergent Work Practices: Ethnographic Choice of Method Fragments. In *Realigning Research and Practice in Information Systems Development: The Social and Organizational Perspective*, pp. 11–28. Springer.

[6] Beck, K., M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas (2001). Manifesto for Agile Software Development. `http://www.agilemanifesto.org/`. [Online; Accessed 2016-04-02].

[7] Beedle, M., M. Devos, Y. Sharon, and K. Schwaber. Scrum: An extension pattern language for hyperproductive software development.

[8] Block, M. (2011). Evolving to Agile: A story of agile adoption at a small SaaS company. In *Agile Conference (AGILE'11)*, pp. 234–239. IEEE.

[9] Boehm, B. W. and R. Turner (2003). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley Longman.

[10] Bowers, J., J. May, E. Melander, M. Baarman, and A. Ayoob (2002). Tailoring xp for large system mission critical software development. In *Conference on Extreme Programming and Agile Methods*, pp. 100–111. Springer.

[11] Boychuk Duchscher, J. E. and D. Morgan (2004). Grounded theory: reflections on the emergence vs. forcing debate. *Journal of advanced nursing 48*(6), 605–612.

[12] Brinkkemper, S. (1996). Method Engineering: Engineering of Information Systems Development Methods and Tools. *Information and Software Technology 38*(4), 275–280.

[13] Cao, L., K. Mohan, P. Xu, and B. Ramesh (2004). How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. In *Hawaii International Conference on System Sciences (HICSS'04)*. IEEE.

[14] Charmaz, K. (2000). Grounded Theory: Objectivist and Constructivist Methods. In *Handbook of Qualitative Research* (2nd ed.)., pp. 509–535. SAGE.

[15] Charmaz, K. (2006). *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. SAGE.

[16] Checkland, P. (1981). *Systems Thinking, Systems Practice*. Wiley.

[17] Conboy, K. (2009). Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. *Information Systems Research 20*(3), 329–354.

[18] Conboy, K. and B. Fitzgerald (2006). The Views of Experts on the Current State of Agile Method Tailoring. In *Organizational Dynamics of Technology-Based Innovation: Diversifying the Research Agenda*, pp. 217–234. Springer.

[19] Conboy, K. and B. Fitzgerald (2010). Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion. *ACM Transactions on Software Engineering and Methodology (TOSEM) 20*(1), 2:1–2:30.

[20] Corbin, J. M. and A. L. Strauss (1990). Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology 13*(1), 3–21.

[21] Deemer, P., G. Benefield, C. Larman, and B. Vodde (2012). *The Scrum Primer* (2nd ed.). InfoQ.

[22] Diebold, P., J.-P. Ostberg, S. Wagner, and U. Zendler (2015). What Do Practitioners Vary in Using Scrum? In *Agile Processes in Software Engineering and Extreme Programming*, pp. 40–51. Springer.

[23] Dingsøyr, T., S. Nerur, V. Balijepally, and N. B. Moe (2012). A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software 85*(6), 1213–1221.

[24] Drægert, A. and D. S. Petersen (2015). Understanding scrumbut. Aalborg University. Student Report.

[25] Eloranta, V.-P., K. Koskimies, and T. Mikkonen (2016). Exploring ScrumBut - An empirical study of Scrum anti-patterns. *Information and Software Technology 74*, 194–203.

[26] Elsevier (2016). About Scopus. `https://www.elsevier.com/solutions/scopus`. [Online; Accessed 2016-05-02].

[27] Fernandes, C. (2012). There and back again: From iterative to flow... and back to iterative! In *Agile Conference (AGILE'12)*, pp. 103–110. IEEE.

[28] Fitzgerald, B., N. L. Russo, and T. O'Kane (2003). Software Development Method Tailoring at Motorola. *Communications of the ACM 46*(4), 64–70.

[29] Fowler, M. and J. Highsmith (2001). The agile manifesto. *Software Development 9*(8), 28–35.

[30] Glaser, B. G. (1978). *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory*. Sociology Press.

[31] Glaser, B. G. and J. Holton (2004). Remodeling Grounded Theory. In *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, Volume 5. FQS.

[32] Glaser, B. G. and A. L. Strauss (1967). *The Discovery of Grounded Theory - Strategies for Qualitative Research*. Aldine Transaction.

[33] Grunert, K. G. and S. C. Grunert (1995). Measuring subjective meaning structures by the laddering method: Theoretical considerations and methodological problems. *International journal of research in marketing 12*(3), 209–225.

[34] Gunnerson, E. (2006). Scrumbut. `https://blogs.msdn.microsoft.com/ericgu/2006/10/13/scrumbut/`. [Online; Accessed 2016-05-06].

[35] Heeager, L. T. and J. Rose (2015). Optimising agile development practices for the maintenance operation: nine heuristics. *Empirical Software Engineering 20*(6), 1762–1784.

[36] Heikkilä, V. T., M. Paasivaara, and C. Lassenius (2013). ScrumBut, But Does it Matter? A Mixed-Method Study of the Planning Process of a Multi-team Scrum Organization. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 85–94.

[37] Henderson-Sellers, B. and M. K. Serour (2005). Creating a Dual-Agility Method: The Value of Method Engineering. *Journal of Database Management 16*(4), 1–23.

[38] Hewlett-Packard Development Company (2015). Agile is the new normal: Adopting Agile project management. `http://www8.hp.com/h20195/v2/getpdf.aspx/4AA5-7619ENW.pdf?ver=1.0`. [Online; Accessed 2016-06-28].

[39] Hoda, R., P. Kruchten, J. Noble, and S. Marshall (2010). Agility in context. *ACM SIGPLAN Notices 45*(10), 74–88.

[40] Hoda, R., J. Noble, and S. Marshall (2009). Negotiating Contracts for Agile Projects: A Practical Perspective. In *Agile Processes in Software Engineering and Extreme Programming*, pp. 186–191. Springer.

[41] Hoda, R., J. Noble, and S. Marshall (2010a). Agile Undercover: When Customers Don't Collaborate. In *Agile Processes in Software Engineering and Extreme Programming*, pp. 73–87. Springer.

[42] Hoda, R., J. Noble, and S. Marshall (2010b). Balancing Acts: Walking the Agile Tightrope. In *ICSE Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'10)*, pp. 5–12. ACM.

[43] Hoda, R., J. Noble, and S. Marshall (2011). The impact of inadequate customer collaboration on self-organizing Agile teams. *Information and Software Technology 53*(5), 521–534.

[44] Hong, N., J. Yoo, and S. Cha (2010). Customization of Scrum Methodology for Outsourced E-Commerce Projects. In *Asia Pacific Software Engineering Conference (APSEC'10)*, pp. 310–315. IEEE.

[45] Hossain, E., P. L. Bannerman, and R. Jeffery (2011). Towards an Understanding of Tailoring Scrum in Global Software Development: A Multi-case Study. In *International Conference on Software and Systems Process (ICSSP'11)*, pp. 110–119. ACM.

[46] IEEE (2016). Peer Review. `http://www.ieee.org/publications_standards/publications/authors/publish_benefits.html`. [Online; Accessed 2016-05-02].

[47] Iivari, J., R. Hirschheim, and H. K. Klein (1998). A paradigmatic analysis contrasting information systems development approaches and methodologies. *Information Systems Research 9*(2), 164–193.

[48] Iivari, J. and M. Huisman (2001). The relationship between organisational culture and the deployment of systems development methodologies. In *International Conference on Advanced Information Systems Engineering*, pp. 234–250. Springer.

[49] Iivari, J. and N. Iivari (2011). The relationship between organizational culture and the deployment of agile methods. *Information and Software Technology 53*(5), 509–520.

[50] Inayat, I., M. Noor, and Z. Inayat (2012). Successful Product-based Agile Software Development without Onsite Customer: An Industrial Case Study. *International Journal of Software Engineering and its Applications 6*(2), 1–14.

[51] Jeffries, R. (2013). Fractional Scrum, or "Scrum-But". `http://agileatlas.org/articles/item/fractional-scrum-or-scrum-but`. [Online; Accessed 2016-08-06].

[52] Karlsson, F. (2008). A Wiki-based Approach to Method Tailoring. In *International Conference on the Pragmatic Web (ICPW'08)*, pp. 13–22. ACM.

[53] Karlsson, F. and P. J. Ågerfalk (2007). Multi-Grounded Action Research in Method Engineering: The MMC Case. In *Situational Method Engineering: Fundamentals and Experiences*, pp. 63–78. Springer.

[54] Karlsson, F. and P. J. Ågerfalk (2004). Method configuration: adapting to situational characteristics while creating reusable assets. *Information and Software Technology 46*(9), 619–633.

[55] Karlsson, F. and P. J. Ågerfalk (2005). Method-User-Centred Method Configuration. In *Situational Requirements Engineering Processes (SREP'05)*, pp. 31–43.

[56] Kelly, G. A. (1955). *The psychology of personal constructs*. WW Norton and Company.

[57] Kendall, J. (1999). Axial Coding and the Grounded Theory Controversy. *Western journal of nursing research 21*(6), 743–757.

[58] Khan, A. I., R. J. Qurashi, and U. A. Khan (2011). A Comprehensive Study of Commonly Practiced Heavy and Light Weight Software Methodologies. *International Journal of Computer Science Issues 8*(2), 441–450.

[59] Kniberg, H. (2011). What to do When Scrum Doesn't Work. `https://www.scrumalliance.org/community/articles/2011/february/what-to-do-when-scrum-doesn%E2%80%99t-work`. [Online; Accessed 2016-05-06].

[60] Kniberg, H. (2015). *Scrum and XP from the Trenches - 2nd Edition*. InfoQ.

[61] Krishna, V. and A. Basu (2011). Scrum+ :: Is it "ScrumBut" or "ScrumAnd". pp. 1–4. IEEE.

[62] Lee, G. and W. Xia (2010). Toward agile: an integrated analysis of quantitative and qualitative field data on software development agility. *MIS Quarterly 34*(1), 87–114.

[63] Lorber, A. A. and K. D. Mish (2013). How We Successfully Adapted Agile for a Research-Heavy Engineering Software Team. *Agile Conference (AGILE'13)*, 156–163.

[64] Meso, P. and R. Jain (2006). Agile Software Development: Adaptive Systems Principles and Best Practices. *Information Systems Management 23*(3), 19–30.

[65] Mingers, J. (2001). Combining IS Research Methods: Towards a Pluralist Methodology. *Information Systems Research 12*(3), 240–259.

[66] Molnar, W. and J. Nandhakumar (2009). Managing Projects in an Embedded System Development Context: An In-Depth Case Study from an Improvisational Perspective. In *Hawaii International Conference on System Sciences (HICSS'09)*, pp. 1–10. IEEE.

[67] Myers, M. D. and M. Newman (2007). The qualitative interview in is research: Examining the craft. *Information and organization 17*(1), 2–26.

[68] Nejmeh, B. and D. S. Weaver (2014). Leveraging scrum principles in collaborative, inter-disciplinary service-learning project courses. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pp. 1–6. IEEE.

[69] Ngwenyama, O. and P. A. Nielsen (2003). Competing Values in Software Process Improvement: An Assumption Analysis of CMM From an Organizational Culture Perspective. *IEEE Transactions on Engineering Management 50*(1), 100–112.

[70] Pauly, D., B. Michalik, and D. Basten (2015). Do Daily Scrums Have to Take Place Each Day? A Case Study of Customized Scrum Principles at an E-commerce Company. In *Hawaii International Conference on System Sciences (HICSS'15)*, Volume 48, pp. 5074–5083. IEEE.

[71] Persson, J. S., J. Nørbjerg, and P. A. Nielsen (2016). Improving ISD Agility in Fast-Moving Software Organizations. In *European Conference on Information Systems (ECIS'16)*, pp. 1–16. AIS.

[72] Quinn, R. E. and M. R. McGrath (1985). The transformation of organizational cultures: A competing values perspective. *Organizational culture*, 315–334.

[73] Rodríguez, P., J. Partanen, P. Kuvaja, and M. Oivo (2014). Combining lean thinking and agile methods for software development: A case study of a finnish provider of wireless embedded systems detailed. In *Hawaii International Conference on System Sciences (HICSS'14)*, pp. 4770–4779. IEEE.

[74] Rubin, K. S. (2012). *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Addison-Wesley Professional.

[75] Schreiber, R. S. and P. N. Stern (2001). *Using Grounded Theory In Nursing*. Springer.

[76] Schultze, U. and M. Avital (2011). Designing interviews to generate rich data for information systems research. *Information and Organization 21*(1), 1–16.

[77] Schwaber, K. (1997). SCRUM Development Process. In *Business Object Design and Implementation*, pp. 117–134. Springer.

[78] Schwaber, K. (2015). ScrumBut. `https://www.scrum.org/scrumbut`. [Online; Accessed 2015-12-14].

[79] Schwaber, K. and M. Beedle (2001). *Agile Software Development with Scrum*. Prentice Hall PTR.

[80] Schwaber, K. and J. Sutherland (2013). The Scrum guide. `http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.pdf`. [Online; Accessed 2015-12-15].

[81] Schwaber, K. and J. Sutherland (2015). The History of Scrum. `http://www.scrumguides.org/history.html`. [Online; Accessed 2015-12-01].

[82] ScrumAlliance (2015). The 2015 State of Scrum Report. `https://www.scrumalliance.org/scrum/media/scrumalliancemedia/files%20and%20pdfs/state%20of%20scrum/scrum-alliance-state-of-scrum-2015.pdf`. [Online; Accessed 2016-07-28].

[83] Sienkiewicz, L. D. and L. A. Maciaszek (2011). Adapting scrum for third party services and network organizations. In *Federated Conference on Computer Science and Information Systems (FedCSIS'11)*, pp. 329–336. IEEE.

[84] Stacey, P. and J. Nandhakumar (2008). Opening up to agile games development. *Communications of the ACM 51*(12), 143–146.

[85] Strauss, A. L. and J. M. Corbin (1998). *Basics of Qualitative Research: Second Edition: Techniques and Procedures for Developing Grounded Theory*. SAGE.

[86] Strode, D. E., S. L. Huff, and A. Tretiakov (2009). The impact of organizational culture on agile method use. In *Hawaii International Conference on System Sciences (HICSS'09)*, pp. 1–9. IEEE.

[87] Swaminathan, B. and K. Jain (2012). Implementing the lean concepts of continuous improvement and flow on an agile software development project: An industrial case study. In *Agile India 2012*, pp. 10–19. ASCI.

[88] Takeuchi, H. and I. Nonaka (1986). The New New Product Development Game. *Harvard Business Review 64*(1), 137–146.

[89] Tanner, M. and A. Mackinnon (2013). Sources of Disturbances Experienced During a Scrum Sprint. In *International Conference on Information Systems Management and Evaluation (ICIME'13))*, pp. 255–262. ACPI.

[90] Tanner, M. and C. Wallace (2012). Towards an Understanding of the Contextual Influences on Distributed Agile Software Development: a Theory of Practice Perspective. In *European Conference on Information Systems*.

[91] Thomsom Reuters (2016). The Thomsom Reuters Journal Selection Process. `http://wokinfo.com/essays/journal-selection-process/`. [Online; Accessed 2016-05-02].

[92] Tiwana, A. and M. Keil (2004). The one-minute risk assessment tool. *Communications of the ACM 47*(11), 73–77.

[93] Upender, B. (2005). Staying agile in government software projects. In *Agile Development Conference (ADC'05)*, pp. 153–159. IEEE.

[94] Vallon, R., C. Drager, A. Zapletal, and T. Grechenig (2014). Adapting to changes in a project's DNA: A descriptive case study on the effects of transforming agile single-site to distributed software development. In *Agile Conference (AGILE'14)*, pp. 52–60.

[95] Vallon, R., S. Strobl, M. Bernhart, and T. Grechenig (2013). Inter-organizational Co-development with Scrum: Experiences and Lessons Learned from a Distributed Corporate Development Environment. In H. Baumeister and B. Weber (Eds.), *Agile Processes in Software Engineering and Extreme Programming*, pp. 150–164. Springer.

[96] Verdugo, J., M. Rodríguez, and M. Piattini (2014). Using agile methods to implement a laboratory for software product quality evaluation. In *Agile Processes in Software Engineering and Extreme Programming*, pp. 143–156. Springer.

[97] Verheyen, G. (2014). Maximizing Scrum. `https://www.scrum.org/Blog/ArtMID/1765/ArticleID/15/Maximizing-Scrum`. [Online; Accessed 2016-06-06].

[98] VersionOne (2015). 10th Annual State of Agile™ Survey. `https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf`. [Online; Accessed 2015-09-28].

[99] Vilain, P. and A. J. B. Martins (2011). Neglecting agile principles and practices: A case study. In *International Conference on Software Engineering and Knowledge Engineering (SEKE'11)*, pp. 596–601.

[100] Wang, X., K. Conboy, and M. Pikkarainen (2012). Assimilation of agile practices in use. *Information Systems Journal 22*(6), 435–455.

[101] Wolfswinkel, J. F., E. Furtmueller, and C. P. M. Wilderom (2013). Using grounded theory as a method for rigorously reviewing literature. *European Journal of Information Systems 22*, 45–55.

Methodology Considerations A

In this chapter we go over some of the considerations made when deciding how to combine the literature study with an interview study. To learn about conducting interviews, we read "Designing interviews to generate rich data for information systems research" by Schultze and Avital [76]; a discussion of the different approaches presented in the paper has no place in the main report, but we want to include our argumentation for why we ultimately went with the approach we did.

A.1 Combining the Studies

A number of different methods were considered for the overall methodology of the project. We will not present all in detail, but to present some of the considerations we made, some of the possible configurations we considered are presented here.

The first configuration separates the two studies completely. Coding of the empirical data is done in a separate NVivo project in a sequential fashion. That way the two studies are treated as two completely different studies – one based on [101] and the other on [2]. The conclusions can then be compared in the end, but the approach has a big amount of uncertainty, as the conclusions might not have much in common. This is particularly true given the open-minded nature of the approach by Adolph et al. [2]. Unless the theory from the literature study is directly taken into account during the analysis of the interviews, it is unlikely the findings of the two studies would align.

The second configuration would still keep the two studies in separate NVivo projects, but the studies would run in parallel. This approach still impedes the ability to be open-minded in the analysis of the interviews, but the two studies would influence each other much more directly. Instead of simply comparing the results, the studies would shape each other and findings from one study can help guide the direction of the other. Consequently, the main category used as a focus in the interview study is influenced, not only by patterns identified in the interviews themselves, but also by the ones found in the literature.

The third configuration joins the NVivo projects, but the studies are carried out sequentially. That way, the empirical study is conducted on top of the literature study, allowing a quick identification of a main category to pursue, but also assuming that the interviews have been carried out in such a way that there is a direct relation

between the findings from the literature and the content of the interviews. Coding in the same NVivo project requires a mapping of the two coding styles. For example, it is not certain that the selective codes from the Straussian approach directly correlate to the understanding of selective codes in the Glaserian approach.

The fourth configuration also stays within one NVivo project, but as with the second configuration the studies are carried out in parallel. A parallel approach makes it even more important to keep the coding styles compatible. This configuration would completely integrate the two studies, essentially making one study based on two data sets. With the two data sets being as different as they are – one being much more detailed than the other – it can be perceived that one might overshadow the other, thus losing the nuanced perspective that is the goal of using the two different data sources.

As described in Chapter 3 we decided to go with an approach resembling the fourth configuration.

A.2 Interview Approaches

As for conducting the actual interview Schultze and Avital [76] suggest three different approaches: Appreciative, photo-diary, and laddering.

In appreciative interviewing, the interviewer uses positive feedback to encourage the interviewee to give a rich account of the subject of the interview. The interview switches between a retrospective phase (the current situation) and a prospective phase (possible future situation). Appreciative interviewing especially focuses on making the participant envision possibilities, based on what already is, and it is especially well suited for coming up with new or developing existing ideas. As the purpose of this study is of a more reflective manner, the appreciative interviewing approach is not ideal.

In the photo-diary interview method the interviewee prepares for the interview by taking pictures and writing a diary/log relevant to the interview subject. The photos are then used as a visual aid in the interviews, allowing for deeper reflections on what happened in a given situation. The photo-diary approach has a good fit with the purpose of this study, but due to the relatively big amount of work required by the interviewee, we decided to instead go with the third option: laddering.

The laddering approach recommended by Schultze and Avital [76] has two parts. Partly they suggest using Repertory Grid (RepGrid) method, based on and devised by Kelly [56]. The RepGrid method is very good for comparing the interviewee's views on different aspects of the topic in a systematic manner, but it also takes a lot of time to set up. The second part is the laddering itself. Paired with the RepGrid, a means-ends chain is identified by asking a sequence of why's about the distinctions found in the RepGrid. As we are interested in covering all aspects of the interviewee's Scrum use, we decided to go with only the laddering part. Without the RepGrid as support the process becomes much simpler, and we minimise the risk that we spend so much time on setting up the RepGrid that we run out of time to ask the important questions. This way we simply ask questions about each Scrum

practice and extend the questions with as many levels of "why?" as the interviewee is able to answer. Using this approach allows us to get a better understanding of their arguments by using laddering, while still leaving enough flexibility to get around every Scrum practice.

The use of laddering (i.e., continuously asking for the reasoning behind the previous answer) does not guarantee that the answers reflect an absolute truth. Instead it gives a reasoning that is true in the eyes of the interviewee, and by combining the answers with a Grounded Theory approach we hope to be able to find patterns that explain the situation in more general terms.

B Complete Literature Search Queries

```
"query": {
    acmdlTitle:(+scrumbut) OR recordAbstract:(+scrumbut) OR
    keywords.author.keyword:(+scrumbut) OR
    acmdlTitle:(+scrum +adapt*) OR recordAbstract:(+scrum +adapt*) OR
    keywords.author.keyword:(+scrum +adapt*) OR
    acmdlTitle:(+scrum +tailor*) OR recordAbstract:(+scrum +tailor*) OR
    keywords.author.keyword:(+scrum +tailor*) OR
    acmdlTitle:(+scrum +modif*) OR recordAbstract:(+scrum +modif*) OR
    keywords.author.keyword:(+scrum +modif*) OR
    acmdlTitle:(+scrum +customi*) OR recordAbstract:(+scrum +customi*) OR
    keywords.author.keyword:(+scrum +customi*)
}
"filter": {"publicationYear":{ "gte":2001 }}
```

Figure B.1: Query for *ACM*.

```
queryText=(scrumbut) OR (scrum adapt*) OR (scrum tailor*) OR (scrum modif*) OR
(scrum customi*) AND ranges=2001_2016_Year[1]
```

Figure B.2: Query for *IEEE Xplore*

```
(TITLE-ABS-KEY(scrumbut) AND PUBYEAR > 2001 AND SUBJAREA(COMP)) OR
(TITLE-ABS-KEY(scrum adapt*) AND PUBYEAR > 2001 AND SUBJAREA(COMP)) OR
(TITLE-ABS-KEY(scrum tailor*) AND PUBYEAR > 2001 AND SUBJAREA(COMP)) OR
(TITLE-ABS-KEY(scrum modif*) AND PUBYEAR > 2001 AND SUBJAREA(COMP)) OR
(TITLE-ABS-KEY(scrum customi*) AND PUBYEAR > 2001 AND SUBJAREA(COMP))
```

Figure B.3: Query for *Scopus*.

```
TS=(scrumbut) AND PY=(2001-2016) AND SU=(Computer Science) OR
TS=(scrum AND adapt*) AND PY=(2001-2016) AND SU=(Computer Science) OR
TS=(scrum AND tailor*) AND PY=(2001-2016) AND SU=(Computer Science) OR
TS=(scrum AND modif*) AND PY=(2001-2016) AND SU=(Computer Science) OR
TS=(scrum AND customi*) AND PY=(2001-2016) AND SU=(Computer Science)
```

Figure B.4: Query for *Web of Science*

[1] `http://ieeexplore.ieee.org/search/searchresult.jsp?action=search&sortType=&rowsPerPage=&searchField=Search_All&matchBoolean=true&queryText=(scrumbut)%20OR%20(scrum%20adapt*)%20OR%20(scrum%20tailor*)%20OR%20(scrum%20modif*)%20OR%20(scrum%20customi*)&ranges=2001_2016_Year`

Reviewed Papers C

Outlet	Article	Ref.
Agile Conference (4)	Block (2011)	[8]
	Fernandes (2012)	[27]
	Lorber and Mish (2013)	[63]
	Vallon, Drager, Zapletal, and Grechenig (2014)	[94]
International Conference on Agile Software Development (3)	Diebold, Ostberg, Wagner, and Zendler (2015)	[22]
	Vallon, Strobl, Bernhart, and Grechenig (2013)	[95]
	Verdugo, Rodríguez, and Piattini (2014)	[96]
Agile Development Conference (1)	Upender (2005)	[93]
Asia Pacific Software Engineering Conference (1)	Hong, Yoo, and Cha (2010)	[44]
Empirical Software Engineering (1)	Heeager and Rose (2015)	[35]
Federated Conference on Computer Science and Information Systems (1)	Sienkiewicz and Maciaszek (2011)	[83]
Hawaii International Conference on System Sciences (1)	Pauly, Michalik, and Basten (2015)	[70]
Information and Software Technology (1)	Eloranta, Koskimies, and Mikkonen (2016)	[25]
International Conference on Information Systems Management and Evaluation (1)	Tanner and Mackinnon (2013)	[89]
International Conference on Software and Systems Process (1)	Hossain, Bannerman, and Jeffery (2011)	[45]
International Conference on Software Engineering and Knowledge Engineering (1)	Vilain and Martins (2011)	[99]
International Journal of Software Engineering and its Applications (1)	Inayat, Noor, and Inayat (2012)	[50]

Table C.1: Reviewed articles

Interview Guide D

Generel info

Har du eller andre på teamet certifikater inden for Scrum?

Hvor stort er teamet, og hvordan er det sammensat?

Hvor længe har du arbejdet med Scrum? I forhold til gennemsnittet på teamet?

Hvad er din(e) rolle(r)?

Probes og uddybende spørgsmål

Spørg ind til (husk 3-5x why):

- Scrum Master, Product Owner, Development Team
- Sprint Backlog Product Backlog, Increment
- Daily Scrum, Retrospective, Review, Planning, Sprint

Mulige uddybninger:

Kan du uddybe, hvordan I bruger X?

Kan du nævne noget I aktivt har valgt at ændre (i forhold til X)?

Er der noget I har valgt at undlade?

Er der noget ekstra I har valgt at inkludere?

Hvad har fungeret godt/skidt i det projekt du arbejder på?

Pris i forhold til budget.

Features i forhold til forventninger.

Miljø / arbejdsmoral.

Hvordan går det i forhold til forventninger generelt?

Bekymringer.

Scrum vs. plandrevet?

Afslut med: Har du en afsluttende kommentar eller spørgsmål til os?

Rollefordeling

Interviewperson:

Snakker om Scrum!

Så vidt muligt kun én person.

Interviewer:

Stiller spørgsmål og holder gang i samtalen.

Spørg specielt ind, når der nævnes ændringer.

Vær opmærksom på at få så mange niveauer med som muligt (laddering).

Notetager:

Holder styr på tid.

Skriver noter i forhold til hver Scrum practice og noterer tidspunkt for note.

Bryd ind i samtalen, hvis vigtigt emne misses.

Optagelse:

1 stk computer med mikrofon

1 stk tablet

Sørg for at starte optagere og tid på samme tid.

Start stopur, så det er let for notetager at se tidspunkt.

Tools E

In this chapter the tools we used for reference management (Zotero) and coding (NVivo) are described. The tools deserve a description because of their central role in both data collection and coding.

E.1 Zotero

Zotero is a reference management tool, and it was used to organise and filter the papers found in the four databases as described in Section 3.2. Through a browser extension all results from any given query can be easily imported into the tool for further processing. As opposed to NVivo, Zotero synchronises with a server online, allowing multiple people to work seamlessly on the same project.

Figure E.1 shows a screenshot of the main window of Zotero. The window contains three parts (left, middle, and right), and most of the work is done in those three parts. The left view provides an overview of the folders used to organise the filtering process illustrated on Figure 3.2. For example, all search results are added to folder the folder *Scrumbut [1] - With Duplicates*, and after finishing all queries the entries are *copied* to *Scrumbut - [2] Without Duplicates*. In the second folder, the tool's duplication removal tool is used to filter most duplicates, and the ones that were missed are removed manually. The entries are copied rather than moved to preserve the numbers needed for illustrating the process in Figure 3.2.

The middle view is for interacting with the papers, including deleting and moving them to different folders. It allows sorting by any of the columns, which is especially useful for preventing clashes when more than one person is working on it at the same time.

The right view shows details about the selected paper, including tags and notes, which are especially useful for marking papers that are in the grey zone in terms of whether or not they should be included. More importantly the *Info* tab includes the abstract, which is very useful for step 3 (refining by abstract). Because of the aforementioned synchronisation with a server, the storage space is limited to 300 MB in the free version, which is not enough to store the PDF with each entry. Showing the abstract directly in the tool means the PDFs do not need to be downloaded

until after refining by abstract, leaving us with only 62 papers to download instead of 236.

The tool does have some quirks, most notably the deletion process. If an item is deleted by pressing the *delete* button it is not removed from the index listing all items, which can be confusing. Instead the item has to me moved to the *Trash* folder and then deleted from there. If items are deleted the wrong way, the easiest way to restore the project is by creating a new project from scratch and move the contents from each folder over to the new project.

Overall the tool was invaluable for keeping track of the refinement process, and especially the browser extension saved us countless hours; adding almost 400 papers manually would have taken many hours.

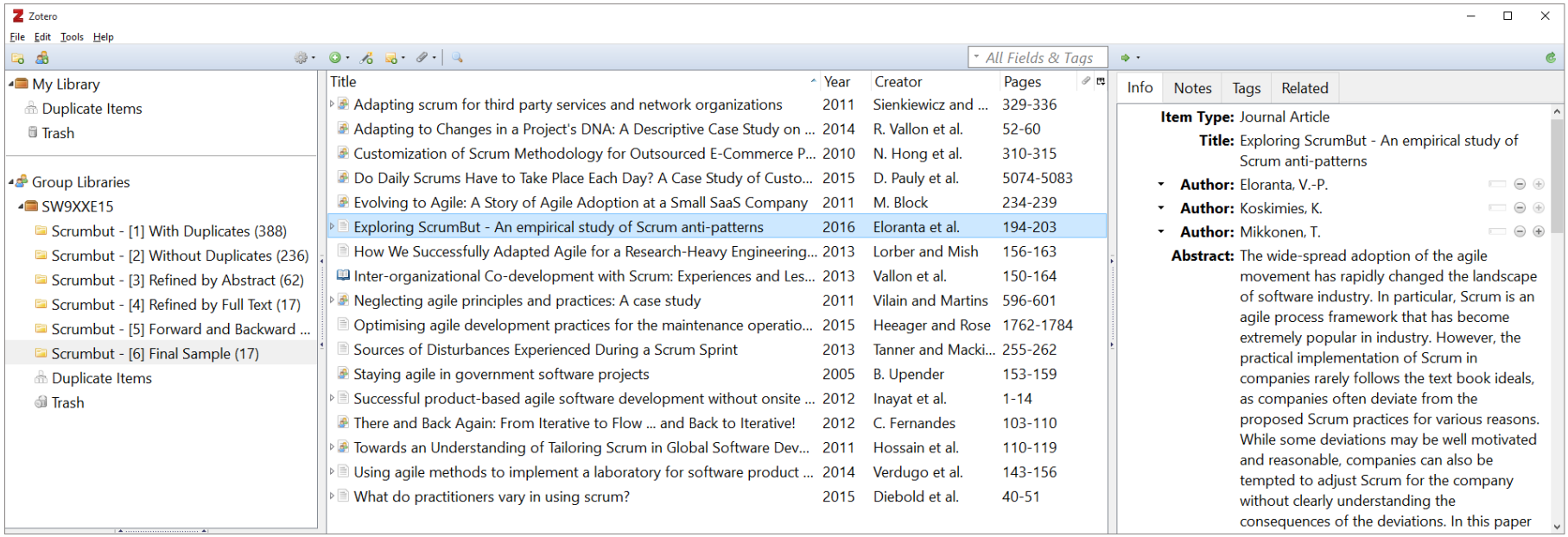


Figure E.1: Screenshot from the research tool *Zotero*.

E.2 NVivo

NVivo is a tool for organising qualitative data analysis, and it was useful for managing both papers and audio from the interviews. Each source, e.g. audio file or pdf, can be imported into the program, and when an NVivo project is saved the sources are embedded directly into the project file, making it easy to move around and share without having to worry about each individual file. Figure E.2 shows an overview of all the imported sources. Double-clicking a source opens either the audio editor or pdf editor, allowing to code the source.

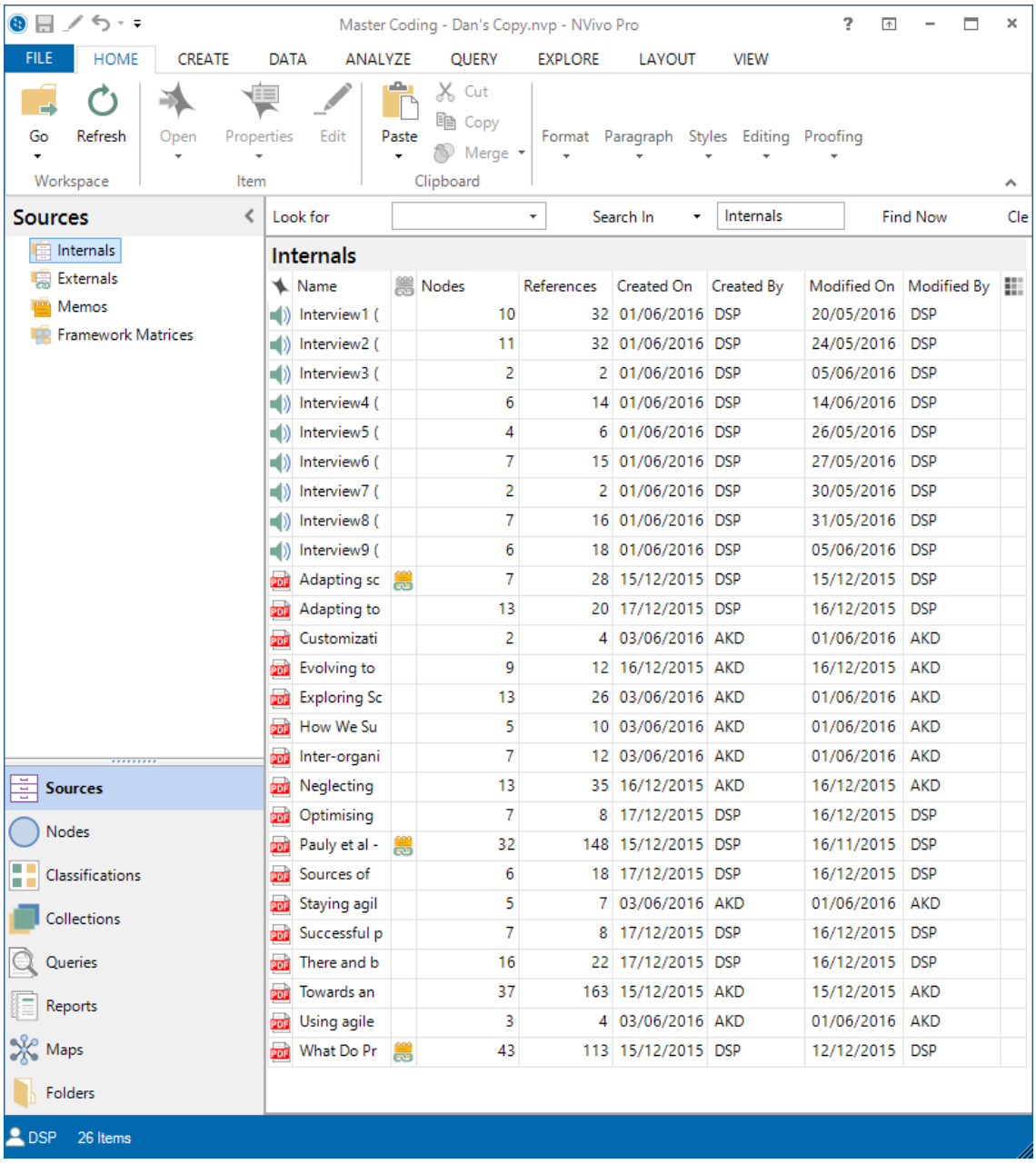


Figure E.2: Sources, interviews and papers, in NVivo.

Figure E.3 shows a paper opened in the pdf editor. To mark an area as relevant, an *area annotation* was added, covering the paragraph with a transparent blue area. When a ScrumBut is identified, the area is highlighted and added to an existing node (shown in the figure) or a new node. Already coded areas are marked by yellow.

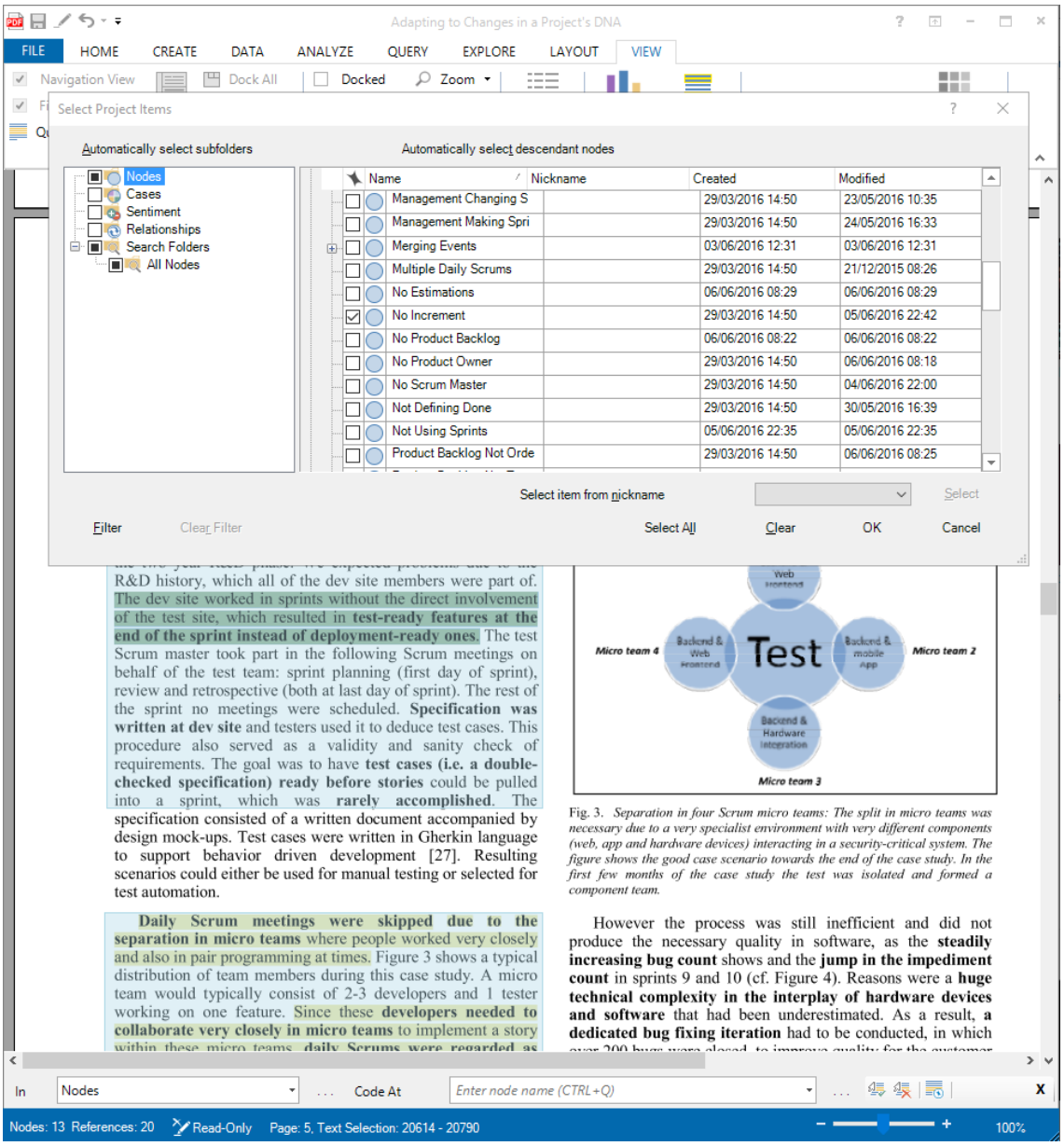


Figure E.3: Coding a paper in NVivo.

Figure E.4 shows the editing of an audio file. One of the options in the *Playback* menu allows for speeding up the audio, making it possible to go through the interview much faster. It was found to be unnecessary to transcribe the whole file, and instead only areas where ScrumButs were mentioned were transcribed and added to one or more relevant nodes. In addition it was found to not add enough value to mark relevant areas as we did in the papers because any part of the interview could be relevant, and marking every area would be too time consuming.

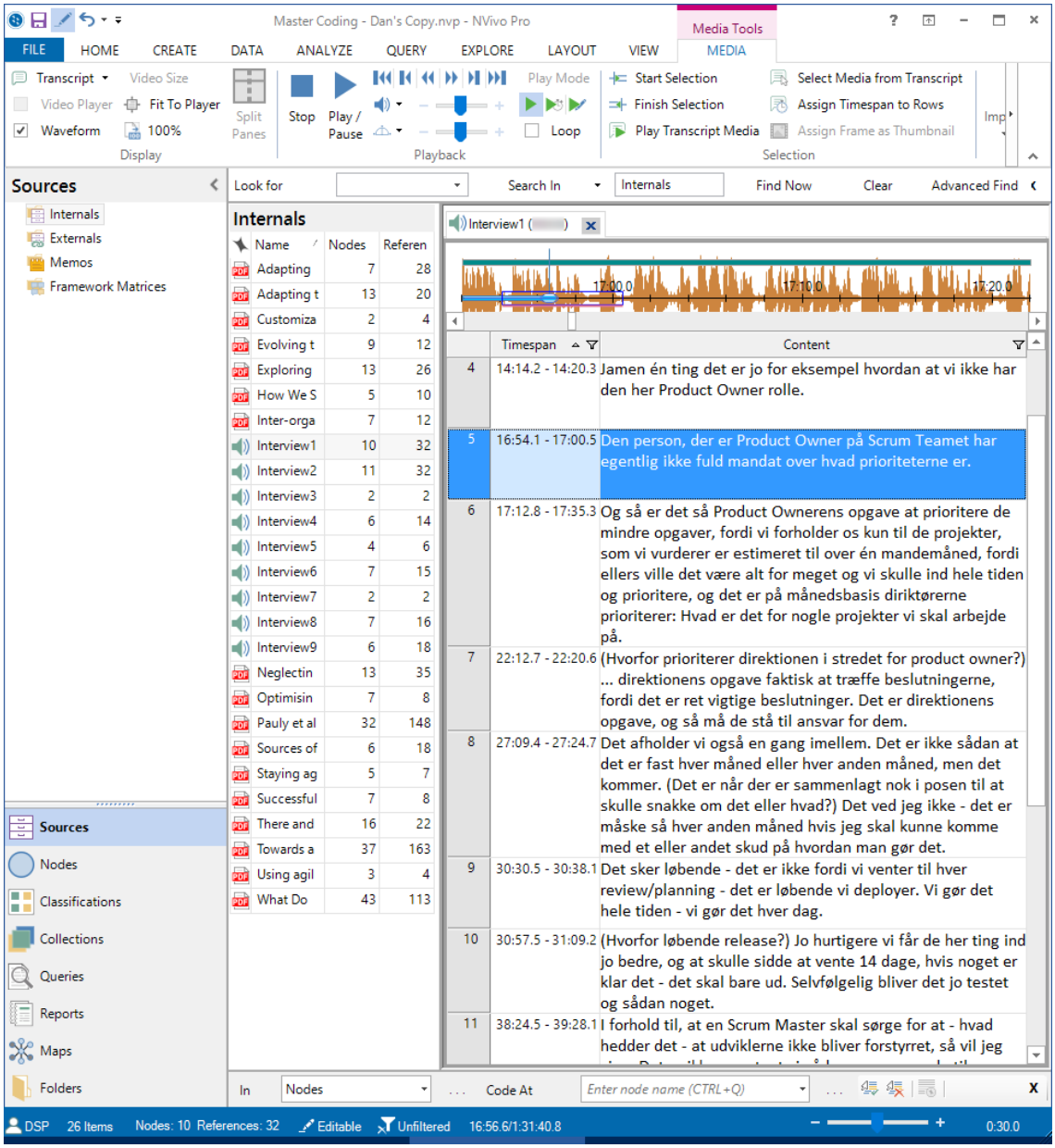


Figure E.4: Editing the transcript for an audio file.

Figure E.5 shows the node view, giving an easy overview of all identified nodes. In our case we have one node for each ScrumBut. The nodes are easily sortable on, for example, number of sources or number of codes, and multiple layers can be added, allowing groups of nodes to be collected – useful for managing the granularity of the ScrumButs.

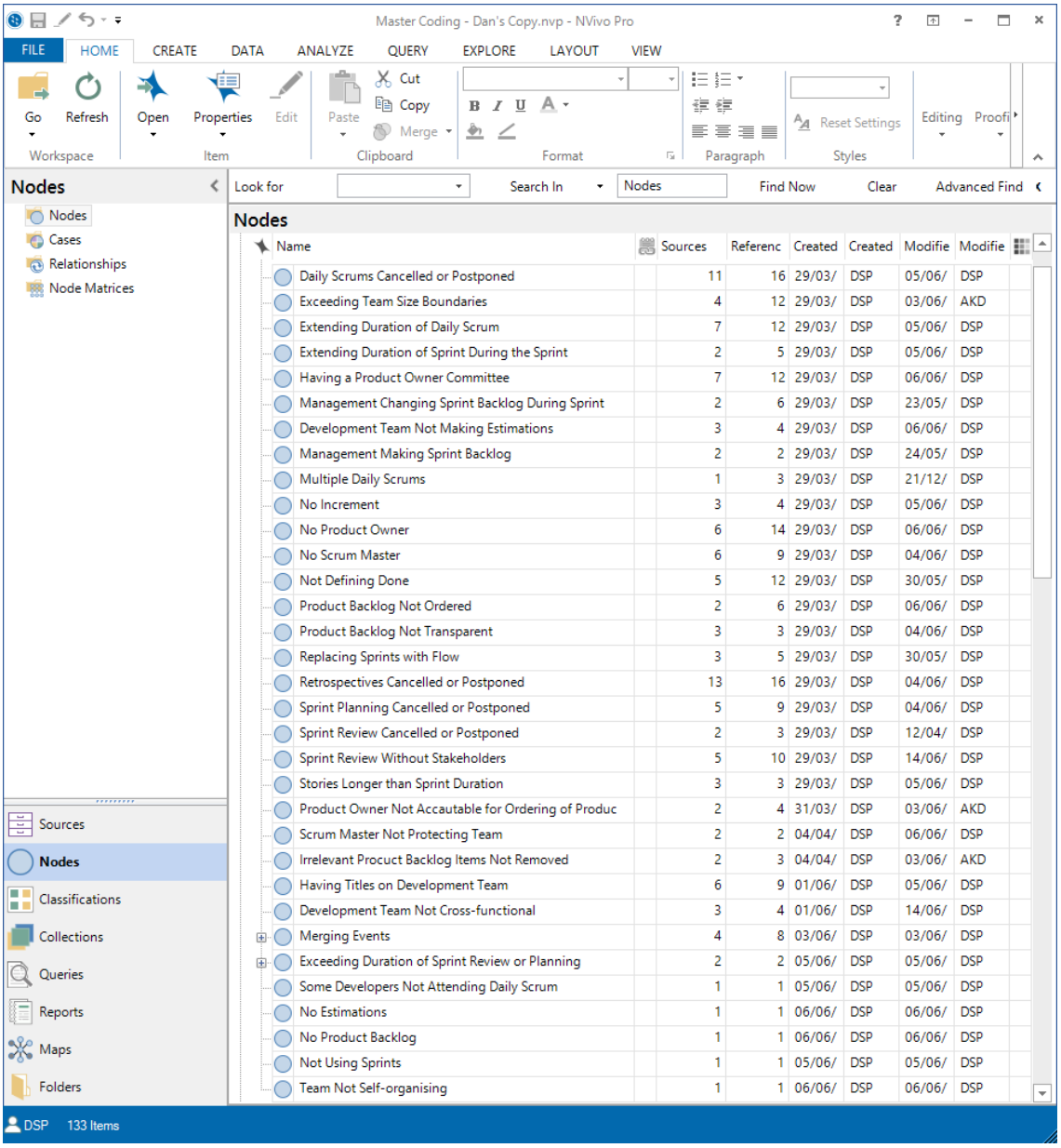


Figure E.5: Nvivo's node view, showing all nodes.

Each code collected under a node from any source is easily viewable by double-clicking a node from the node view. Figure E.6 shows some of the codes for *Daily Scrums Cancelled or Postponed* with examples from both papers and interviews. One code is highlighted in blue, indicating it has an annotation adding a comment to it. In some cases the codes can be difficult to read in this view, especially those made from highlighting text in a pdf. In cases like that, or if further context is needed, the title of the source can be clicked to open the full source.

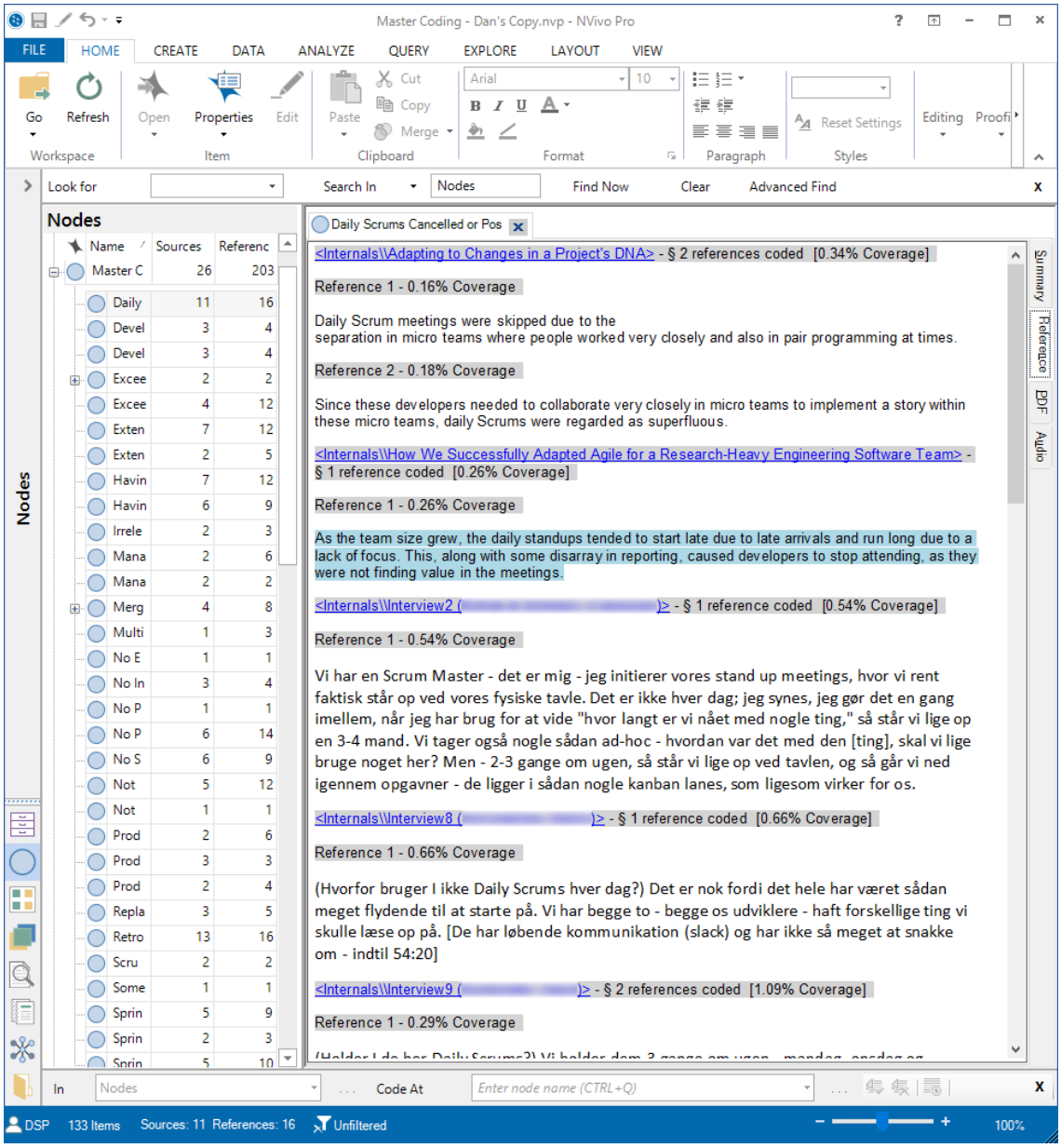


Figure E.6: Codes belonging to the node *Daily Scrums Cancelled or Postponed*.

Overall, NVivo was found to have a number of quirks making it slightly harder to use, but it still was a much better solution than organising the coding manually.

Article F

The article starts on the next page.

ScrumBut in Professional Software Development

Alexander Drægert and Dan Skøtt Petersen

Aalborg University, Aalborg Ø 9220, Denmark,
{adrage11, dspe11}@student.aau.dk

Abstract. Agile methodologies are widely used in the software industry with Scrum being the most common framework. Scrum is a small framework with few, but well-defined, practices. While many companies still deviate from the textbook version of Scrum, limited research has investigated the underlying explanation of why. Against this backdrop, we report an investigation of ScrumButs in professional software development based on analysis of 17 empirical research papers on Scrum modifications and interviews with 9 Scrum practitioners from 7 software companies. We found instances of ScrumButs pertaining to all parts of the Scrum framework and analyze the associated reasoning with the competing values model of organizational culture. The analysis shows how particular ScrumButs may involve different forms of reasoning reflecting different forms of organizational culture. We discuss how these findings may nuance our assessments of ScrumButs beyond implicit value judgments of being inherently good or bad.

1 Introduction

Software development is a complex endeavour with numerous variables determining whether or not a project is successful. Tiwana and Keil [23] present six risk factors which impact the success rate of a project. Among those risk factors the one with the highest impact factor is *Methodology Fit*. The most commonly used agile methodology is, according a survey from VersionOne Inc. [28], Scrum with nearly 70% usage. However, when inspected in more detail, many make changes to Scrum for one reason or another. These changes to Scrum are referred to as ScrumButs, and it is debated whether these ScrumButs are signs of dysfunction or a natural part of any software development practice (e.g. [12, 17]).

In this study we will first investigate **which** ScrumButs can be identified in professional software development and **how** their emergence is reasoned. Secondly, we will investigate **why** these ScrumButs emerge. To do so, we review 17 empirical research papers and interview 9 practitioners from 7 different companies.

This paper is structured as follows: Section 2 introduces ScrumBut and organizational culture, which forms the foundation for this paper. Section 3 provides an overview of the methodology used to conduct the study. Section 4 summarizes the findings of the study in three tables. Section 5 discusses related research and the contributions of this study, followed by limitations and suggestions for future research. Section 6 concludes the paper.

2 Background

Scrum is used by the majority of organizations practising agile software development [18], [28], but many of the organizations claiming to use Scrum modify it to fit the context in which they are working. Schwaber [17] calls these changes *ScrumButs*. In this section we describe ScrumBut and organizational culture as a foundation for the discussion of why ScrumButs emerge.

2.1 ScrumBut

ScrumBut, as a phenomenon, is a prevalent occurrence when studying Scrum practices in professional software development as seen in for example [3], [8], [15], but only recently has a more elaborate study been made on the subject [4]. A ScrumBut can be written as **(ScrumBut)(Reason)(Workaround)**, and an example could be "(We use Scrum, but) (having a Daily Scrum every day is too much overhead,) (so we only have one per week)" [17].

Some argue that using ScrumButs is a sign of dysfunction [17], while others argue that you should do what works best for you [11, 12]. Conboy and Fitzgerald [2] argue that for a method to be agile, it should be amenable to change. Others again take a neutral standpoint [4]; we concur with the latter, as claiming ScrumButs to be universally good or bad requires evidence that does not exist.

2.2 Organizational Culture

Iivari and Iivari [9] hypothesize that a connection exists between organizational culture and the deployment of agile methodologies. ScrumButs exist as part of the deployment of Scrum, and according to the hypotheses a connection between ScrumButs and organizational culture exists. To be able to illustrate the connection, a brief introduction to organizational culture and the Competing Values Model (CVM) is necessary. CVM is based on the relationship between two dimensions: change vs. stability and internal focus vs. external focus [9]. Based on the two dimensions, four types of culture can be distinguished: Developmental (change and external focus), Consensual (change and internal focus), Rational (stability and external focus) and Hierarchical (stability and internal focus). The key values of the four cultures are shown in Table 1 (based on [14], [16]). Note that an organization can have values from several cultures. The values will be used to place the ScrumBut reasonings in a culture, putting our findings into perspective.

3 Research Approach

To answer the questions of which, how, and why ScrumButs are prevalent in software development, we performed a Grounded Theory study of existing empirical research (based on [30]). The literature study was supplemented with

Table 1. Overview of competing values in organizational culture [14], [16].

Aspect	Developmental	Consensual	Rational	Hierarchical
Organizational orientation	Flexibility, adaptability and readiness	Cohesion and morale	Productivity and efficiency	Stability and control
Organizational objectives	Growth and development	Group maintenance	Pursuit of objectives	Execution of regulations
Organizational structure	Complex tasks; Collaborative work groups	Complex tasks; Collaborative work groups	Complex tasks; Responsibilities based on expertise	Routine tasks and technology; Formal rules and policies
Base of Power	Values	Ability to cultivate relationships	Competence	Knowledge of organizational rules and procedures
Decision making	Organic, intuitive	Participatory, deliberative	Goal-centred, systematic and analytical	Top-down pronouncements
Leadership style	Idealistic, risk oriented, empowering	Team builder, concerned, supportive	Rational achiever, goal oriented	Dominance, conservative, cautious
Compliance	Commitment to values	Commitment to process	Contractual agreement	Monitoring and control
Evaluation of members	Intensity of effort	Quality of relationships	Level of productivity	Adherence to rules
Orientation to change	Change is embraced as part of growth	Open to change	Open to goal driven change	Resistant (orientated to maintaining the status quo)

data from in-depth laddering interviews of practitioners, based on [19], for a deeper understanding of ScrumBut as a phenomenon. This research method was selected due to its favorable properties towards investigating social processes, especially where previous research is lacking in depth, or where a new point of view appears promising.

Existing empirical research was selected from the four largest peer-reviewed software engineering digital libraries (ACM, IEEE, Scopus, WoS) to ensure quality and broadness. Included research had to pass the criteria of explicitly stating both workaround and reasoning for presented ScrumButs. Additional criteria were: research must explicitly state ScrumBut as a focus, and empirical evidence of ScrumButs must be from professional software development. Meeting these criteria were the 17 papers: [1], [3, 4, 5, 6, 7, 8], [10], [13], [15], [20], [22], [24, 25, 26, 27], [29].

As supplement, we conducted semi-structured interviews with subjects about their experienced ScrumButs and deployment of Scrum. Candidates were selected with an aim to maximize variation by seeking candidates from different types of companies with different roles and experience levels. The 9 interviewed candidates and general info about the interviews are listed in Table 2. Note that interview [I2A] and [I2B] were conducted in the same session, hence the shared index.

Table 2. Table of interviews.

Interview	Scrum Role	Org.	Team Size	Scrum Exp.	Interview Duration	Ref.
1	Scrum Master	α	6-7	3 years	1:32	[I1]
2A	Developer	β	7	13 years	1:38	[I2A]
2B	Project Manger	β	7	8 years	1:38	[I2B]
3	Developer	β	10	2 years	1:14	[I3]
4	Developer	γ	10	12 years	1:05	[I4]
5	Scrum Master	δ	4	<1 year	1:08	[I5]
6	Product Owner	ϵ	7-10	10 years	1:09	[I6]
7	Scrum Master	γ	10	11 years	1:18	[I7]
8	Developer & Scrum Master	θ	3	1 year	1:05	[I8]
9	Product Owner	ω	5-7	10 years	0:55	[I9]

Interviews were conducted with one of the authors as interviewer and another as note taker. During the interview, when a ScrumBut was mentioned, an elaboration or explanation was requested, until the interviewer was satisfied with the explanation or the candidate was unable to answer, thereby achieving richer explanations.

To ensure a common frame of reference for what constitutes a ScrumBut we base our definition of Scrum on "The Scrum Guide" [21] as it is concise and exact in its definition, and was written by the authors of Scrum, Schwaber and Sutherland.

4 Findings

4.1 Reasoning of ScrumButs

ScrumButs were identified in all Scrum practices, indicating that no Scrum practice works for everyone in every context. The identified ScrumButs and their reasoning are presented in Table 3 and Table 4. Each table is structured with the ScrumButs and references to where they emerge on the left, and the reasonings and their source on the right.

For example, in Table 3 it is shown that the ScrumBut *Development Team Not Making Estimations* emerges in [I2A], [4], and [29]. This constitutes a ScrumBut as "The Scrum Guide" [21] states that *"The Development Team is responsible for all estimates."* One reasoning for the ScrumBut is that *estimated product cost is needed in advance* which is explicated as:

Example 1. "Working practices of the company need to know an estimate in advance on how much the product will cost [so] a product manager or the Product Owner produces work estimates." [4, p. 200]

Table 3. ScrumButs and their reasonings (cont.)

ScrumBut	Reasoning
Daily Scrums Cancelled or Postponed [I2B, I8, I9, 3, 5, 8, 13, 15, 24, 25, 29]	Only done when Scrum Master needs updates [I2B]. Frequent meetings with externals [I9]. Team is working closely together [I8, 5, 25]. Difficult for team members to meet daily [8, 15, 29]. Difficult for team members to meet on time [13, 24]. Progress is not sufficient to warrant meeting [15]. The team is very small [8].
Retrospectives Cancelled or Postponed [I1, I2B, I4, I5, I6, I8, I9, 3, 5, 6, 8, 15, 26]	People, especially Product Owner, cannot attend [I4, 5]. Problems are solved as they appear [I8, I9, 5]. Not enough to talk about every Sprint [I1, I6, 8]. Time pressure from outside team (e.g. management) [15]. Team is too large [8]. Lack of feedback [8]. Issues handled through other meetings [8].
Sprint Review Cancelled or Postponed [5, 20]	Stong client relationship with frequent contact[5]. A *Weekly Meeting* makes Sprint Reviews unnecessary [20].
Sprint Planning Cancelled or Postponed [I2B, I4, I9, 20, 26]	No estimations, so planning is done on demand [20]. Sprints are replaced with Flow [I2B, I9]. People, especially Product Owner, cannot attend [I4].
Not Using Sprints [I2B, I9, 4, 5]	Project durations are very short [I2B]. Necessary to be very responsive to clients' needs [I9, 5].
Merging Events [I6, 3, 8, 15]	Separating the meetings feels unnecessary [I6]. Separate meetings take too much time [15]. No Retrospective planned; agenda added to Review [15].
Multiple Daily Scrums [8]	Development Team too big for only one meeting [8].
Sprint Review Without Stakeholders [I4, 3, 8, 13, 15]	Stakeholders do not have time to participate [I4]. Internal reviews by other teams [8]. Product Owner or other acts as stand-in [I4, 8]. Meeting too long and detailed [13].
Development Team Not Making Estimations [I2A, 4, 29]	Enough to include those with domain knowledge [I2A, 29]. It takes too much time [I2A]. Estimated product cost is needed in advance. [4]
No Estimations [4]	N/A
Not Defining Done [I5, I8, 3, 15, 25]	Done criteria are defined on a backlog item level [I5, 15]. Development not started; planned to be included later [I8].
No Increment [I1, 4, 25]	Separate test site test at end of Sprint [25]. Alternating dev./testing; Increment every other Sprint [4]. New features are released immediately [I1].
Not Everybody at Daily Scrum [13]	A combination of delay and unfocused discussion [13].
Extending Dur. of Daily Scrum [I2B, I9, 3, 8, 10, 13, 27]	Detailed discussions during meeting [I2B, I9, 3, 8, 13, 27]. Daily Scrum not held daily [3].
Extending Dur. of Sprint During the Sprint [4, 29]	Use of new technologies without prior training [29].
Extending Dur. of Sprint Planning [24]	Estimating tasks is complicated by inability to agree on required complexity [24].
Extending Dur. of Sprint Review [3]	N/A

Table 4. ScrumButs and their reasonings.

ScrumBut	Reasoning
Having a Product Owner Committee [I1, I8, 1, 3, 4, 6, 24]	Different stakeholders need authority [I1]. Decision makers do not want to give up mandate [I8]. The product has several branches [1, 24]. Multiple client relations [6].
Product Owner Not Accountable for Ordering of Product Backlog Items [I1]	Board of directors accountable for product [I1].
No Product Owner [I1, I2A, I2B, I6, 3, 4, 26]	Product Owner tasks handled by legacy roles [I2B]. Internal tasks can be handled by team lead [I6]. Redundant because of separate project mgm. office [I1, 3]. Requirements received directly from client [I2A, 3].
No Scrum Master [I2A, I8, 3, 5, 10, 26]	It is easier to engage the team without a Scrum Master [5]. Replaced by a Project Manager [10]. Extra resources for development gives more value [3]. Team does not use enough events [I8].
Scrum Master Not Protecting the Team [I1, 4]	N/A
Exceeding Team Size Boundaries [I1, I8, 3, 8]	Project does not require many resources [I1]. The company is very new [I8]. The team was large before Scrum was introduced [3].
Having Titles on Development Team [I1, I4, I6, 7, 26]	It makes personnel management easier [I1]. Architects are highly skilled and in high demand [I4]. Dictated by organization / part of people's identity [I6]. Easier for people to move between departments [I6].
Development Team Not Cross-functional [I4, 4]	Development may depend on legacy components [I4]. There is a separate testing team [4].
Management Changing Sprint Backlog During Sprint [I2B, 22]	Mgm. requires ad-hoc tasks to take priority [I2B, 22].
Mgm. Making Sprint Backlog [I3, 3]	Remains from old hierarchical development process [3].
Product Backlog Not Transparent [1, 15, 26]	Several Product Owners each maintain part of backlog [1].
Irrelevant Product Backlog Items Not Removed [I1, I6]	Responsible people lacks required knowledge [I1]. Too time-consuming; important items already first [I6].
Product Backlog Not Ordered [4, 15]	It takes to much time and effort [15, 4]. No Product Owner [4]. Product Owner does not have required knowledge [4].
No Product Backlog [4]	N/A
Stories Longer than Sprint Duration [I5, 25, 27]	Student developers only work part time [I5]. Some items require research/experimentation [27].
Team Not Self-organising [4]	N/A

4.2 Organizational Culture and ScrumBut

Utilizing the previous example, it can be shown that this ScrumBut is associated with a hierarchical culture through the common values of its reasoning and the values shown in Table 1. The reasoning in Example 1 shares the *organizational objective* of *execution of regulation*, and the *organizational structure* of *formal rules and policies* with the hierarchical culture. This association between organizational culture and ScrumBut can nuance the debate of whether ScrumButs are good or bad. The nuance lies in moving the attribution of a ScrumBut's nature,

from an inherent nature, to a nature dependent on its values and the values of its environment. An example of this can be demonstrated by the ScrumBut of *Development Team Not Making Estimations*: Instead of assuming its inherent nature as bad, looking at its reasoning and the values behind it, in association with the values of organizational cultures, shows its nature as more nuanced. It is shown that the reasoning for this ScrumBut (Example 1) is associated to the value-set of the hierarchical culture. Given this association it can be argued that this ScrumBut is benign, or even beneficial, in a company with strong hierarchical values, as the change is made to better support the organization's values. Thus, the nature of a ScrumBut might not be as simple as the binary judgment of being inherently good or bad.

Table 5. Cultures of ScrumButs.

ScrumBut	**Dev.**	**Con.**	**Rat.**	**Hie.**
Having a Product Owner Committee	[I8]		[6, 24]	[I1, 1]
Product Owner Not Accountable for Ordering of Product Backlog Items				[I1]
No Product Owner			[I6]	[I1, I2A, I2B, 3]
No Scrum Master		[I8, 5]	[3, 10]	
Exceeding Team Size Boundaries	[I8]			[I1, 3]
Having Titles on Development Team			[I4]	[I1, I6]
Development Team Not Cross-functional				[I4]
Daily Scrums Cancelled or Postponed	[I8, 13]	[5, 25]	[I9, 24, 29]	[I2B, 8, 15]
Retrospectives Cancelled or Postponed	[I8]	[I6, I9, 5]	[8]	[I4, 8, 15]
Sprint Review Cancelled or Postponed	[5]			
Sprint Planning Cancelled or Postponed			[I2B, I9]	[I4]
Replacing Sprints with Flow		[5]	[I9]	[I2B]
Merging Events	[I6]		[15]	
Multiple Daily Scrums				[8]
Sprint Review Without Stakeholders			[13]	[I4, 8]
Some Developers Not Attending Daily Scrum			[13]	
Extending Duration of Daily Scrum		[27]	[I2B, I9, 13]	[8]
Extending Duration of Sprint During Sprint			[29]	
Extending Duration of Sprint Planning			[24]	
Management Changing Sprint Backlog During Sprint				[I2B, 22]
Development Team Not Making Estimations			[I2A, 29]	[4]
Management Making Sprint Backlog				[3]
Product Backlog Not Transparent				[1]
Irrelevant Procuct Backlog Items Not Removed			[I6]	[I1]
Procuct Backlog Not Ordered			[15]	
Stories Longer than Sprint Duration	[27]	[I5]		
Not Defining Done	[I5, I8]		[15]	
No Increment			[I1]	[25]

5 Discussion

The contribution of this study is twofold: Firstly it extends the current knowledge of why ScrumButs emerge in practice, and secondly it shows the existence of a relationship between organizational culture and the reasoning behind ScrumButs.

Diebold et al. [3] present a general discussion of which changes practitioners make to Scrum, and suggest possible explanations. Eloranta et al. [4] take a more detailed look at the individual ScrumButs, presenting them as *Scrum anti-patterns* including listing consequences and company recommendations for working with them. This study extends current understanding of ScrumButs by providing an extensive overview of ScrumButs and their reasonings.

Iivari and Iivari [9] discuss a number of hypotheses indicating that organizational culture influences the deployment of agile methodologies. We show that an association can be made between ScrumBut reasonings and values pertaining to organizational culture, confirming that a connection exists.

5.1 Limitations

Organizational culture was found to influence the emergence of ScrumButs, but it may not be the only source of influence. Other sources of influence could be project budget, experience of developers, etc. As no other sources of influence are accounted for in this study, the findings may have been affected. It can, however, be argued that these factors impact organizational culture as well, reducing the need for accounting for them directly.

The candidates interviewed for the study were all situated within a relatively small area, as a result of which their social cultures may have been similar enough to influence the results. This limitation is partially neutralized by also studying literature with interviews from many different social cultures. In the data obtained from the literature, however, the explanations are some times not very thorough, often without evidence of the underlying cultural values.

5.2 Future Research

We have two suggestions for possible future research: First, we suggest conducting a study to quantify the connection between organizational culture and ScrumBut reasonings. Such a study could improve the understanding of how organizational culture affects the adaptation of Scrum and provide new insight for the discussion of whether ScrumButs are inherently good or bad.

Secondly, we propose an expansion of this study, but with a focus on the evolution of Scrum and ScrumButs over time in the same company, shedding light on how organizational culture influences the emergence of ScrumButs, and on how ScrumButs influence organizational culture.

6 Conclusion

ScrumButs were identified in all Scrum practices and their reasons ranged from precipitate, e.g. not ordering the Product Backlog because it takes too much time and effort [4, 15], to highly deliberate arguments, e.g. extending a Sprint after it starts to ensure there is enough time to experiment with new technologies [29]. For most of the identified ScrumButs one or more reasons were given to explain why the practice was changed. By holding each reason up against the values in the competing values model, it was found that most reasons can be linked to values of an organizational culture. The link between organizational culture and ScrumBut reasonings can help nuance the discussion of whether ScrumButs should be considered good or bad, by focusing on the underlying values of the reasoning and relating them to the organization's culture.

References

1. Block, M.: Evolving to Agile: A story of agile adoption at a small SaaS company. In: Agile Conference, pp. 234–239. IEEE (2011)
2. Conboy, K., Fitzgerald, B.: Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion. ACM Transactions on Software Engineering and Methodology 20(1), 2:1–2:30 (2010)
3. Diebold, P., Ostberg, J.-P., Wagner, S., Zendler, U.: What Do Practitioners Vary in Using Scrum? In: Lassenius, C., Dingsøyr, T., Paasivaara, M. (eds.) XP 2015. LNBIP, vol. 212, pp. 40–51. Springer International Publishing (2015)
4. Eloranta, V.-P., Koskimies, K., Mikkonen, T.: Exploring ScrumBut–An empirical study of Scrum anti-patterns. Information and Software Technology 74, 194–203 (2016)
5. Fernandes, C.: There and back again: From iterative to flow... and back to iterative! In: Agile Conference, pp. 103–110. IEEE (2012)
6. Heeager, L.T., Rose, J.: Optimising agile development practices for the maintenance operation: nine heuristics. Empirical Software Engineering 20(6), 1762–1784 (2015)
7. Hong, N., Yoo, J., Cha, S.: Customization of Scrum Methodology for Outsourced E-Commerce Projects. In: Asia Pacific Software Engineering Conference, pp. 310–315. IEEE (2010)
8. Hossain, E., Bannerman, P.L., Jeffery, R.: Towards an Understanding of Tailoring Scrum in Global Software Development: A Multi-case Study. In: Proceedings of the 2011 International Conference on Software and Systems Process, pp. 110–119. ACM, New York (2011)
9. Iivari, J., Iivari, N.: The relationship between organizational culture and the deployment of agile methods. Information and Software Technology 53(5), 509–520 (2011)

10. Inayat, I., Noor, M.A., Inayat, Z.: Successful Product-based Agile Software Development without Onsite Customer: An Industrial Case Study. International Journal of Software Engineering and its Applications 6(2), 1–14 (2012)
11. Jeffries, R.: Fractional Scrum, or "Scrum-But" (2013), http://agileatlas.org/articles/item/fractional-scrum-or-scrum-but
12. Kniberg, H.: What to do When Scrum Doesn't Work (2011) https://www.scrumalliance.org/community/articles/2011/february/what-to-do-when-scrum-doesn%E2%80%99t-work
13. Lorber, A.A., Mish, K.D.: How We Successfully Adapted Agile for a Research-Heavy Engineering Software Team. In: Agile Conference, pp. 156–163. IEEE (2013)
14. Ngwenyama, O., Nielsen, P.A.: Competing Values in Software Process Improvement: An Assumption Analysis of CMM From an Organizational Culture Perspective. IEEE Transactions on Engineering Management 50(1), 100–112 (2003)
15. Pauly, D., Michalik, B., Basten, D.: Do Daily Scrums Have to Take Place Each Day? A Case Study of Customized Scrum Principles at an E-commerce Company. In: 48th Hawaii International Conference on System Sciences, pp. 5074–5083. IEEE (2015)
16. Quinn, R.E., McGrath, M.R.: The Transformation of Organizational Cultures: A Competing Values Perspective. Organizational Culture, 315–334 (1985)
17. Schwaber, K.: ScrumButs and Modifying Scrum, https://www.scrum.org/scrumbut
18. Schwaber, K., Sutherland, J.: The History of Scrum, http://www.scrumguides.org/history.html
19. Shultze, A., Avital, M.: Designing interviews to generate rich data for information systems research. Information and Organization 21(1), 1–16 (2011)
20. Sienkiewicz, L.D., Maciaszek, L.A.: Adapting scrum for third party services and network organizations. In: Federated Conference on Computer Science and Information Systems, pp. 329–336. IEEE (2011)
21. Sutherland, J., Schwaber, K.: The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game (2013), http://scrumguides.org
22. Tanner, M., Mackinnon, A.: Sources of Disturbances Experienced During a Scrum Sprint. In: Proceedings of the 4th International Conference on IS Management and Evaluation, pp. 255-262. ACPI (2013)
23. Tiwana, A., Keil, M.: The one-minute risk assessment tool. Commun. ACM 47(11), 73–77 (2004)
24. Upender, B.: Staying agile in government software projects. In: Agile Development Conference, pp. 153–159. IEEE (2005)
25. Vallon, R., Drager, C., Zapletal, A., Grechenig, T.: Adapting to changes in a project's DNA: A descriptive case study on the effects of transforming agile single-site to distributed software development. In: Agile Conference, pp. 52–60. IEEE (2014)

26. Vallon, R., Strobl, S., Bernhart, M., Grechenig, T.: Inter-organizational co-development with scrum: experiences and lessons learned from a distributed corporate development environment. In: Baumeister, H., Barbara, W. (eds.) XP 2013. LNBIP, vol. 149, pp. 150–164. Springer, Heidelberg (2013)
27. Verdugo, J., Rodríguez, M., Piattini, M.: Using Agile Methods to Implement a Laboratory for Software Product Quality Evaluation. In: Cantone, G., Marchesi, M. (eds.) XP 2014. LNBIP, vol. 179, pp. 143–156. Springer International Publishing (2014)
28. VersionOne Inc. 10th Annual State of Agile Report (2015), http://stateofagile.versionone.com/
29. Vilain, P., Martins, A.J.B.: Neglecting Agile Principles and Practices: A Case Study. In: The 23rd International Conference on Software Engineering and Knowledge Engineering, pp. 596–601. (2011)
30. Wolfswinkel, J.F., Furtemueller, E., Wilderom, C.P.M.: Using grounded theory as a method for rigorously reviewing literature. European Journal of Information Systems 22(1), 45–55 (2013)