Artefacts in Agile Team Communication

Gerard Wagenaar

SIKS Dissertation Series No. 2019-18

The research reported in this dissertation has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Artefacts in Agile Team Communication

ISBN: 978-94-028-1543-6

Cover image by Wilma Wagenaar, KunstwerkMaatwerk; Reisefreiheit_eu, Pixabay
Layout and design by Eduard Boxem, persoonlijkproefschrift.nl
Printed by Ipskamp Printing, proefschriften.net

Artefacts in Agile Team Communication

Artefacten in Agile Team Communicatie
(met een samenvatting in het Nederlands)

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Universiteit Utrecht
op gezag van de rector magnificus, prof.dr. H.R.B.M. Kummeling,
ingevolge het besluit van het college voor promoties
in het openbaar te verdedigen op

maandag 1 juli 2019 des middags te 12.45 uur

door

Gerard Wagenaar
geboren op 29 maart 1960, te Purmerend

Promotoren:

prof. dr. S. Brinkkemper

prof. dr. ir. R.W. Helms

Co-promotor:

dr. S.J. Overbeek

Dit proefschrift werd mede mogelijk gemaakt door van Avans Hogeschool.

Preface

When I graduated in 1986 and started looking for a place to start my professional career, I was offered several opportunities to do so, despite the economical crisis of the early 1980s. Having studied for some years, I decided not to join the Dr. Neher laboratory of PTT, nowadays KPN, where pursuing a PhD would have been a natural consequence. Instead, I joined the Netherlands Organization for Applied Scientific Research (TNO) to become involved in practice, although from a scientific point of view. In the years afterwards practice gained influence over science.

In 1998 I started lecturing at Avans University of Applied Sciences and some time thereafter my interest in returning to science rose. However, for quite a number of years this interest remained rather in the background. The push I needed to move them to the foreground came when Avans University decided to adopt a stimulation programme for employees interested in obtaining a PhD. This was the moment.

I did some background research and as a result I approached Sjaak Brinkkemper from Utrecht University with some very preliminary ideas about doing some PhD research about some agile software development. Yes, I am afraid it was about as vague as that. Many thanks go to Sjaak, who could have been negative or uninterested at this point, but rather encouraged me to continue. He opened a door for me.

Many thanks also go to Remko Helms, who showed me the scientific hallway after the door. His guidance on the return path from practice to science, more specifically, drawing up a research proposal, was crucial. As a lecturer writing

in general was not unknown to me, but academic writing had faded away somewhat.

At the same time I am grateful to my collegues of Avans University who supported me in organizing and finalizing all necessary preparations. I want to mention Petra Koenders and Chris Otto, who, as the manegment team of the former Academy of ICT & Business (AIB), supported me wholeheartedly in this trajectory. Special thanks go to Nanske Wiltholt, who acted as intermediary, or troubleshooter, between Avans University and Utrecht University. I also include Avans' Executive Board for its support through the stimulation programme.

Remco Helms and Sietse Overbeek coached my during my PhD years. They are the ones who encouraged me, helped me, and saved me; every paper met their positive criticism. Sjaak, Remko and Sietse: Thank you for your share in writing six papers which are at the heart of this thesis. Being a lecturer at Avans University, I was used to providing feedback on student performance. Now I was on the other side, receiving your feedback.

I also want to thank my fellow PhD students I met over the years. Special mention goes to Garm Lucassen with whom I worked together in the course "Software Product Management"; results from this course provided my research with valuable base data.

In a broader sense, the results in this dissertation would not have been possible without the generous cooperation from all organizations, which were visited as part of this research. My special thanks go to all discussion partners who were interviewed as part of the research.

I would also express my gratitude to the graduate students who participated at various stages as members of a research group as component of their course 'Software Product Management'.

I would like to thank my colleagues at the Department of Information Science, who might have been under the impression that working on a PhD dissertation is an interaction between being absent and visiting conferences all over the world, but who were always were willing to fulfil tasks I could temporarily not participate in.

As a parent, some of this preface's final words go to my children, Abram and Miriam. They sometimes laughed at me, as the road towards this dissertation seemed to grow longer and longer. They blamed me for procrastination, perhaps because they recognized it so well. Well, I'm done, just the same as you two always did. I hope you will join me as paranimphs.

My very final words and most special thanks go to my wife Wilma. In first instance she had to get used to me doing a lot of the PhD research at home. Somewhere along the way she changed to perhaps wishing the research to go on eternally. That, unfortunately, will not be the case. Your support was invaluable and in a way this dissertation is as good yours as it is mine.

List of Tables

List of Figures

Acronyms

AACD	Agile Artefact Class Diagram, Agile Artefact Class Diagram
Agile Manifesto	Manifesto for Agile Software Development
ASD	Agile Software Development
CMM	Capability Maturity Model
CSP	Case Study Protocol
HAMRA	**H**ybrid **A**daptive **M**ethodology **R**eference **A**rchitecture
RUP	Rational Unified Process
SACD	Scrum Artefact Class Diagram
SAM	*Situational Assessment Method*
SPI	*Software Process Improvement*
TAM	Technology Assessment Model, Technology Assessment Model
VSO	*Visual Studio Online*

Contents

CHAPTER 1

Introduction

Since the publication of the Manifesto for Agile Software Development (Beck et al., 2001), or Agile Manifesto for short, in 2001, Agile Software Development (ASD) has evolved considerably, as is confirmed by recent surveys on the use of agile software development processes (*Agile is the new normal - Adopting Agile project management*, 2017; VersionOne, 2017). Although the Agile Manifesto certainly pushed ASD forward, its publication certainly was not the first step in ASD. An evolutionary map of agile methods marks the first influence on a path towards ASD already back in 1983 (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003). From this map it is also unequivocally clear, that ASD methods as DSDM (Stapleton, 1997), Scrum (Schwaber, 1997; Schwaber & Beedle, 2002), or XP (Beck, 1999) were proposed before the Agile Manifesto.

While the rise of ASD followed a gradual trajectory upwards, full waterfall software development followed a similar path downhill. However, ASD did not wipe it out, but rather integrated some of its elements in a Water-Scrum-Fall, or hybrid, approach (West, 2011). Moreover, an ongoing survey abbreviated as the Helena[1] survey, investigates what the current state of practice in software and systems development is and, in particular, which development approaches (traditional, agile, mainstream, or home-grown) are used in practice, how they are combined, and how such combinations were developed over time (Kuhrmann, Diebold, MacDonell, & Münch, 2017; Kuhrmann, Münch, Diebold, Linssen, & Prause, 2016; Kuhrmann, Münch, Tell, & Diebold, 2018; Theocharis, Kuhrmann, Münch, & Diebold, 2015). As phrased in (Kuhrmann, Diebold, et al., 2018): " *… hybrid development approaches: (i) have become reality for nearly all companies; (ii) are applied to specific projects even in the presence of company-wide*

1 Helena is an abbreviation for Hybrid dEveLopmENt Approaches in software systems development

policies for process usage; (iii) are neither planned nor designed but emerge from the evolution of different work practices; and , (iv) are consistently used regardless of company size or industry sector", where "*a hybrid development approach is any combination of agile and /or traditional (plan-driven or rich) approaches that an organizational unit adopts and customizes to its own context needs (e.g., application domain, culture, processes, project, organizational structure, techniques, technologies, etc.)*". In survey results from several countries and continents (Aymerich, Díaz-Oreiro, Guzmán, López, & Garbanzo, 2018; Felderer, Winkler, & Biffl, 2017; Klünder et al., 2017; Kuhrmann, Diebold, et al., 2018; Nakatumba-Nabende, Kanagwa, Hebig, Heldal, & Knauss, 2017; Paez, Fontdevila, & Oliveros, 2017; Scott, Pfahl, Hebig, Heldal, & Knauss, 2017; Tell, Pfeiffer, & Schultz, 2017), it was repeatedly shown that traditional and agile software development methods go well together. As an example, the Austrian survey (Felderer et al., 2017) concluded that traditional approaches, or combinations with agile practices, were favoured for architecture and design, configuration management, and risk management. Core agile approaches were used for integration and testing, change management, quality, and project management. Although other countries slightly deviate in the areas of application, the common denominator is clear: Contemporary software development may predominantly be agile, traditional approaches still play a major role. An advice given already shortly after the inception of the Agile Manifesto still seems to be valid: "*Although many … consider the agile and plan-driven software development methods polar opposites, synthesizing the two can provide developers with a comprehensive spectrum of tools and options*" (Boehm, 2002, *p.64*).

Blending traditional and agile software development methods may thus be considered a fact. However, stepping one level below, a question arises: What is being blended? In other words, which elements of traditional software development methods have survived in ASD? Some answers to this question

have been discussed at length, especially since the publication of the Agile Manifesto, whereas other have not. Among the former ones are architecture (Yang, Liang, & Avgeriou, 2016), using maturity models in ASD (Selleri Silva et al., 2015), and testing (Hellmann, Sharma, Ferreira, & Maurer, 2012). Other answers have been less prominently discussed, such as the use of documentation.

Use of architecture in ASD has been a topic of debate. How is the role of architecture in traditional software development transfered to ASD? In this respect it is interesting to note how research in this area is sometimes labelled, interrogatory in form:

- Agility and Architecture: Can They Coexist? (Abrahamsson, Babar, & Kruchten, 2010)
- Software Architecture and Agile Software Development - A Clash of Two Cultures? (Kruchten, 2010).

This format illustrates the tension in applying a traditional approach, architecture in this case, in an agile world. However, there is no disagreement about the fact they blend in practice. In a literature survey this was summarized as: "*… it is hard to conclude neither that agile and architecture is like oil and water, nor that the two are the perfect marriage*" (Breivold, Sundmark, Wallin, & Larsson, 2010, *p.36*). Despite the fact that no final answer is yet provided, blending traditional architectural approaches with ASD is a well-researched area (Yang et al., 2016) .

Another well-researched area is the use of maturity model in ASD, in particular the use of the Capability Maturity Model, CMM (Paulk, Curtis, Chrissis, & Weber, 1993). On the one hand incongruities are signalled: "*Even though agile methodologies may be compatible in principle with the discipline of models such as the CMM, the implementation of those methodologies must be aligned with the spirit of the*

agile philosophy and with the needs and interests of the customer and other stakeholders. Aligning the two in a government-contracting environment may be an insurmountable challenge" (Paulk, 2002, *p.16*). On the other hand several agility maturity models have been put forward, such as the Agile Maturity Model (AMM) as a software process improvement framework for ASD practices (Chetankumar & Ramachandran, 2009). AMM links the agile software development practices to maturity levels, in a CMM-like style. An overview of various such models is given in Schweigert, Vohwinkel, Korsaa, Nevalainen, and Biro (2013). The overview revealed lots of models, but they were found to be incomparable. Thus, although a final agility maturity model has not been identified, numerous suggestions have been put forward.

Blending traditional and agile software development methods in general was a fact already. However, about what is being blended and how, some topics, of which two examples were discussed above, are well-researched, whereas others are not. One of those is documentation; the use of traditional documents in ASD has not been an intensive topic of research.

Waterfall software development was and is often associated with an abundance of documentation, certainly in popular press (Lienitz, 2017; Maniuk, 2016). Although the waterfall model itself in general does not explicitly specifies documentation artefacts, the notion is widespread. In a historical tour, Ruparelia (2010) derives at least six distinct types of document to be produced according to the waterfall model: Requirements document, preliminary design specification, interface design specification, final design specification, and operations manual or instructions. Furthermore, if, for instance, RUP (Rational Unified Process) (Kruchten, 2000) is considered as a modern waterfall variant, some evidence for the proposition is again available, because of RUP's prescriptive

character of documentation. The same goes for SDM, short for System Development Methodology (Turner et al., 1987), a waterfall paradigm from the Netherlands, which lists a large number of (documentation) deliverables. The use of documentation, or (document) artefacts usage, in waterfall-like system development methods thus can be considered a fact.

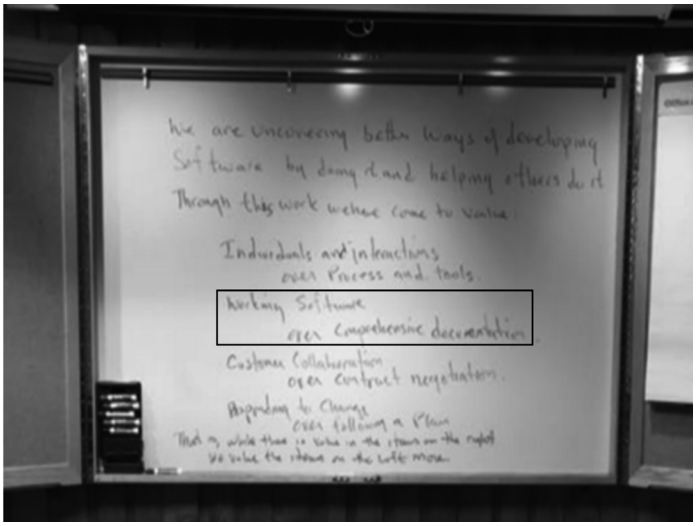

Figure 1.1 - Agile Manifesto

On the other hand ASD, as in the Agile Manifesto (Beck et al., 2001), is unequivocally: "*… we have come to value: … Working software over comprehensive documentation*", as illustrated in Figure 1.1 (adapted from (Prathan, 2018)). As a further specification one of its principles states: "*The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*". Nevertheless, it has been recognized that "*Since documentation is compromised when applying agile methods, important knowledge may be lost during and after system development*" (Rubin & Rubin, 2011, *p.117*). ASD overcomes documentation scarcity by relying on constant interaction between agile team members, as stated in the aforementioned principle. ASD does not

preclude the use of documentation, but rather, in comparison with traditional software processes, it is at least less document-oriented, as it restricts itself to comprehensive documentation. Therefore, in essence, it is possible to support documentation in agile development methods without compromising the Agile Manifesto. If documentation is adaptive and if the documentation supports people collaboration rather than replacing it, then documentation can be well-aligned with agile development principles (Rubin & Rubin, 2011). This line of reasoning is supported by including the right amount of documentation, as part of using agile software engineering techniques, as a critical success factor in agile software projects (Chow & Cao, 2008), although this effect was not confirmed in a later study (Stankovic, Nikolic, Djordjevic, & Cao, 2013).

The importance of collaboration, co-ordination and communication in agile teams is generally recognized. Sharp and Robinson (2010) defines them as follows:

- Collaboration takes place when two or more people are working together on a task.
- Co-ordination is the process of managing dependencies among activities.
- Communication takes place when two or more people exchange information or knowledge through verbal or non-verbal means.

They further notice that collaboration and co-ordination depend on communication, and communication – in one form or another – is central to successful software development (Cockburn, 2002; Coplien & Harrison, 2005). Understanding communication in ASD however is challenging given the agile discounting of comprehensive documentation and its valorisation of interactions,

but communication in ASD is crucial yet also tacit, informal, and predominantly verbal (Sharp & Robinson, 2010).

From another point of view four approaches to coordination were addressed in relation to challenges with large-scale agile development: A software engineering one, a sociological, an organizational psychological, and a management science one (Dingsøyr, Bjørnson, Moe, Rolland, & Seim, 2018). In a software engineering approach, a coordination strategy consists of three components: Synchronization, structure, and boundary spanning where the latter includes activities, artefacts, and roles to coordinate with other people or units beyond the project. This strategy puts emphasis on coordination with a team, and coordination using oral communication and physical artefacts. From a sociological viewpoint, main determinants for coordination are through persons or through artefacts, where the latter, impersonal coordination, is codified, for example through plans or written coding standards.

In summary, there certainly is a role for artefacts in the communication within agile teams. This is partially confirmed in other studies. Gröber (2013) investigated, in a systematic literature study, the use of artefacts in agile methods and drew up a class diagram (AACD) with nineteen artefacts and relations between them. However, this research was not based on empirical data. Bass (2016), in a study on large-scale offshore software development programmes, identified twenty-five artefacts. The context of the study was large-scale offshore ASD, which should be considered divergent from a collocated team. Both studies will be discussed in more detail in Chapter 2.

In a more ambitious endeavour, recent research explicitly posed the question: "*Which elements or components (agile or non-agile) are relevant for developing a context-*

aware hybrid adaptive methodology reference architecture?" (Gill, Henderson-Sellers, & Niazi, 2016, *p.316*), where hybrid refers to the integration of agile and non-agile elements. The resulting reference model, HAMRA (**H**ybrid **A**daptive **M**ethodology **R**eference **A**rchitecture), identifies work products with examples like backlog, document, and model (Gill et al., 2016). They certainly classify as artefacts. However, since HAMRA is a reference architecture, it does not further detail the work products, nor does it claim exhaustiveness in its work products.

To investigate the role of artefacts in agile team communication the overall research question was phrased as follows:

> ***What is the role of artefacts in the communication in teams involved in agile software development?***

An artefact is understood to be a tangible deliverable produced during software development, where tangible here means being easily seen or recognized rather than being restricted to only being touched or felt, thus including materials in both physical and electronic format.

The remainder of this introductory chapter is structured as follows. In Section 1.1 the global research question is operationalized into several more detailed research questions, each of which can be answered by an applicable research method. The research approach as well as respective research methods for sub questions are introduced in Section 1.2. The last part of this chapter, Section 1.3, outlines the structure of the remaining chapters of this dissertation.

1.1 Research Questions

Where the role of architecture or maturity models in ASD has been well-researched areas, the same is not true for the role of artefacts in ASD. Investigating

their role in the communication in agile teams then begins with examining which artefacts are being used by agile teams. A first set of artefacts originates straightforward from ASD methods themselves. Although an exhaustive list of such artefacts, originating from the methods themselves, has not been compiled, there is some consensus in this respect. Scrum explicitly lists artefacts, such as product and sprint backlog (Schwaber & Sutherland, 2013). User stories are generally considered to be included (Cohn, 2004). Additional artefacts arise, such as a definition of done or a burndown chart arise when applying ASD in practice (Deemer, Benefield, Larman, & Vodde, 2010).

Blending traditional and agile software development methods, as is current practice, then raises the issue which other artefacts are being incorporated in ASD and what characterizes them. A reference model like HAMRA, although developed based on a relevant question, does not yet address the level of detail, i.e. the artefacts involved, relevant for providing deeper insight into the blending of agile and non-agile artefacts.

The hypothesis underlying the first two research questions is that documentation has proven its value in traditional software development and that agile teams will at least carry over some of this value to their current practices. Therefore, introductory research questions were formulated as:

Research question A.1	**Which artefacts do agile teams produce?**
Research question A.2	**How can those artefacts be characterized to designate their use in agile team practices?**

Both research questions were operationalized in Scrum teams.

In daily practice agile teams extensively use tools to support their agile processes. However, little attention has been given to criteria teams apply in their selection of such a tool. Existing research either concerns individual experiences (Engum, Racheva, & Daneva, 2009; Møller, Nyboe, Jørgensen, & Broe, 2009; Uy & Rosendahl, 2008) or research explicitly addressing more general criteria, but in the latter case existing criteria were used rather than first defining criteria to select tools thereafter (Azizyan, Magarian, & Kajko-Mattson, 2011; Taheri & Sadjad, 2015).

Moreover, whether these criteria relate to the use of agile or non-agile artefacts or not, has not been the topic of extensive research. The research question in this respect was formulated as:

Research question A.3	**Which criteria do Scrum teams adopt when choosing computer-based support for their Scrum process?**

Having established the blend of traditional and agile artefacts that either are supported by tools or they are not, it is evident that agile teams do consider them both to be important. However, this does not shine light on rationales for using them. In a first holistic view it was investigated whether or not there is a relation between organizational factors, i.e. maturity of a team's software development process and its use of artefacts. Underlying the choice for maturity was the assumption that artefacts usage is a quality consideration, such as in the Capability Maturity Model Integration for Development (CMMI-DEV), where documentation artefacts, for instance architecture documentation and design data, are explicitly mentioned and contribute to achieving higher maturity levels (CMMI Product Team, 2010). Since this part of the research was done in organizations developing software products, CMMI-DEV was not the model of

choice, but a more dedicated maturity model was introduced instead, specifically aiming at software product management. The research question was:

Research question B.1	**To what extent does software product management maturity relate to the usage of artefacts in ASD?**

In a follow-up study, the holistic view was abandoned. To determine rationales for artefacts usage in agile teams, team members were approached and interviewed directly. Within nineteen agile teams twenty-five interviews were held with prominent team members to discuss the team's artefacts usage and their motivation for using them. The research question was phrased as:

Research question B.2	**What are rationales for agile teams to use artefacts?**

Formal and informal communication in ASD are still often regarded as two end points on a measuring scale, where as ASD nowadays includes both (Pikkarainen, Haikara, Salo, Abrahamsson, & Still, 2008). The two resemble the distinction between explicit and tacit knowledge (Nonaka, 1994; Polanyi & Sen, 2009) and where the distinction between those two is recognized not to be a black and white one, this raises the question whether or not agile teams also experience intermediate appearances between formal communication (artefacts) and informal communication (face-to-face). This appearance was coined a 'fuzzy' artefact. This is an artefact which is not formally documented, but one that is still explicitly recognized by an agile team. Its existence was investigated with the research question:

Research question C.1	**Which fuzzy artefacts do agile teams use in between formal communication with artefacts and informal face-to-face communication?**

Despite the support by artefacts, it still is imaginable that internal communication for some topics fails. Who do agile teams address when additional competencies are needed and how are they aware of them? Additional competencies might be sought for in external partners who, in an ad hoc fashion or perhaps even in a structural way, contribute to an agile team. The research questions were:

Research question C.2	**To whom do Scrum teams turn to for additional competencies and which competencies are involved?**
Research question C.3	**How are Scrum teams aware of what additional competencies are needed?**

The research questions, and their themes, are summarized in Figure 1.2.

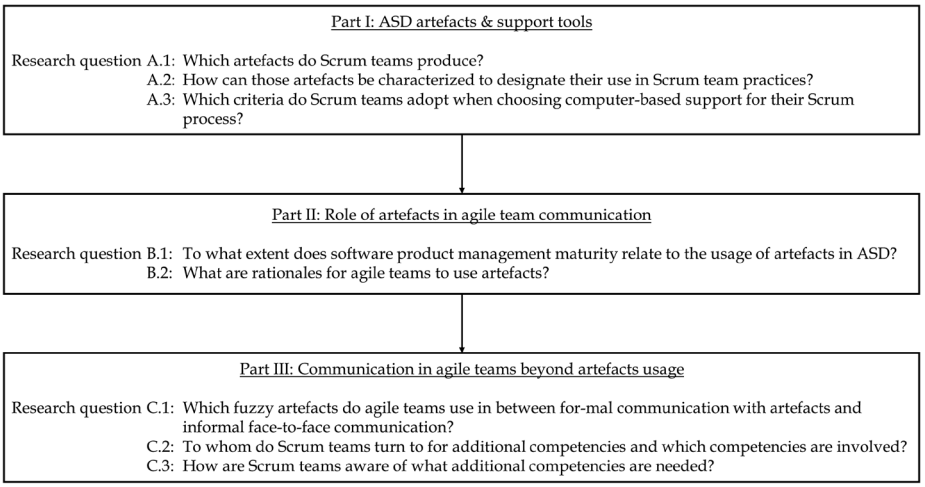

Figure 1.2 - Themes and research questions

1.2 Research Approach

A research approach contains plans and the procedures for research that span the steps from broad assumptions to detailed methods of data collection, analysis, and interpretation (Creswell & Creswell, 2017). In a research approach three components come together (Creswell & Creswell, 2017):

- A worldview is defined as *"a basic set of beliefs that guide action"* (Guba, 1990, *p. 17*).
- A research design is a type of inquiry within qualitative, quantitative, and mixed methods approaches that provides specific direction for its procedures.
- A research method involves the forms of data collection, analysis, and interpretation that researchers propose for their studies.

In this introductory chapter the current research will be positioned in the research approach's first two components, because they steer the research in this

dissertation in general. The third component, research methods involving the forms of data collection, analysis, and interpretation, will be discussed at length in the individual chapters to come, Chapters 2 – 7, in relation to the research questions (A.1-A.3, B.1-B.2, C.1-C.3). A summarizing table will be presented in Section 1.2.2.

1.2.1 Worldview for research

Within a research approach a worldview represent a fundamental choice. Creswell & Creswell (2017) distinguish four of them (Table 1.1).

Table 1.1 - Four worldviews for research approach

Post positivism Determination Reductionism Empirical observation and measurement Theory verification	***Constructivism*** ***Understanding*** ***Multiple participant meanings*** ***Social and historical construction*** ***Theory generation***
Transformative Political Power and justice oriented Collaborative Change-oriented	***Pragmatism*** ***Consequences of actions*** ***Problem-centred*** ***Pluralistic*** ***Real-world practice oriented***

Among others, knowledge developed through a post positivist lens is based on careful observation and measurement of the objective reality that exists 'out there' in the world; developing numeric measures of observations and studying the behaviour of individuals becomes paramount for a post positivist (Creswell & Creswell, 2017). This is not a worldview that easily aligns with the main research question nor its derived research questions. The research questions will not be answered numerically and the interest is in (agile) teams rather than individuals.

A transformative worldview holds that research inquiry needs to be intertwined with politics and a political change agenda to confront social oppression at whatever levels it occurs (Mertens, 2010). This, again, is not a worldview that fits the research questions.

Moreover, both the constructive and the pragmatic worldview tend to use qualitative research and the research questions are oriented to qualitative rather than quantitative research.

In the constructive worldview research relies as much as possible on the participants' views of the situation being studied (Creswell & Creswell, 2017). Researchers in a constructive approach inductively develop a theory of meaning, rather than starting with a theory, and as the main research question cannot be based on extensive previous research, its aim is theory building rather than theory testing. Constructive researchers are often concerned with the processes of interaction between individuals, which in the current research compares to artefact usage in agile teams.

Pragmatism as a worldview arises out of actions, situations, and consequences and has a concern with solutions where, instead of focusing on methods, researchers emphasize the research problem and use all approaches available to understand the problem (Creswell & Creswell, 2017). The dominant worldview in empirical software engineering is pragmatism and often multiple methods (qualitative or quantitative) of the other worldviews are used (Petersen & Gencel, 2013).

Given the characteristics of the current research, it is best positioned within the constructive worldview. In addition, a minor part of the pragmatic worldview

is to be noticed, especially where a real-world practice orientation is involved as the current research has its domain in the practices of agile teams.

2.2.2 Research Design

A research design provides specific directions for procedures within the realm of qualitative, quantitative, and mixed methods approaches (Creswell & Creswell, 2017):

- Qualitative research explores and understands the meaning individuals or groups ascribe to a human or social problem. The process of research involves emerging questions and procedures, data typically collected in the participant's setting, data analysis inductively building from particulars to general themes, and the researcher making interpretation of the meaning of the data.
- Quantitative research tests objective theories by examining the relationship between variables. These variables, in turn, can be measured, so that numbered data can be analysed using statistical procedures,
- Mixed methods research is an approach to inquiry involving collecting both quantitative and qualitative data, integrating the two, and using distinct designs.

In this tripartite classification mixed methods research are often referred to as the third methodological movement, with quantitative and qualitative methods representing the first and second movements respectively. Mixed methods research designs have suggested to be potentially superior to single method designs, but equally there has been intense debate regarding whether or not it is even appropriate to combine multiple methods that are often based on radically different paradigmatic assumptions (Venkatesh, Brown, & Bala, 2013).

An overview of their characteristics is summarized in Table 1.2 (Creswell & Creswell, 2017).

Table 1.2 - Alternative research designs

Research design		
Quantitative	**Qualitative**	**Mixed Methods**
• Experimental designs • Nonexperimental designs, such as surveys	• Narrative research • Phenomenology • Grounded theory • Ethnographies • Case study	• Convergent • Explanatory sequential • Exploratory sequential • Transformative, embedded, or multiphase

A constructive worldview does barely concord with a qualitative research design, especially because of its lack of an underlying existing theory. This is the reason the current research primarily uses qualitative research designs in which both case studies and grounded theory play a prominent role. Only for answering research question A.3 and B.1 (refer to Section 1.1) some minor quantitative research was applied, especially statistical analysis. Relating to the research questions stated in Section 1.1, Table 1.3 summarizes the research methods used for the various questions.

Table 1.3 - Research methods

Research question	Research method
A.1 / A.2	Exploratory comparative case study
A3	• Case study research (greenfield approach) • Grounded theory
B.1	Multiple case study
B.2	Multiple case study
C.1	Multiple case study
C.2 / C.3	Exploratory comparative case study

Research question A1 & A2 used an exploratory comparative case study with as unit of analysis one iteration of software development in a co-located team. The

goal of this case study was to collect evidence to revise and/or expand elements of a hypothesized preliminary model. The research method was exploratory to allow for rich evidence from which elements and relationships in the model could be reasoned about. This approach is an accustomed way to investigate phenomena in a context where events cannot be controlled and where the focus is on contemporary events (Yin, 2013).

For research question A.3 a greenfield approach was used, in analogy with the greenfield project: "*In software engineering jargon, a greenfield is a project which lacks any constraints imposed by prior work*" (Gupta, 2011, *p.21*). This approach provided a relatively unbiased way to explore features a Scrum team thinks of as being important in its support and it assured that prior organizational or commercial influences were excluded. Using student teams, the research may be characterized as a variant of case study research with the teams being the cases, but explicitly instructed through assignments. Data was analysed according to Grounded Theory, a systematic research method known for the generation of theory derived from systematic and rigorous analysis of data (Glaser, 1992; Glaser & Strauss, 1967). To analyse data further, three abstraction levels were used: category, concept, code (Allan, 2003). For the highest level, category, the Technology Assessment Model, TAM (Davis, 1989; Davis, Bagozzi, & Warshaw, 1989) was adopted. Applying open and axial coding (Corbin & Strauss, 2015), next level concepts were identified.

To investigate research question B.1, a multiple case study was used. Data collection took place through single-site case studies following widely accepted guidelines for case studies (Yin, 2013). Basic data on artefacts and maturity was collected on basis of a protocol including: (1) Software product management theory and research, (2) interview instructions, and (3) reporting guidelines.

A multiple-case study was used for research question B.2 with as unit of analysis an agile team and its artefacts. Since research results were scarce with regard to our research question, the aim was theory building rather than theory testing; the former is useful in early stages of research on a topic or when a fresh perspective is needed, while the latter is useful in later stages of knowledge (Eisenhardt, 1989). Data collection took place through the use of various single-site case studies.

A multiple single-case study was also used for research question C.1. This case study was executed in analogy with the previous one, but its unit of analysis was slightly different, again an agile team, but now with its concepts between tacit and explicit knowledge.

To investigate the cross-functionality of teams in research questions C.2 & C.3, the research analysed the communication of teams outside their team boundaries. In this way a team's competencies could be identified: Which ones do they require and on which basis? To allow for rich evidence we used an exploratory comparative case study approach as our research method with as unit of analysis one sprint of agile software development. The primary data collection method was semi-structured interviewing of team members. Our questionnaire addressed communication both between team members, exemplified in the use of agile practices, as well as communication between team members and non-team members. Furthermore, available documents and/or the contents of information systems was inspected.

A more detailed description of research methods, including forms of data collection, analysis, and interpretation, is given in the Chapters 2 – 7, in relation to the research questions.

1.3 Dissertation Outline

This dissertation is, after the introduction, divided in three main parts, each bundling a set of coherent research questions (refer to Section 1.1).

Part I focuses on the existence of artefacts and their computer-based support as worded in research questions A.1 – A.3. Chapter 2 contains research questions A.1 & A.2; Chapter 3 research question A.3.

Chapter 2 is published as:

Artefacts in Agile Software Development

Gerard Wagenaar, Remko Helms, Daniela Damian, Sjaak Brinkkemper

In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (eds.) Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015) Bolzano, Italy, 2 - 4 December 2015, 133 – 148 (Springer International Publishing)

2015

Chapter 3 is published as:

Describing Criteria for Selecting a Scrum Tool Using the Technology Acceptance Model

Gerard Wagenaar,Sietse Overbeek, Remko Helms

In: Nguyen N., Tojo S., Nguyen L., Trawiński B. (eds.) Proceedings of the 9th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2017), Kanazawa, Japan, 3 - 5 April 2017, 811 – 821 (Springer Cham)

2017

Part II comprises 2 chapters which contain research questions regarding the role of artefacts in the communication in agile teams, research questions B.1 – B.2. In Chapter 4 an influencing organization factor, maturity, is considered. Chapter 5 deals with rationales.

Chapter 4 is published as:

Influence of Software Product Management Maturity on Usage of Artefacts in Agile Software Development

Gerard Wagenaar, Sietse Overbeek, Garm Lucassen, Sjaak Brinkkemper, Kurt Schneider

In: *Felderer M., Méndez Fernández D., Turhan B., Kalinowski M., Sarro F., Winkler D. (eds.)*
Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbruck, Austria, 29 November – 1 December 2017, 19 – 27 (Springer, Cham)

2017

Chapter 5 is published as:

Working software over comprehensive documentation – Rationales of agile teams for artefacts usage

Gerard Wagenaar, Sietse Overbeek, Garm Lucassen, Sjaak Brinkkemper, Kurt Schneider

In: *Journal of Software Engineering Research and Development 6.1 : 7*

2018

Part III contains again two chapters, which have in common research questions (C.1 – C.3) that are related to communication in agile teams beyond the use of artefacts. In Chapter 6 describes the use of informal (fuzzy) artefacts; Chapter 7 identifies external partners who join their expertise with that of the agile team.

Chapter 6 is published as:

Fuzzy Artefacts: Formality of Communication in Agile Teams

Gerard Wagenaar, Sietse Overbeek, Sjaak Brinkkemper

Accepted for: *International Conference on the Quality of Information and Communications Technology (QUATIC 2018), Coimbra, Portugal, 4 – 7 September 2018*

2018

Chapter 7 is published as:

Competencies outside Agile Teams' Borders: The Extended Scrum Team

Gerard Wagenaar, Sietse Overbeek, Remko Helms

In: *Position Papers of the Federated Conference on Computer Science and Information Systems (FedSYS), 36th IEEE Software Engineering Workshop (SEW-36), Gdansk, Poland, 11 - 14 September 2016, 291 - 298*

2016

Chapter 8 concludes this dissertation and provides answers to the research questions as well as the main research question based on the results presented in previous chapters. In addition, an overview of relevant findings and contributions is provided, and the implications and limitations thereof are discussed. Finally, directions for future work are indicated.

As a summary, Figure 1.3 displays the outline of this dissertation.

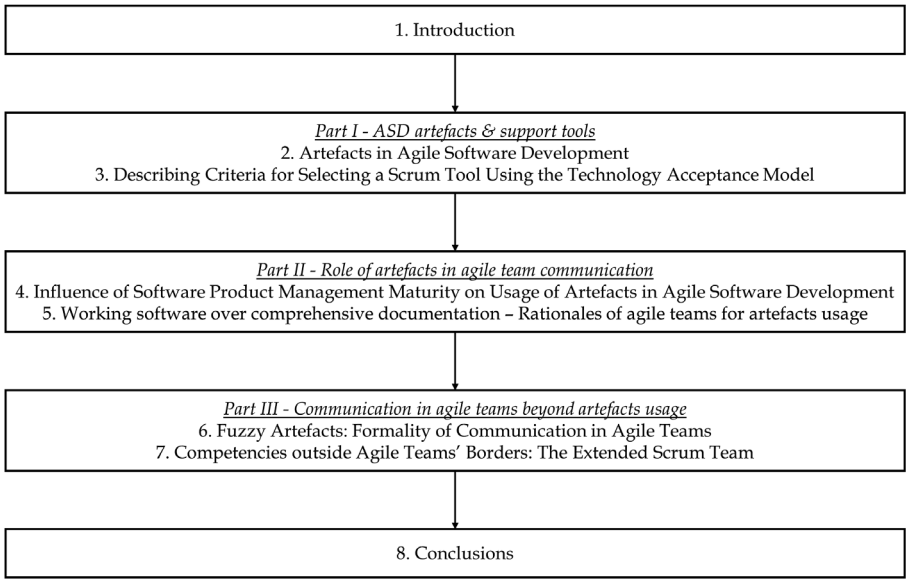

Figure 1.3 - Outline of dissertation

PART I

ASD artefacts & support tools

CHAPTER 2

Artefacts in agile software development

Agile software development methods prefer limited use of artefacts. On the basis of existing artefact models and results from three case studies we present a Scrum artefact model. In this model we notice teams using Scrum artefacts, but they, in addition, decided to produce various non-Scrum artefacts, most notably design documents, test plans and user or release related materials.

This work was originally published as:

Wagenaar, G., Helms, R., Damian, D., Brinkkemper, S.: Artefacts in Agile Software Development. In: Abrahamsson, P., Corral, L., Oivo, M., Russo, B. (eds.) Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015), Bolzano, Italy, 2-4 December 2015, 133–148. Springer International Publishing (2015)

2.1 Introduction

The application of an agile software development method, for instance XP or Scrum, is very common nowadays in delivering state-of-the-art software (Bustard, Wilkie, & Greer, 2013; Rodríguez, Markkula, Oivo, & Turula, 2012). All agile methods have in common a profound focus on communication, as articulated in one of the principles in the Agile Manifesto: "*The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*" (Beck et al., 2001). This emphasis on communication in software development does not come as a surprise. A field study on software design process for large systems (Curtis, Krasner, & Iscoe, 1988) already indicated that developing large software systems must be treated, at least in part, as a learning, communication, and negotiation process.

One form of communication is documentation through the use of artefacts. Most agile methods do not have artefacts as method of choice in communicating. Explicitly documenting (design) issues which are of little or no value to customers is not encouraged in agile thinking, but it is certainly not prohibited to produce and use artefacts. Citing another agile principle, "*The best architectures, requirements, and designs emerge from self-organizing teams*" (Beck et al., 2001), the use of artefacts is situational and a choice made by the team, dependent on the value it attaches to them. One agile method, DSDM (Stapleton, 1997), even defines deliverables which bear a great similarity to artefacts.

But the use of artefacts certainly has to be considered in agile methods, if only because face-to-face communication is simply not always feasible. In global software development direct face-to-face conversation is not possible and other mechanisms to support communication have to be applied (Herbsleb & Moitra, 2001). Knowing of such circumstances, agile artefacts have to be considered.

In our research we investigated the existence of artefacts as well as their use in three agile software development teams. We have focussed on Scrum teams, which is in fact not a major restriction, because Scrum certainly is an agile software development method (Abrahamsson, Oza, & Siponen, 2010; Abrahamsson et al., 2003; Glaiel, Moulton, & Madnick, 2013). Based on evidence from practice we will show for three Scrum teams their artefacts and we will provide a typology for their use. With this research we will on the one the hand provide another building block in the theory of artefacts within agile software development and on the other hand provide practitioners with a reference to mirror their own way of working relative to the context of use in the companies we studied.

The remainder of this paper is organised as follows. In the next section we will outline the theoretical background motivating our work. Then we will present our research method. The results will be presented for three case studies, followed by a discussion. A final section presents our main conclusions.

2.2 Theoretical Background

In this section we will first discuss artefacts and their use as a communication mean in (agile) software development. We will then describe existing artefact models to develop a preliminary Scrum artefact model thereafter.

2.2.1 Artefacts in Agile Software Development

A general description of an artefact is an object made by humans. In software development many definitions have been used, for example: "*A software artefact is understood to be a deliverable or an outcome of a software process*" (Georgiadou, 2003), "*An artefact is a deliverable that is produced, modified, or used by a sequence of tasks that have value to a role*" (Méndez Fernández, Penzenstadler, Kuhrmann, & Broy, 2010). For our purpose we define:

An artefact is a tangible deliverable produced during software development.

Tangible here means being easily seen or recognized rather than being restricted to only being touched or felt, thus including materials in both physical and electronic format.

Artefacts function as a mean of communication in software development (Curtis et al., 1988), but most agile methods do not have artefacts as method of choice in communicating. Their importance is recognized nevertheless and the agile approach to artefacts may be compared with "travelling light": "*Create just enough models and documentation to get by*" (Ambler, 2002, *p.29*). To emphasize their significance it has been shown that agile software development practitioners indeed perceive their internal documentation as important, whilst at the same time acknowledging that too little of it was available (Stettina & Heijstek, 2011). This result was partially confirmed in a case study on the impact of agile principles and practices on large-scale software development projects (Lagerberg & Skude, 2013).

In a case study on knowledge management usage in agile software projects (Yanzer Cabral et al., 2009) it was seen that even outdated and inadequate documentation was used, because it provided a context for knowledge sought for. And it was used even instead of face-to-face communication: "*Despite the documentation be reduced and outdated the team uses [it] as a source of knowledge to … reduce the direct communication*" (Yanzer Cabral et al., 2009, *p.634*).

We conclude that 'working agile' and 'using artefacts' are no contradictory terms.

2.2.2 Artefact Models

As starting point for a Scrum artefact model we take the Scrum process framework, which distinguishes artefacts, people and events (Schwaber & Beedle, 2002; Schwaber & Sutherland, 2013). Core Scrum artefacts are the Product Backlog, the Sprint Backlog and the Potentially Shippable Product Increment (or Increment for short). The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the future software product. The Sprint Backlog contains a subset of Product Backlog items selected for one Sprint. The Increment is the sum of all the Product Backlog items completed during a Sprint. Sometimes supplementary artefacts are associated with Scrum, although they are not labelled as such in Scrum's process framework. These are the 'Definition of Done' which is a set of acceptance criteria, and the 'Burn down chart' which is a visualization of progress.

Existing research on artefact models had its foundation in modelling artefacts from a theoretical point of view to confront the model with some experiences from practice thereafter. As a first example, a theoretical Scrum Process-Deliverable Diagram[2] (Blijleven, 2012) served as foundation for a Scrum Artefact Class Diagram (SACD) (Dijkstra, 2013). Using experiences from one project, the main components of the SACD became a Product Backlog, a Sprint Plan and a Build / Release (including a Build History). These are easily recognizable as the core Scrum artefacts model. In addition the model includes:

- Project management information, including a project initiation document (budget, resources) and a risk list.

2 A Process-Deliverable Diagram describes both processes and deliverables in one diagram (Weerd & Brinkkemper, 2008); the interpretation of deliverable in a PDD is analogous to our artefact.

- Architectural information, consisting of a domain model and a system architecture.
- Release-related materials, such as installation and maintenance guide, training materials.

As a second example a systematic literature study investigated the use of artefacts in agile methods and resulted in the development of an Agile Artefact Class Diagram (AACD) with 19 artefacts and relations between them (Gröber, 2013). This diagram was subsequently extended to a more generic model to abstract it from local, project-specific processes (Figure 2.1) and process interfaces were added to support the model's use in distributed project settings, where a process interface is a mean to exchange data between software processes (Kuhrmann, Méndez Fernández, & Gröber, 2013). On the basis of experiences from practice the diagram was further refined (Femmer, Kuhrmann, Stimmer, & Junge, 2014).

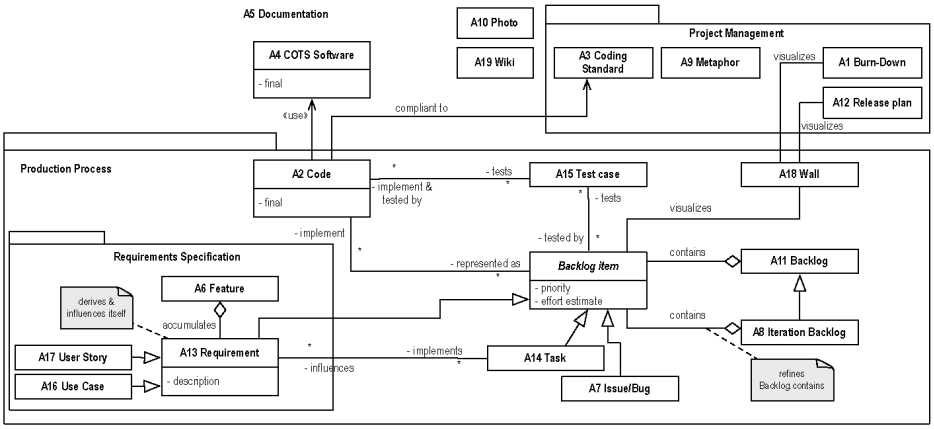

Figure 2.1 - Agile artefact class diagram (Kuhrmann et al., 2013)

Although the diagram was not confronted with descriptions of agile methods themselves nor explicitly based on experimental data (with exception of later refinements), the AACD provides an overview of agile artefacts.

Both SACD and AACD contain agile and/or Scrum artefacts. But they also contain other artefacts, which are similar to an approach to software development which advocated program design first and documenting it thoroughly, using such artefacts as a design and a test document (Royce, 1970).

2.2.3 Use of Artefacts

The AACD also launches a typology with its clustering of artefacts to packages, such as 'Production process' or 'Project management' (Figure 2.1). A well-known distinction separates process from product artefacts (Sommerville, 2001). The former are used in the process of development (for instance project plans, risk assessments, schedules); the package 'Project management' in the AACD is an example. The latter supports the product that is being developed (for instance a requirements document or a user guide). We will make use of these categories to characterize agile artefacts.

But product and process artefacts do not suffice. In a later version of the AACD tools are introduced (Kuhrmann et al., 2013). Other research also indicated the use of tools by agile teams, for instance story cards and the Wall (Sharp & Robinson, 2010; Sharp, Robinson, & Petre, 2009). In fact, using Scrum leads to an absolute need for tools (Hossain & Babar, 2009). This need, found in the context of global software development, includes tools for communication and collaboration in general, but also, more specifically, to support project management, backlog management and visualization. Examples of such tools are wikis, electronic workspaces, whiteboards, et cetera.

Neither the physical artefacts (story cards, Wall) nor the tools adhere to our definition of artefact. This is a major reason to introduce a third category: Supporting tools. These are not tangible deliverables produced during software

development, but they do support their production. In this line of thought a whiteboard (the wall) is a support tool, supporting an artefact like the Sprint Backlog. We do however, in the context of our current research, only include tools specifically aimed at supporting Scrum product or process artefacts, thus excluding some more general tools as, for instance, e-mail or a text processor. The SACD does not explicitly address supporting tools as a category, whereas the AACD does.

2.2.4 Towards a Scrum Artefact Model

When reflecting upon the core Scrum artefacts and the two class diagrams there is similarity up to a certain level. All address the core Scrum artefacts, although not always by the same name. But both diagrams reveal other, additional, artefacts. Combining them the contours of a Scrum artefact model arises (Figure 2.2).

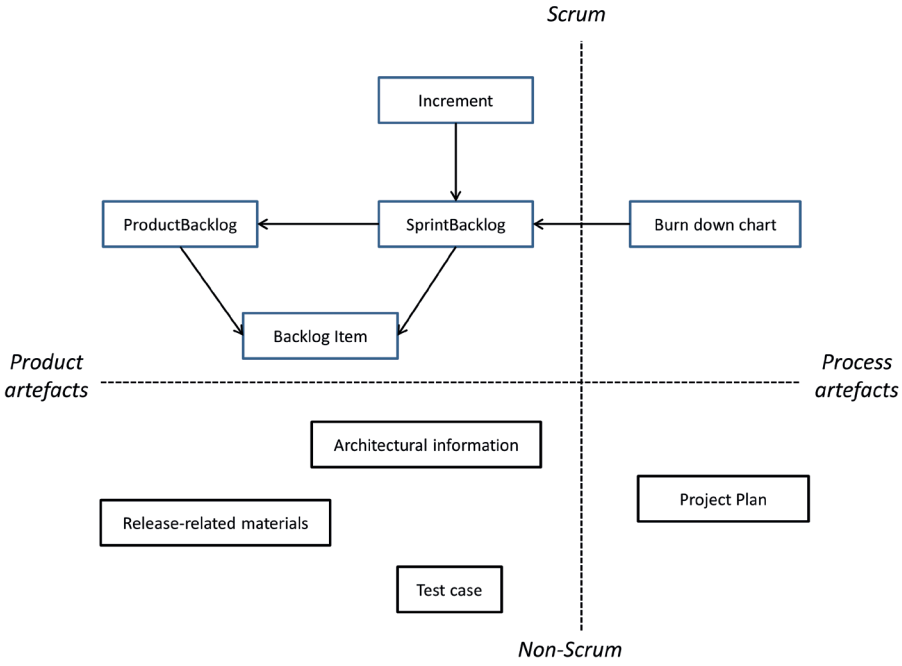

Figure 2.2 - Preliminary Scrum artefact model

In the upper half of Figure 2.2 our preliminary model contains the core Scrum artefacts, supplemented with a Burn down chart. The situation for the non-Scrum artefacts is less clear. The two existing models differ from each other and for the moment we include in our preliminary model their non-Scrum artefacts, but they are by no means meant to be complete; they serve as example for future refinements, based on empirical evidence. In the composition of the model we mapped, and summarized, elements from the diagrams. For instance, project plan represents both the 'Project management' package from the AACD as well as the project management information from the SACD. Relations for the non-Scrum artefacts (lower half of Figure 2.2) are deliberately left out, because there, again, is a need for validation.

2.3 Research Method

We are interested in artefacts in agile software development. Limiting ourselves to Scrum as the agile development method of choice, our goal is to collect evidence to revise and/or expand the elements of the preliminary model (Figure 2.2). Our research method should be exploratory to allow for rich evidence from which elements and relationships in the model can be reasoned about. We selected an exploratory comparative case study approach as our method with as unit of analysis one iteration of Scrum software development in a co-located team. This approach is an accustomed way to investigate phenomena in a context where events cannot be controlled and where the focus is on contemporary events (Yin, 2013). For the case study we operationalized our goal into the following research questions:

RQ1 Which artefacts do Scrum teams produce?

RQ2 How can those artefacts be characterized to designate their use in Scrum team practices?

A case study protocol was drawn up to guide the case study (Maimbo, 2005). According to the protocol organizations were required to use an agile software development method with a team of at least 5 members. We found 3 organizations willing to participate in our research, all having Scrum with a team size between 5 and 10 members. The organizations will be named Controller, Sunflower and Local for reasons of confidentiality.

The primary data collection method was semi-structured interviewing of team members, accompanied by examination of documents and/or information systems. A questionnaire with both level 1 and level 2 questions was drawn up to

guide the interviews (Yin, 2013). The interviews focussed on 2 topics, apart from some questions on the interviewee's background. The first topic was an overview of the agile practices within the team and the second one highlighted (the use of) artefacts. Representatives of the team were interviewed for approximately 1 hour each. Interviews comprised in total approximately 12 hours with 13 interviewees. Additionally 6 more interviews were held either before the series of interviews started to provide some general context, or afterwards, to clarify some remaining issues. Examples of artefacts, when available, were studied; they included, among others, contents of information systems and (design and test) artefacts.

Thirteen interviews were transcribed and coded. For all interviews the interviewee was provided with a summary for his or her approval. For each organization the results of the interviews and additional material was bundled in a case study report.

2.3.1 Validity

In terms of validity of our research method four criteria are widely used: construct validity, internal validity, external validity and reliability (Yin, 2013).

Construct validity identifies correct operational measures for the concepts being studied. To enhance construct validity (1) key informants should review draft case study reports, (2) multiple sources of evidence should be used and (3) a chain of evidence should be established (Yin, 2013). All 3 components have been applied. Key informants did comment on a draft case study report, various team members were involved to complement viewpoints and there is a direct link

from interviews (and other material) to conclusions by the use of the qualitative data analysis tool Nvivo[3].

Internal validity, although not obligatory for descriptive or exploratory studies, seeks to establish a causal relationship, where conditions are believed to lead to other conditions. Pattern matching and explanation building, rival explanations and logic models are constructs to promote internal validity (Yin, 2013). Consistently coding material contributes to pattern matching, and by this, since the case studies are exploratory in nature, internal validity is sufficiently addressed.

External validity defines the domain to which a case study's findings can be generalized. The use of replication logic is listed as the main tactic to guarantee this validity (Yin, 2013). Using a multiple-case study with a replication design on the basis of a questionnaire contributes to external validity, and thus to generalizability of results.

Reliability should demonstrate that the study can be repeated. The use of a case study protocol and the development of a case study database (Yin, 2013) were both applied in our study to increase reliability.

2.4 Results

The results from the case study will be described next, each organization in turn. We will first give an impression of (the structure of) each organization and then list the core Scrum artefacts they use. We will then describe the non-Scrum

3 NVivo is a software package to aid qualitative data analysis designed by QSR (http://www.qsrinternational.com).

artefacts and categorize them according to our different usages: Product artefact, process artefact, supporting tool.

2.4.1 Organizations

We will start with a short summary describing some key characteristics of the organizations / teams (Table 2.1). Most numbers are indicative; the size of the Scrum team is not.

Table 2.1 - Key characteristics case study organizations

Organization	Size		Installed base	Number of users
	Company	Scrum team		
Controller	120	5	1500	n/a[4]
Sunflower	15	6	125	750
Local	-	10	50	1000

The organizations all deliver product software (Xu & Brinkkemper, 2007). The installed base refers to the number of software packages in use with different customers (companies). Since one installation of a package may have one or more users, the number of users is significantly higher than the installed base. Since Local is a business unit of a larger organization, the number of employees of Local itself is difficult to establish because of the connections with its parent company.

Organization Controller is a company, providing the branch of management of (technical) objects, such as fleet management, with a software solution. Software development within Controller takes place in the Development department with circa 35 employees, half of them operating in 2 to 3 Scrum-teams with 5 – 6 members each. The other employees perform supporting tasks, from providing

4 Controller has no direct insight in the number of users.

architectural building blocks of source code to database management. The composition of the team under study is: 1 Product Owner, 2 developers and 2 testers, one of whom functions as Scrum Master. A Sprint within Controller takes 2 weeks, occasionally 3 weeks.

Organization Sunflower is a small Dutch company with around 15 employees. It provides floral industry (trading of flowers, plants and bulbs) with its Enterprise Resource Planning product. The product is developed and maintained by a Scrum team of six employees: 3 developers and 3 consultants. One additional, junior, developer assists the team in the background. A Sprint within Sunflower takes one month.

Organization Local is a business unit of a larger organization and develops product software for the public sector, particularly local government. It has a range of products, among which is a taxing application. This product is developed by a Scrum-team of 10 persons: a Product Owner, a Scrum Master, 2 designers, 4 developers and 2 testers. A Sprint within Local takes 2 weeks, occasionally 3.

2.4.2 Scrum Artefacts

All teams use all core Scrum artefacts. Product Backlog, Sprint Backlog and Backlog Items are all registered with a supporting tool, whether commercially available (Scrumworks[5], Controller) or developed within the organization itself. All tools are also used to register progress information, from project planning to actual status.

5 A description can be found at: www.collab.net.

Furthermore all teams use a Burn down chart and the Increment as result of a Sprint. All source code is admitted to be sparsely commented, but Controller (GIT[6]) and Local (PVCS[7]) track information on versions and associated changes.

Sunflower and Local support the Sprint Backlog and its items with a whiteboard.

2.4.3 Non-Scrum Artefacts

There is one artefact all teams use in a similar way. This is progress information which is for the greater part available through their tools. Although minor details differ from one team to another we will use the term 'Project plan' for all of them, although it is not always mentioned as such. We will now continue with a list of non-Scrum artefacts for each team.

2.4.3.1 ***Controller***

Controller uses (exhaustive list):

- Test scripts which are drawn up by the testers of the Development Team in a so-called GWT-structure: Given defines the initial situation, When defines actions, Then predicts the final situation. Test scripts are typically product artefacts.
- An implementation document, which primary aim is to support consultants in the installation of software at a customer's site. Its contents include a description of the standard functions of a module, extended with instructions for the adjustment of parameters, user roles, et cetera. The implementation guide is a product artefact.

6 A description can be found at: www.git-scm.com.

7 A description can be found at: www.serena.com.

- A user guide containing a description of the software to support customers in their use of the software. The user guide is also a product artefact.

2.4.3.2 Sunflower

In addition to the core Scrum artefacts Sunflower produces a Help file accompanying the software; this file serves as a user guide. The Fix-list provides an overview of features in a release and is sent to customers at the date of a new release. Both are product artefacts.

2.4.3.3 Local

In addition to the standard Scrum artefacts Local also uses (exhaustive list):

- A functional design with major chapters on: Assignment / Scope, Constraints, Current and future situation, Adjustments data model, Adjustments set-up and Functional Description; this is typically a product artefact.
- A test plan containing: Background information, Logical test cases, Physical test cases (description of test configuration and test scenarios) and Expected results & (screen prints showing) Actual result; this is again typically a product artefact, although it has an small and implicit aspect of a process artefact in the sense that from the availability of actual test results (or not), progress may be derived.
- A user guide accompanying the software; it is a product artefact.
- Release notes accompany every release, highlighting its new features. They are a product artefact.

2.5 Discussion

In the previous section we have listed all artefacts in use with the 3 case study organizations. We will now turn to a discussion of the results in relation to our research questions:

RQ1 Which artefacts do Scrum teams produce?

RQ2 How can artefacts be characterized to designate their use in Scrum team practices?

2.5.4 Artefact Model

To answer our first research question we first summarize the artefacts found in the case studies (Table 2.2).

Table 2.2 - Summary team's artefacts

Artefacts	**Artefacts**		
	Controller	**Sunflower**	**Local**
Project plan	Project plan	Project plan	Project plan
Product Backlog	Product Backlog	Product Backlog	Product Backlog
Sprint Backlog	Sprint Backlog	Sprint Backlog	Sprint Backlog
Backlog item	Backlog item	Backlog item	Backlog item
Design	-	-	Functional design
Test	Test script	-	Test plan
Increment	Source code	Source code	Source code
	Release	Release	Release
Release notes	-	Fix list	Release notes
Implementation guide	Implementation document	-	Implementation guide
User guide	User guide	Help file	User guide
Burn down chart	Burn down chart	Burn down chart	Burn down chart

In the leftmost column of Table 2.2 we find Scrum artefacts as well as non-Scrum artefacts. The definition of 'Increment' in the Scrum process framework is

somewhat ambiguous, i.e. it is not definite on whether the increment consists of either source or compiled code. For this reason, and because of the results from the case studies, we separated Increment into source code and release (compiled code). To incorporate non-Scrum artefacts in this column we introduced generic descriptions derived from a general software development life cycle. The three other columns contain the artefacts encountered in the case study organizations.

The leftmost column, synthesis of the results of the study, now contains building blocks to redraw the preliminary model as extended Scrum artefact model (Figure 2.3).

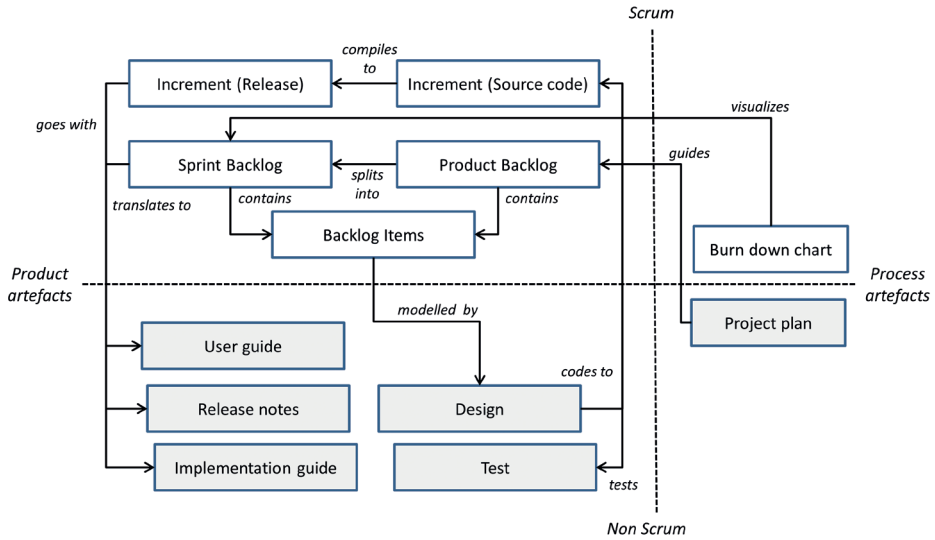

Figure 2.3 - Extended Scrum artefact model

In comparison with our preliminary model (Figure 2.3) we note that all Scrum artefacts were already included. For the non-Scrum artefacts Project plan and Test were al-ready identified, Design was not and Release-related material has been refined.

The extended Scrum model still contains artefacts from both the Scrum artefact and the agile artefact class diagram. Both individual class diagrams do not cover the case studies; the combination does for a majority of artefacts.

At the same time our artefact model is rather reticent in its number of artefacts in comparison with the other diagrams. This is an explicit choice. We choose to nominate artefacts independent of their representation. Where in the AACD Backlog Item is a generalization of (all of) Requirement, Task and Issue/Bug, we restricted ourselves to Backlog Item.

2.5.5 Non-Scrum Artefacts: Design, Test and Release-related Material

Whereas the class diagrams together make a fair prediction of most artefacts in our model, they fail to include a 'design' artefact. The importance of functional documents for computer systems has been established for a long time: Inadequate documentation causes software quality to degrade over time (Parnas & Madey, 1995). And design is an artefact clearly identified in our research. The SACD in general lacks 'intermediate' results of software development, although it could be argued that the system architecture and the domain model represent (parts of) a design. But both system architecture and domain model cover a Sprint Backlog as a whole, perhaps even the Product Backlog. The design we found relates to one requirement, which makes it a new element in agile artefact models. Local explicitly decided to include designers in its Scrum team to draw up a design as an artefact. Local did so because a gap was experienced between the description of some Backlog items and their elaboration in source code by a developer. In addition, Scrum team members from Controller and Sunflower do not make a formal design, but they use whiteboard, computer or pencil and paper to make informal drawings or Visio-diagrams.

More generally we observe that the teams, while using Scrum and its artefacts, do not feel themselves limited in their production of additional, non-Scrum, artefacts. We already mentioned Local's design artefact. Both Controller and Local introduced testers in the team to write and execute test scripts/plans to assure software quality.

Furthermore, the Scrum teams produce 'final' results, other than an increment (release). We found quite some artefacts being in just this category: Implementation guide, user guide, release notes. One could argue that such artefacts are obligatory, dictated by the outside world, such as an ordering party or a customer. And the organizations all produce product software, which might, to some extent, explain their emphasis on such artefacts. But only to some extent, because these artefacts are certainly not exclusively related to product software development. According to Scrum, a choice on the production of artefacts is situational and left to the team; we conclude that most of the teams decide to produce beyond the Scrum minimum set and they choose for non-Scrum product artefacts.

It is also good to notice which artefacts we did not find. A system architecture and a domain model from the SACD (Architectural information in our preliminary model), were not present, for none of the teams. It has been noted that documentation may play a much more important role in agile methods being applied in a distributed or large organisation (Chau, Maurer, & Melnik, 2003). One advantage to this emphasis on knowledge externalisation is that it reduces the likelihood of loss of knowledge as a result of knowledge holders leaving the organization. Although two of the case study organizations are larger organizations, we found no evidence for artefacts aimed at the preservation of knowledge over a number of Sprints. Design and test artefacts play their role

only in the current Sprint and may be considered as to apply to one item of the Sprint Backlog only. An overall architecture for the Product Backlog or even some design to bundle requirements in a Sprint was not encountered. This is in line with previous findings where it was found that internal documentation was perceived as lagging behind (Lagerberg & Skude, 2013; Stettina & Heijstek, 2011). Thus when interviewees were asked how teams would acquaint a new employee with the team's previous efforts and results, all answered: "*Training on the job*", to admit thereafter that this was not a wise practice. Two teams provided some guidance as how to solve this problem. A user guide is not meant for internal use, but when an organization provides courses for (prospective) users, these courses are very often structured using this guide. And the teams admitted to send their new employees to just this course to give them an impression of the software they will be working on. Via a detour external documentation thus becomes internal documentation. But it is still somewhat disturbing that none of the software products is backed up by a sound explicit architecture, the more because the products have quite an extensive installed base and/or number of users. There seems to be a great trust in tacit rather than explicit knowledge (Nonaka, 1994).

2.5.6 Supporting Tools

All teams use electronic workspaces to support Product and Sprint Backlog with their individual items as well as the accompanying project plan. Version control was explicitly used by two of the teams (Controller and Local); the other team used some scripts to produce a release.

A need for tools when using Scrum in the context of global software development was already established (Hossain, Babar, & Paik, 2009). We observed all teams using tools, although they were all co-located. Most tools have in common that

they support both product and process artefacts. We conclude that the co-located teams use integrative tools, perhaps as intensive as distributed teams. But they do not use specific tools to support non-Scrum product artefacts.

2.6 Conclusions

Our paper presents a clear and empirical overview of which artefacts Scrum teams use and what they use them for. The value of our work lies in its thorough foundation in practice, which provides practitioners with a reference model to mirror their own choices, whilst at the same time improving abstract artefact models by introducing new (categories of) artefacts.

No existing individual agile artefact model included the artefacts we encountered in our research, although a majority of them was covered by the combination of SACD and AACD. Thus combining the models already was a step forward. But we also enriched the models by identifying a new artefact not yet included: A design artefact. We also made a clear distinction between product and process artefacts in our model; one that was lacking in the diagrams so far. And finally we abstracted our model by separating function from form in denominating our artefacts independent of their origin (Backlog Item instead of feature, bug, task) and separating supporting tools from artefacts themselves.

Dimensioning our Scrum artefact model in Scrum and non-Scrum as well as product and process artefacts shows that, apart from the core Scrum artefacts (Product and Sprint Backlog, Increment), Scrum teams produce many other artefacts, especially non-Scrum product artefacts: Design, test and release-related material. Despite the popularity of Scrum, elements from previous software development methods linger on.

All teams use support tools for the Product and Sprint Backlog, whether or not combined with a whiteboard, while at the same time including (Sprint) project management information as a process artefact. Albeit the fact that all teams are co-located they use supporting tools (and artefacts); this use is not exclusively reserved for distributed teams.

We did not find evidence for artefacts aiming at the preservation of knowledge over a number of Sprints. All non-Scrum product artefacts play a role only in one Sprint and may be considered even as to apply to one item of the Sprint Backlog only. An overall architecture or even a design bundling the requirements in a Sprint was not encountered.

Limitations are inherent to a choice for case studies as research method. We studied three teams using Scrum. Because of this number we feel some generalization of our results is justified, but the sample is small.

CHAPTER 3

Criteria for selecting a Scrum tool

Scrum teams extensively use tools to support their processes, but little attention has been given to criteria a Scrum team applies in its selection of such a tool. A greenfield approach was used to explore these criteria. To this extent twelve Scrum teams were asked to list criteria and assigned weights in their decision processes. After having chosen and used a tool for a number of Sprints, the teams also evaluated the selected tools. Using the Technology Acceptance Model to structure findings, two major categories were identified: Perceived usefulness, alias criteria directly related to Scrum, and perceived ease of use. Most teams listed more or less the same criteria. Within the categories several specific subcategories were distinguished, for instance burn down chart support or multi-platform aspects. Teams evaluated more issues, positive or negative, within the Scrum-related criteria. The findings indicate that Scrum teams prefer perceived usefulness over perceived ease of use. In other words: Specific support of Scrum, especially its artefacts, are of greater value to a team than general tool considerations.

This work was originally published as:

Wagenaar G., Overbeek S., Helms R.: Describing Criteria for Selecting a Scrum Tool Using the Technology Acceptance Model. In: Nguyen N., Tojo S., Nguyen L., Trawiński B. (eds.) Proceedings of the 9th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2017), Kanazawa, Japan, 3-5 April 2017, 811-821. Springer, Cham (2017)

3.1 Introduction

Agile software development (ASD) prefers "*Individuals and interactions over processes and tools*" with an annotation: "*… there is value in the items on the right …*" (Beck et al., 2001). Tools have proven to be valuable, as, for instance, global Scrum projects depend on a wide range of tool support (Hossain & Babar, 2009). This includes communication and collaborative tools for issue or bug tracking, backlog management, and burn down chart visualization. Also in a collocated project a tool proved to effectively manage a Scrum project (Schatz & Abdelshafi, 2005). In general, communication and collaboration tools, whether used face-to-face or global, are crucial in modern agile teams (Crowder & Friess, 2015).

Agile tools come in abundance (Alyahya, Alqahtani, & Maddeh, 2016); a choice has to be made by an agile team. However, knowledge of criteria applied by a team in this choice is anecdotal and fragmented, whereas such knowledge gives insight in where agile teams need support. This in turn deepens our theoretical understanding of agile processes through the explicit articulation of important ASD elements by its practitioners, while at the same time practically supporting them in choosing appropriate support. Because of our approach, a greenfield one, our results would especially be useful for novel teams involved in an agile transformation.

The remainder of this paper is organised as follows. In Section 3.2 we discuss related work; in Section 3.3 we present our research method. Results are shown in Section 3.4, followed by a discussion and conclusions in Section 3.5, which is the final section.

3.2 Related Work

Research on criteria agile teams use for tool support has not been extensive so far. Fragmented research has been presented, which can be captured under two headings: (1) Individual experiences (Engum et al., 2009; Møller et al., 2009; Uy & Rosendahl, 2008) and (2) Research explicitly addressing the need for a more general approach (Azizyan et al., 2011; Taheri & Sadjad, 2015). However, all tend to assess tools on the basis of existing criteria rather than to tackle the problem of first defining criteria for the selection of tools thereafter. We now first discuss both categories and introduce a more general theoretical viewpoint afterwards.

3.2.1 Experience Reports

Experience reports describe a selection process or, somewhat more general, a transition process towards ASD. (Uy & Rosendahl, 2008) discuss a case where a technology team, responsible for a corporate website as well as a number of other products and information systems, entered a migration trajectory from SharePoint to a tool better suited for Scrum. Three critical factors were used to distinguish between two short-listed alternatives (Rally, VersionOne): Usability, functionality, and configurability. Additional technical considerations concerned were in the areas of enterprise infrastructure, architecture and quality assurance.

(Møller et al., 2009) designed a Scrum tool dedicated to assist in a daily Scrum meeting with four overall requirements: Intuitive user interface, high accessibility, commitment to Scrum, and project history. The requirements were found on the basis of experiences of three Scrum teams.

(Engum et al., 2009) report on a small North European agile company, which transitioned to a tool for managing Product and Sprint Backlog and distilled lessons from its experiences. Impediments the tool should address included

task description and tracking, specifically in order to help organize, specify and prioritize the product backlog and track development, and time usage.

Whether phrased as guidance (Engum et al., 2009), tips (Schatz & Abdelshafi, 2005) or lessons learned (Uy & Rosendahl, 2008) all experience reports acknowledge the fact the criteria used are situational, that is, driven by the specific circumstances of the organization, the project, or the team. This is no surprise, since it is already known that a software development process itself is already regarded as being dependent on the situational characteristics of individual software development settings (Clarke & O'Connor, 2012). If the choice of a process itself is already situational, the choice of a tool to support the process is situational too.

3.2.2 Criteria Models

To move beyond the experiences from a single case, more recent research explicitly addresses the need for this general approach, especially modelling criteria for tool evaluation. (Azizyan et al., 2011) investigated (dis)satisfactory aspects of agile tools through a survey of agile tool usage and needs. The criteria used to evaluate agile tools were: Ease of use, integration with other systems, customizability, availability of reports, and price.

(Taheri & Sadjad, 2015) used a comparison chart with criteria in a selection process for a tool: Lifecycle coverage, Simplicity & ease of use, Collaboration, analytics, visibility & reporting, workspace & process, program management, deployment, and integrity & security. The criteria originated, beside from previous research, for a large part from surveys, sponsored by vendors of agile tools. This introduces a bias, since vendors are of course unlikely to use unfavourable criteria.

3.2.3 Theoretical Viewpoint

Both criteria models, and also the individual experiences, draw heavily upon existing surveys, where their resources are subject to bias. To have a more independent viewpoint the Technology Assessment Model, TAM (Davis, 1989; Davis et al., 1989) can be used. Its central proposition is that user acceptance of technology depends on perceived usefulness and perceived ease of use. Perceived usefulness is "*the degree to which a person believes that using a particular system would enhance his or her job performance*"; perceived ease of use "*the degree to which a person believes that using a particular system would be free of effort*" (Davis, 1989, *p.230*). TAM theorizes that the effects of external variables (e.g., system characteristics, development process, and training) on intention to use are mediated by perceived usefulness and perceived ease of use (Venkatesh & Davis, 2000). Both categories are also included in the Unified Theory of Acceptance and Use of Technology (UTAUT) as performance and effort expectancy, as are 2 others: social influence and facilitating condition (Venkatesh, Morris, Davis, & Davis, 2003). TAM thus provides a useful first distinction in modelling criteria for agile tool support. This is confirmed, although not explicitly by the use of TAM, in an evaluation of four agile project management tools, where both usability evaluation criteria and a task-oriented usability inspection were used (Alomar, Almobarak, Alkoblan, Alhozaimy, & Alharbi, 2016); these categories reflect very well the TAM categories. This supports the use of dichotomy in TAM in modelling criteria that agile teams use for tool support.

3.3 Research Method

In our research we have chosen for a greenfield approach, in analogy with the greenfield project: "*In software engineering jargon, a greenfield is a project which lacks any constraints imposed by prior work*" (Gupta, 2011, *p.21*). Although this approach may be considered as situational as any other, it provides a relatively

unbiased way to explore features a Scrum team thinks of as being important in its support. It at least assures that prior organizational or commercial influences are excluded. To implement this approach we used student groups, where we phrased our research question as follows:

> *Which criteria do Scrum teams adopt when choosing computer-based support for their Scrum process?*

We characterize our research as a variant of case study research with the teams being the cases, but explicitly instructed through their assignments.

In the next two paragraphs we will describe, first, the educational context of the student groups and, second, the assignments the groups were provided with.

3.3.1 Context

The teams in this study are teams of second year students in a four year Information Technology study programme leading to a Bachelor degree. Students have experience with software development methods, some basic experience with waterfall-like development, followed by other methods, such as RUP. In ten weeks students were to develop an application, applying Scrum as ASD method.

In the first three weeks of the course students acquainted themselves with Scrum. After a lecture on agile methods in general students were referred to the Scrum guide (Schwaber & Sutherland, 2016) for self-study and conducted a test consisting of twenty multiple choice questions afterwards. Furthermore, they were provided with training on all major Scrum elements, which are roles, artefacts and events.

From week four onwards the students made three Sprints of four days each, working fulltime on their application. During a Sprint teams held all Scrum meetings: Sprint Planning Meeting (Monday morning), Daily Scrum (Tuesday till Thursday morning), Sprint Review, followed by an informal Sprint Retrospective (Thursday afternoon). Teams were stimulated to collectively work at the same place, for instance a computer or a conference room. A few teams had their members working (more) individually, sometimes from home. From week seven onwards students continued to work on the application individually using team results as a basis. This had didactical reasons and although still working in a Scrum-like way, this part of the course is left out of consideration here, if only because communication between team members did not take place anymore.

3.3.2 Assignments

Students were divided into twelve Scrum teams, with each team between four and six members. Before starting their first Sprint, teams were asked to select a tool to support them in their Scrum process as a first assignment. In more detail teams were asked to specify at least five weighted criteria they thought relevant for their Scrum tool, three different alternatives (Scrum tools), and a score on each criterion for each alternative. In addition, teams were asked to motivate all of the alternatives, criteria and scores.

As a second assignment, teams were asked for an evaluation after their last Sprint. They were specifically asked to answer two questions: On which characteristics did the tool (not) meet your expectations? In other words, where did the tool (not) effectively and efficiently support the Scrum process?

3.3.3 Data Collection and Analysis

For data collection, we first collected reports with regard to the first assignment from all twelve groups at once. We analysed the reports according to Grounded Theory. This is a systematic research method known for the generation of theory derived from systematic and rigorous analysis of data (Glaser, 1992; Glaser & Strauss, 1967). This does not mean that there is not a specific problem, but rather problems and their key concerns emerge during data analysis (Glaser, 1992).

To analyse the data, we used three abstraction levels: category, concept, code (Allan, 2003). For the highest level, category, we adopted TAM: Perceived usefulness and perceived ease of use. Applying open and axial coding (Corbin & Strauss, 2015) we found at the next level concepts like support of a product or sprint backlog (perceived usefulness) or multi-platform support (perceived ease of use). Through coding different wordings were mapped onto the same (sub-) criteria. For instance, the criterion "Ease of use" was phrased as: "*The users must be able to use it easily*", "*The cognitive load should be low*", and "*Often used functions should be clearly visible*". Coding was a fairly straightforward exercise, because the assignment by its nature already led to structured reporting.

Our analysis proceeded from one report to another; we coded a report for criteria and weights, adding to our abstraction levels where necessary. After each change we scrutinized previous reports with respect to the most recent supplements. This was for instance necessary when we met a criterion previously considered as sub-criterion, or vice versa. This overlap between criterion and sub-criterion happened more often and has been resolved by using our judgement in the analysis, supported by the classification already applied by the teams themselves and the number of teams that used the one or the other.

We also collected reports for the second assignment. We analysed them for positive and negative remarks, but only superficially coded them, because most teams did not explicitly refer to their (sub-)criteria, thus allowing coding at the category and concept level only.

3.4 Results

The results from our research are described next. We first describe the criteria which were used by the teams to evaluate the tools and show the weights teams attached to them. We then provide a summary of the tools considered by the teams and present a brief overview of the teams' evaluation after having used their tool.

3.4.1 Criteria

All twelve Scrum teams reported criteria, alternatives, scores as well as considerations leading to them. They reported their sources as (Internet) research, acquaintance with a tool (for instance through experience from a part time job), advice from experts, including teaching staff, or a combination. Since the teams worked independently, we applied coding to highlight similarities and discrepancies, especially between (sub-)criteria, which includes homonyms and synonyms.

Six teams used a flat list, that is, all criteria were on the same level of abstraction. The others had a hierarchical approach, in which a classification of criteria with sub-criteria was used, yielding a two level hierarchy. After analysis a hierarchical list of criteria and sub-criteria arose, structured at the top level according to TAM (Table 3.1); this list is the superset of all individual lists. All (sub-)criteria could be classified as a TAM category, perhaps with the exception of price, which we included under perceived ease of use, but might have had its own category.

Table 3.1 - Overview of categories, criteria and sub-criteria

Category Criterion *Sub-criterion*	#[8]
Perceived usefulness	1
Contains visualization task board	5
split in to do, doing, testing & done	*7*
Supports backlog(s)	-
contain product backlog (items)	*5*
contain sprint backlog (items)	*5*
assign priorities to items	*2*
split items in tasks	*3*
link items/tasks to team members	*4*
allow comments on item/task	*5*
allow documents per item/task	*2*
register time per task	*1*
sort backlog items on priority	*2*
show status backlog item	*1*
show activities per member	*1*
show deadlines for tasks	*3*
show effort (per member)	*1*
show statistics	*4*
couple sprint backlog to releases	*1*
Supports burn down chart	8
show team	*1*
show individual	*1*
Has additional features	1
Has Scrum compatibility	2
allow (user) stories	*1*
have predictions	*1*
support planning poker	*1*

Category Criterion *Sub-criterion*	#
Perceived ease of use	
Ease of use	9
can be personalized	*3*
have an easily accessible lay out	*3*
have short loading & response time	*2*
have user guide / tutorial	*1*
Multiple platform support	7
have a website	*1*
be app/application/board	*4*
be physical	*1*
can be reached from other locations	*1*
Communication support (e-mail, chat)	2
Security mechanism support	-
require log in	*2*
administer member rights	*3*
Integration with other tools	-
Visual Studio	*1*
GIT	*2*
Acceptable price (or trial available)	10

8 # is the number of occurrences of the (sub-)criterion.

The frequency for criteria, not being sub-criteria, represents the number of times the criterion itself is mentioned directly, i.e. it does not in any way accumulate frequencies of sub-criteria.

All criteria and sub-criteria were rated with relative importance by the teams, see Table 3.2.

Table 3.2 - Overview of weights of criteria

Category	Criterion	Teams	Weight[9]	σ[10]
Perceived usefulness	Contains visual task board	8	12,5	14
	Supports backlog(s)	10	27,8	19
	Supports burn down chart	9	8,3	6
	Has additional features	1	1,3	4
	Has Scrum compatibility	3	3,4	6
Perceived ease of use	Ease of use	10	17,2	16
	Multiple platform support	11	11,8	9
	Communication support (e-mail, chat)	2	2,1	5
	Security mechanism support	3	2,3	5
	Integration with other tools	3	3,3	7
	Acceptable price (or trial available)	10	10,0	7

In Table 3.2 the number of teams refers to teams that have listed the criterion or one of its sub-criteria. The weight is calculated as the sum of the relative weights for teams who listed the criterion and/or one of its sub-criteria divided by the total number of teams (twelve). To this extent the scale of weights for each individual team has been normalized to a scale of 0 – 100. Furthermore, we have restricted ourselves to the level of criteria in Table 3.2. Incorporating sub-criteria in the table would have little value, because the vast majority of sub-criteria is

9 All numbers are rounded to 1 decimal.

10 σ: Standard deviation, rounded to an integer.

mentioned three times or less, which would in general result in lots of weights smaller than one.

3.4.2 Choice of Tools and Evaluation

Teams were free in their choice of alternatives. A total of eighteen different Scrum tools was considered. All tools were fairly dedicated Scrum tools with the exception of Trello (a tool to organize anything), and Proofhub and Dapulse (project management tools). Five of them were finally chosen by the teams. Visual Studio Online (Team Foundation Server) was the most chosen one, five times, followed by Trello, four times. The other three teams used Scrumwise, ScrumDo and QuickScrum.

In the evaluation almost all teams reported to be satisfied with their choice of a tool. Their satisfaction was backed up by both positive and negative remarks of (the use of) their tools. We counted the number of remarks, related them to our two (TAM-)categories and subdivided them into positive or negative feedback (Table 3.3).

Table 3.3 - Feedback on chosen tools

		Team												Total
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	
Perceived usefulness	**Positive feedback**	7	2	4	1	1	1	4	3	2	3	1	5	34
	Negative feedback	3	3	2	3	-	-	1	1	2	1	1	-	17
Perceived ease of use	**Positive feedback**	2	1	-	4	5	5	3	-	1	1	2	-	24
	Negative feedback	2	-	-	1	-	-	-	1	-	1	1	2	8

We have related remarks to the category only, because the evaluation data most often was not related to only one of the (sub-)criteria. Some examples of remarks are:

"*The basic features of the tool, such as the task board, the description of Sprint Backlog items and the planning of Sprints, were easy to use*".

"*We would have liked planning poker points instead of hours to measure velocity*".

One team switched from tool after Sprint one. It had chosen for Trello, but started using Visual Studio Online for Sprint two and three.

3.5 Discussion and Conclusions

In this section we first compare our results with previous research. We then discuss the (sub-)criteria in detail and analyse the teams' preferences. Finally, we discuss limitations of our study.

3.5.1 Comparison with Previous Models

The teams collectively came up with 45 criteria, including sub-criteria, far beyond and more detailed than any of the previous models (Azizyan et al., 2011; Taheri & Sadjad, 2015). However, our sheer amount still does not warrant a claim to completeness. And we do notice a match between criteria from previous research and our current research, at least at the level of criteria. Furthermore, there was consensus among the teams. Six criteria were mentioned by at least 75% of the teams (visual task board, backlog(s), burn down chart, ease of use, multiple platform, acceptable price), with the remaining ones all below 25% (see Table 3.1).

We conclude that our Scrum teams, most likely unaware, followed theory (TAM) and practice (experience reports and models) in the constitution of their list of relevant (sub-)criteria, but added lots of detail concerning practical considerations, such as wishes to prioritize and sort backlog items or to integrate with GIT. This level of detail was not established in previous research and in

this way our results are especially attractive for teams in agile transformation, when adopting agile tools.

4.5.2 Classification of Criteria

We found criteria could easily be classified according to the TAM, with no overlap between Perceived usefulness and Perceived ease of use. Notwithstanding the high standard deviations in the attribution of weights (Table 3.2), a separation between '(sub-)criterion mentioned often / high weight' and '(sub-)criterion mentioned seldom / low weight' is clear and cuts through the categories: Teams value support of Scrum artefacts most, as is shown by the number of criteria devoted to the Product and Sprint Backlog (and the burn down chart), and acknowledged by the weights they attached to those criteria. This observation is also confirmed by the evaluation where teams devoted more remarks, positive or negative, to Perceived usefulness than Perceived ease of use, approximately in a ratio 5:3 (Table 3.3). This suggests Scrum teams value perceived usefulness over perceived ease of use.

However, this is not confirmed in the scores they assign to their alternatives / criteria, and hence the choice of their tool. Trello is the second best chosen tool, and Trello may certainly be suited for the task, but it is a general tool and certainly not a Scrum specific one. However, the one team that switched from Trello to Visual Studio Online (VSO), when comparing the two, commented: "*VSO seems, as far as burn- down chart and overall look & feel are concerned, far more professional than Trello*". "*Professional*" here also implies the existence of more and better Scrum features in VSO. Probably Trello's ease of use and flexibility lured some teams away from more specific Scrum tools.

We conclude that Scrum teams prefer perceived usefulness over perceived ease of use. Especially support of Scrum's artefacts is of greater value to a team than general tool considerations.

4.5.3 Limitations

We excluded commercial and organizational influences by involving student Scrum teams with basic and theoretical knowledge of Scrum. A drawback of this greenfield approach is that our findings apply to novice Scrum practitioners. Generalizability of results is therefore limited.

The teams tended to prefer freely available Scrum tools, or a trial version thereof. Although this may have restricted their choice of a tool, it did not influence the constitution of the list with criteria, which was our main research goal.

PART II

Role of artefacts in agile team communication

CHAPTER 4

Influence of maturity on artefacts usage

Context *Agile software development (ASD) uses 'agile' artefacts such as user stories and product backlogs as well as 'non-agile' artefacts, for instance designs and test plans. Rationales for incorporating especially non-agile artefacts by an agile team mainly remain unknown territory.* ***Goal*** *We start off to explore influences on artefacts usage, and state our research question as: To what extent does maturity relate to the usage of artefacts in ASD in software product organizations?* ***Method*** *In our multiple case study 14 software product organizations were visited where software product management maturity was rated and their artefacts usage listed.* ***Results*** *We found maturity to be negatively correlated with the non-agile/all artefacts ratio. In other words, the more mature software product management is, the fewer non-agile artefacts are used in ASD.* ***Conclusions*** *This suggests that an organizational factor influences an agile team in its artefacts usage, contradictory to the concept of self-organizing agile teams.*

This work was originally published as:

Wagenaar G., Overbeek S., Lucassen G., Brinkkemper S., Schneider K.: Influence of Software Product Management Maturity on Usage of Artefacts in Agile Software Development. In: Felderer M., Méndez Fernández D., Turhan B., Kalinowski M., Sarro F., Winkler D. (eds.) Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbruck, Austria, 29 November – 1 December 2017, 19-27. Springer, Cham (2017)

4.1 Introduction

Agile software development (ASD) has been introduced in the domain of product software development (Dzamashvili Fogelström, Gorschek, Svahnberg, & Olsson, 2010; Vlaanderen, Jansen, Brinkkemper, & Jaspers, 2011), with product software defined as: *"A packaged configuration of software components or a software-based service, with auxiliary materials, which is released for and traded in a specific market"* (Xu & Brinkkemper, 2007, *p.534*), where auxiliary materials consist of software documentation, user material and the like. Product software differs from tailor-made software in, among other aspects, the importance of architecture (Xu & Brinkkemper, 2007). The necessity of auxiliary materials and the requirement of a future-proof architecture are indicative for the use of documentation artefacts in the product software development lifecycle. Research in ASD has devoted attention to the usage of artefacts, where a distinction can be made between 'agile' and 'non-agile' artefacts. Agile artefacts are artefacts which are inherent to an ASD (for instance user stories or a backlog); all other artefacts are considered to be non-agile (for instance architectures or designs). In agile product software development, software product organizations (SPOs) as manufacturers of such software could be expected to use non-agile artefacts precisely because of their needs with regard to architecture and auxiliary materials. In this research we explore one influencing factor on the decision to use, especially non-agile, artefacts. To this extent we assume that artefacts usage is a quality consideration and relates to the quality of software product management (SPM) in an SPO, where software product management is the discipline and role, which governs a product from its inception to the market/ customer delivery in order to generate biggest possible value to the business (Ebert, 2007). In the Capability Maturity Model Integration for Development (CMMI-DEV), documentation artefacts, for instance architecture documentation

and design data, are explicitly mentioned and they contribute to achieving higher maturity levels (CMMI Product Team, 2010).

To explore influencing factors we formulate our research question as: ***To what extent does SPM maturity relate to the usage of artefacts in ASD?***

Fourteen organizations were visited as part of a multiple case study. Our findings show a negative correlation between SPM maturity and the usage of non-agile artefacts. Altogether our findings contribute to a better understanding of factors that influence an agile team in its artefacts usage, an area in which research is scarce. From a practitioner's perspective one of the principles behind the Agile Manifesto, "The best architectures, requirements, and designs emerge from self-organizing teams" (Beck et al., 2001), is put to the test if an organizational factor can be shown to relate to artefacts usage.

The remainder of this paper is organized as follows. In Section 4.2 we outline the theoretical background with an overview of research on the usage of artefacts in ASD (Section 4.2.1), and SPM in general and a method to establish its maturity in particular (Section 4.2.2). In Section 4.3 we present our research method, a multiple case study, including data collection and coding, leading to our findings. Section 4.4 discusses our conclusions, which may be summarized as a new insight in the relation between SPM maturity in SPOs and the usage of non-agile artefacts in ASD in SPOs.

4.2 Theoretical Background

4.2.1 Artefacts in Agile Software Development

An artefact (in ASD) is defined as a tangible deliverable produced during software development (Wagenaar, Helms, Damian, & Brinkkemper, 2015). In ASD artefacts, such as architectures, requirements, and designs, are used as a decision of the self-organizing ASD team (Beck et al., 2001), dependent on the value it attaches to them. ASD practitioners perceive their internal documentation as especially important but feel that too little of it is available (Stettina & Heijstek, 2011). A decision on usage of artefacts is in fact a decision on 'non-agile' artefacts, because agile artefacts already are part of an ASD method itself. Previous research shows the dilemma of the optimal level of agile and non-agile artefacts in ASD. (Gröber, 2013) constructed an (agile) artefact class diagram with artefacts and relationships between them as result of a systematic literature study on the usage of artefacts in agile methods. Based on this research and adding findings from three case studies (Wagenaar et al., 2015) developed a Scrum artefact model, distinguishing product from process artefacts and Scrum from non-Scrum artefacts. In a study on large-scale offshore software development programmes (Bass, 2016) identified 25 artefacts on five levels of abstraction: Programme governance, Product, Release, Sprint, and Feature.

In summary, various models show a mixture of agile and non-agile artefacts, although based on different viewpoints varying from agile or Scrum development to offshore software development. The models classify both agile and non-agile artefacts, but, with one exception, do not explicitly address the distinction between the two. This precludes, as one consequence, insight in reasons for using them.

4.2.2 Software Product Management Maturity

Assessing maturity of software development processes and thus contributing to their improvement has led to several maturity models. General ones, like CMM (Paulk et al., 1993), or ISO/IEC 15504 (*ISO/IEC 33002:2015*, 2015) and more specialized agile models are all composed of hierarchical maturity levels, but are otherwise quite different in their domains, backgrounds, structures, and contents (Leppänen, 2013). However, because of our focus on SPM a more dedicated model, but similar in its constitution, is available, which describes the SPM process as consisting of four business functions: Portfolio management, Product planning, Release planning, and Requirements management (Bekkers et al., 2012; Bekkers, Spruit, Weerd, Vliet, & Mahieu, 2010). Each business function is in turn divided into focus areas. In case of the business function Requirements management, these are: Requirements gathering, Requirements identification, and Requirements organizing. The model has an associated method, the Situational Assessment Method (SAM), which can be used to measure a maturity level specifically for SPM (Bekkers, Spruit, et al., 2010). To this extent the SAM provides a matrix with an overview of capabilities at different levels that need to be implemented to reach a full-grown maturity. The matrix is used in a bottom up way. Maturity is ranked per focus area, and then aggregated to SPM maturity on a scale from 0 – 10.

4.3 Research Method

To investigate our research problem, we used a multiple case study, which is an accustomed way to investigate phenomena in a context where events cannot be controlled and where the focus is on contemporary events (Yin, 2013). Data collection took place through single-site case studies following Yin's widely accepted guidelines for case studies (Yin, 2013). We first collected basic data on artefacts and maturity on basis of a protocol including: (1) SPM theory and

research, (2) interview instructions, and (3) reporting guidelines. We found 14 organizations willing to participate, all using ASD[11]. The organizations develop product software (1) for a broad range of domains, from (semi-)government to software development, (2) with five to over hundred employees (organization as a whole), and (3) for ten to several thousands of customers. In case a SPO produced more than one product, one of them was selected. In Section 4.3.4 we discuss threats to external validity regarding our participating organizations.

4.3.1 Data Collection

Data collection was the same for all organizations. Interviews were held, ranging from one interview through one interview with two interviewees to two or more interviews with several interviewees. Interviewees were in general product manager or owner, although some Scrum masters were also included. Interviews lasted on average one hour. They were semi-structured to allow interviewees to speak freely and to be able to ask follow-up questions. The interview instructions concerned two tasks: (a) Determine SPM maturity, and (b) Describe artefacts during ASD[12].

For the establishment of maturity, a description of capabilities required to achieve a certain maturity level is already provided in the SAM (Bekkers et al., 2012). For each capability the organization being assessed has to answer the question "*Have you implemented this capability within your organization?*" with either Yes or No, for example: "*Can stakeholders submit requirements directly to the central database?*".

11 A description of the organizations is available at https://osf.io/dez9k/?view_only=3171388053194c549f09b22fe4fbcfc0.

12 Interview instructions are available at https://osf.io/dez9k/?view_only=3171388053194c549f09b22fe4fbcfc0.

For the listing of artefacts the interview guidelines were based on the life cycle of a user story or a requirement, starting with the SPM's pre-development stage (portfolio management, product planning). Then they continued with questions about the activities in ASD, often starting with user stories in a product backlog and ending with the production of source code. Finally, post-development activities were identified, such as bugs, again leading to requirements. For the description of this life cycle a common vocabulary was established by using the FLOW modelling technique (Stapel, Knauss, & Schneider, 2009; Stapel & Schneider, 2014). FLOW's emphasis on information and its distinction between solid and fluid information makes it suitable for the representation of artefacts. Documented information is called solid information if it is long term accessible, repeatedly readable, and comprehensive for third parties. In contrast, undocumented or fluid information is information that violates any one of the above criteria.

4.3.2 Coding

Data on maturity needed no further coding, because answers to questions from the SAM directly translate to a maturity level for each focus area (see Section 4.2.2).

Data analysis for artefacts started by extracting solid information as artefacts from the FLOW models, identifying 201 artefacts. Because of differences in SPO's terminology this initial list was subject to: (1) lexical analysis, and (2) semantic analysis (Jurafsky & Martin, 2008). In lexical analysis we removed distinctions in singular and plural forms, for instance 'Bug report' (listed 5 times) and 'Bug reports' (1 appearance). We removed adjectives, for instance mapped both 'Product roadmap' and 'Company annual roadmap' on 'Roadmap', and we unified words having the same lexical roots, for instance 'Acceptance criteria'

and 'Acceptation criteria'. This reduced the number of 201 to 123 artefacts. In further semantic analysis we used the description of solid information in the FLOW model to identify similarities and differences in artefacts. Based on this description and also guided by the artefact model (Wagenaar et al., 2015) and the artefact list (Bass, 2016) we further pruned our list. For instance, 'Application' with description "*Code implemented by developers based on the release and sprint plan*" and 'Code', "*A (set of) implemented and unit-tested product feature(s)*", were mapped.

Finally we excluded a number of artefacts, since not all artefacts in our findings are artefacts directly related to ASD. Since our interviews used the pre-development stage as starting point we identified some 'Business Artefacts': Business case, Business plan, Market intelligence, Market requirement, Strategy, and Roadmap. An SPO's strategy certainly influences decisions with an impact on ASD, but it is not an ASD artefact. Business artefacts are important SPM artefacts, but are neither produced nor used directly by an agile team.

4.3.3 Findings

We aggregated maturity in a maturity level per business function where this maturity is calculated as the average maturity of underlying focus areas (Table 4.1)[13]. For example, the focus areas 'Gathering', 'Identification', and 'Organizing' within the business function 'Requirements management', scored 7, 9, and 10 respectively for organization A. This results in (7 + 9 + 10) / 3 = 8.7 for 'Requirements management' for organization A. The last row shows the overall SPM maturity as the average of the four business functions.

13 Scores per focus area are available at: https://osf.io/dez9k/?view_only=3171388053194c549f09b-22fe4fbcfc0.

Table 4.1 - Maturity of SPM

	SPO	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Maturity	**Requirements management**	*8.7*	*4.3*	*5.0*	*7.3*	*4.3*	*5.7*	*7.0*	*5.7*	*4.7*	*5.3*	*4.7*	*4.7*	*4.7*	*10.0*
	Release planning	*6.3*	*3.8*	*9.0*	*5.8*	*5.3*	*6.5*	*8.5*	*7.2*	*5.3*	*7.5*	*3.7*	*7.3*	*6.0*	*7.2*
	Product planning	*5.0*	*3.7*	*5.3*	*6.7*	*6.7*	*8.3*	*10.0*	*6.0*	*7.0*	*6.7*	*5.3*	*4.0*	*8.0*	*5.7*
	Portfolio management	*5.0*	*5.0*	*7.0*	*8.0*	*8.7*	*8.3*	*8.7*	*7.0*	*7.0*	*4.3*	*5.3*	*4.3*	*5.7*	*7.3*
	Overall SPM maturity	6.25	4.21	6.58	6.96	6.25	7.21	8.54	6.46	6.00	5.96	4.75	5.08	6.09	7.54

We found a total of eighteen artefacts, which were mentioned by at least two organizations (Table 4.2). The one but last row in Table 4.2 lists the number of artefacts (per organization) which were mentioned by that organization only. Since they tend to be rather organization-specific we have aggregated them in this way.

Table 4.2 - Artefacts per SPO

Artefact	A	B	C	D	E	F	G	H	I	J	K	L	M	N
User story	v	v	v			v	v		v	v	v	v	v	v
Code	v	v		v	v	v	v		v	v	v			
Sprint backlog		v	v	v		v		v	v				v	v
Epic	v		v			v	v						v	v
Product backlog		v	v	v		v		v						v
Definition of done	v	v				v		v					v	
Estimated user story		v		v							v			
Agile artefacts	*4*	*6*	*4*	*4*	*1*	*6*	*3*	*3*	*3*	*2*	*3*	*1*	*4*	*4*
Product requirement	v		v		v				v	v	v		v	v
Bug report	v				v				v	v	v		v	
Release note		v	v		v		v		v		v			
Test deliverables		v			v		v		v			v		
Request for change		v			v	v						v	v	
Acceptance criteria	v					v					v			
Release	v				v					v				
Functional design		v										v		
Release plan					v				v					
Technical design		v									v			
User documentation			v				v							
Non-agile artefacts	*4*	*5*	*3*	*0*	*7*	*2*	*3*	*0*	*5*	*3*	*5*	*3*	*3*	*1*
Organization-specific	*5*	*2*	*3*	*2*	*1*	*2*	*1*	*0*	*1*	*0*	*0*	*6*	*2*	*0*
Non-agile ratio	***0.69***	***0.54***	***0.60***	***0.33***	***0.89***	***0.40***	***0.57***	***0.00***	***0.67***	***0.60***	***0.63***	***0.90***	***0.56***	***0.20***

Artefacts in Table 4.2 are also classified in one of two categories: (1) Agile artefacts, and (2) Non-agile artefacts, since we are especially interested in the usage of additional, non-agile artefacts. We identified 'Agile artefacts' as: Product backlog, Sprint backlog, Code, User story, Epic, Definition of done, and Estimated user stories, because those are explicitly part of agile practices (Cohn, 2004; Schwaber & Sutherland, 2016). Various artefacts all are non-agile artefacts. To be able to compare between organizations we calculated the ratio of non-agile artefacts compared to the total number of artefacts.

Our research question was: To what extent does SPM maturity relate to the usage of artefacts in ASD? We identified both SPM maturity (Table 4.1) and the usage of ASD artefacts (Table 4.2). A measure of correlation dependency between two variables is the Pearson correlation coefficient (Wohlin et al., 2012). We calculated it between SPM maturity and non-agile artefacts ratio as $\rho(14) = -0.3576$. This outcome is considered to be of a weak to moderate strength. The answer to our research question thus is: SPM maturity is negatively correlated with the non-agile/all artefacts ratio. In other words, the more mature SPM is, the fewer non-agile artefacts are used in ASD.

4.3.4 Validity

Validity of our research depends on four criteria: Construct validity, internal and external validity, and reliability (Yin, 2013). To enhance construct validity we: (1) had interviewees comment on results of interviews, (2) complemented viewpoints in the interviews with more than one interviewee, and (3) followed a strict procedure in interpreting an interview, by means of the FLOW modelling technique as well as in applying the SAM. Nevertheless, organizations were not visited by one and the same interviewer, so interpretation may have influenced especially the listing of artefacts. Additionally, the maturity level is based on self-

assessment, which may introduce bias. Internal validity is mainly a concern for explanatory case studies, but we did apply pattern matching in translating solid information in the models through lexical and semantic analysis to our artefacts listing. External validity benefits from using a multiple case study on the basis of a common procedure. It has to be noted however, that our results only show a weak to moderate correlation. Furthermore, we visited SPOs, which was also reflected in our choice for measuring maturity. Generalizability to non-SPOs is therefore limited. From our findings organizations E and L show remarkable ratios, using (far) more non-agile artefacts than agile ones. This may be reason to question their application of indeed an ASD method. Finally, reliability increases because of our use of a procedure with interview instructions and the use of a case study database.

4.4 Conclusions and Future Research

We rated SPM maturity for 14 organizations and listed their artefact usage. We found evidence for SPM maturity to be negatively correlated with the non-agile/all artefacts ratio. A possible explanation could be that a 'mature' SPO' has organized its software product management already in such a way that additional documentation during ASD is hardly required, but further research should be carried out to prove this. Although a causal relationship has not been proven, our evidence suggests that an organizational factor – maturity in SPM – influences an agile team in its usage of artefacts. This would be quite contradictory to self-organizing teams, from which the best architectures, requirements, and designs emerge. Our research goes beyond the sole modelling of artefacts and provides initial knowledge about factors that influence agile teams in their artefacts usage.

Our research also strengthens empirical evidence with regard to the usage of artefacts in ASD. Our current findings confirm both artefacts that appeared in the artefact list (Gröber, 2013), but not in the Scrum artefact model (Wagenaar et al., 2015), as well as vice versa. The relatively small yield of 'new' artefacts, proves an already high degree of coverage in the research on the existence of artefacts in ASD.

Further research is necessary, not only to prove a causal relationship between (SPM) maturity and artefacts usage, but also to identify other factors influencing agile teams in their choice for (non-agile) artefacts. Candidates are team composition (size, experience), project characteristics or explicit team decisions as opposed to maturity of an organization as a whole. This would provide an answer to the question whether, especially non-agile, artefacts emerge from an agile team or are used for reasons which originate from outside the team. More general, how does an agile team reach a balance between agile and non-agile artefacts?

CHAPTER 5

Agile team rationales for artefacts usage

Agile software development (ASD) promotes working software over comprehensive documentation. Still, recent research has shown agile teams to use quite a number of artefacts. Whereas some artefacts may be adopted because they are inherently included in an ASD method, an agile team decides itself on the usage of additional artefacts. However, explicit rationales for using them remain unclear. We start off to explore those rationales, and state our primary research question as: What are rationales for agile teams to use artefacts? Our research method was a multiple case study. In 19 agile teams we identified 55 artefacts and concluded that they in general confirm existing research results. We introduce five rationales underlying the usage of artefacts in ASD: (1) Adoption of ASD leads to agile artefacts, (2) team-internal communication leads to functional and technical design artefacts, (3) quality assurance leads to test-related artefacts, (4) agile teams impose governance on their own activities, and (5) external influences impose user-related material. With our contribution we substantiate the theoretical basis of the Agile Manifesto in general and contribute to the current research with regard to the usage of artefacts in ASD in particular. Agile teams themselves may from this research extract guidelines to use more or less comprehensive documentation.

This work was originally published as:

Wagenaar G., Overbeek S., Lucassen G., Brinkkemper S., Schneider K.: Working software over comprehensive documentation – Rationales of agile teams for artefacts usage. Journal of Software Engineering Research and Development 6.1 (2018): 7.

5.1 Introduction

Fifteen years have passed since the Agile Manifesto (Beck et al., 2001) was published. After half of this period, some 7½ years after its inception, the research community had lavished attention on issues related to agile software development (Dybå & Dingsøyr, 2008). Yet at the same time "*exciting research areas that can further our understanding of the effectiveness of agile methods and practices*" (*p.1219*) were still identified, among which not the least one: the 'core' of agile. Research thus continued, but five years later a research gap with regard to the implications of agile information system development on the coordination, collaboration and communication mechanisms within agile teams was still noted (Hummel, 2014).

The Agile Manifesto itself values "*working software over comprehensive documentation*" and emphasizes "*The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*". However, (Stettina & Heijstek, 2011) concluded: "*Agile practitioners do not seem to agree with this agile principle*" (*p. 159*). Instead developers find documentation important but at the same time observe that too little of it is available in their projects.

Despite the fact that documentation usage in ASD has a somewhat 'old-fashioned' connotation, recent research has shown that traditional software development approaches and ASD are being blended in a hybrid approach; among the different combinations, Scrum, the classic Waterfall model, and V-shaped processes account for the majority (Kuhrmann et al., 2016; Theocharis et al., 2015). This result is further supported by a survey outcome, where approximately 75% of the participants answered that they (intentionally) combine different development approaches (Kuhrmann et al., 2017). Models have emerged to describe combinations of traditional and agile methodologies. (Bustamante &

Rincón, 2017) developed the WYDIWYN (What You Define, Is What You Need) model to define agile and traditional methodologies oriented to what a company needs, simplifying the identification, correlation and selection of phases, roles, tasks and work products among different frameworks; (Gill et al., 2016) describe a reference model for hybrid traditional-agile software development methodologies.

More in particular not only traditional and agile software development processes are blended, but also elements from them, especially artefacts. We define an artefact, in line with previous research, as a tangible deliverable produced during software development, including materials in both physical and electronic format (Wagenaar et al., 2015). This definition certainly includes physical artefacts, such as story cards and the Wall (Sharp et al., 2009), but allows in fact for a much broader spectre of artefacts. We join (Sharp et al., 2009) in their description of the connection between documentation and artefacts in ASD as "*Documentation is kept to a minimum, in favour of close collaboration, and simple artefacts*" (*p. 108*). This definition equates (minimum) documentation and simple artefacts, opposing the both to the comprehensive documentation of the Agile Manifesto. Following this observation we prefer using the term 'artefacts' in the remainder of this article; this also allows us to overcome the perceived ravine between agile and old-fashioned documentation.

Recent studies on artefacts usage in ASD shows that agile developers use quite a number of artefacts (Bass, 2016; Gröber, 2013; Liskin, 2015; Wagenaar et al., 2015; Wagenaar, Overbeek, Lucassen, Brinkkemper, & Schneider, 2017). Some of them are inherent to an ASD method, for instance, user story or backlog, but others are not, recalling bygone memories from a Waterfall era, that advocated program design first and documenting it thoroughly, using, for instance, a design and a

test document (Royce, 1970). This by no means implies that the latter, for instance test documents, should not be part of a sound software development process, in fact they should be, but merely that they are not explicitly included as element of an ASD method, for instance, Scrum.

While there is value in comprehensive documentation the authors of the Agile Manifesto value working software more. Documentation functions as a mean of communication in software development in general (Curtis et al., 1988), but its importance is recognized in ASD as well and the agile approach to artefacts may be compared with 'travelling light': "*Create just enough models and documentation to get by*" (Ambler, 2002, *p. 29*). Turk, France. Robert, & Rumpe (2005) also discussed this tension when they formulated a documentation proposition, "*The de-emphasis of documentation as a communication aid is based on an assumption that tacit knowledge is to be valued over externalized knowledge*" (*p. 11*) and mention that this proposition has both proponents and critics. They also observe that "*the agile process community claims that more is gained through informal personal communications than through communication based on formal documentation*" (*p. 10*).

These observations again illustrate the turbid relation between ASD and artefacts. One way to shine light on this relationship is to explicitly investigate in agile teams what artefacts they use and, even more important, why they use them. The need for research in this area was recently confirmed again: "*In-depth insight into agile software engineers' needs for information that can be covered by documentation is still lacking*" (Voigt, Garrel, Müller, & Wirth, 2016, *p. 4.2*). With our research we add another brick in the wall in the research on communication in ASD, more in particular in the extent to which formal communication in the form of artefacts is used.

We formulate our research question as:

What are rationales for agile teams to use artefacts?

With this research question we investigated the rationales behind decisions an agile team take with regard to its usage of artefacts. The answer to this question should reveal how members of an agile team, with the Agile Manifesto in mind, apply one of its statements. Since ASD methods are commonly used in delivering state-of-the-art software (Bustard et al., 2013; Melo et al., 2013; Rodríguez et al., 2012; VersionOne, 2017), it becomes even more important to annotate or elaborate the Agile Manifesto with regard to its use of artefacts.

To answer our research question we performed case studies within 19 organizations to ask representatives of agile teams our above-mentioned question. All organizations were SPOs, involved in software product management (Fricker, 2012). The organizations varied (1) in size, from rather small meaning less than 50 to large with one as large as several thousands and others as several hundreds of employees, (2) in business domains, with finance, customer relationship management, health care, and others, (3) in team size, where in general the 7 ± 2 advice (Schwaber & Beedle, 2002) was followed, but also 10+ teams, and (4) in agile experience, from (over) ten years to recent, with a year order of magnitude.

Our conclusions from the case study articulate that agile teams' rationale for the usage of artefacts has five major elements: (1) artefacts may simply come with ASD as 'prescribed', (2) they provide useful governance for the team, (3) they are useful and/or necessary for internal communication, which is then not face-to-face at all, (4) they are useful and/or necessary for quality reasons, or (5) external parties need it.

With our contribution we substantiate the theoretical basis of the Agile Manifesto. We also add another piece in the puzzle for the hybrid agile / waterfall research arena, when we show that agile teams in practice use all kinds of artefacts, inherent to an ASD method or not. Finally, agile teams themselves may extract guidelines to enlarge or diminish their usage of artefacts.

The remainder of this paper is structured as follows. In Section 5.2 we review related work where we focus on research with regard to the usage of artefacts in ASD in general and its implications for the derivation of a rationale for their usage in particular. In Section 5.3 we present our research method, a multiple case study, including data collection and analysis. Section 5.4 presents our findings. Finally, section 5.5 presents our conclusions, discusses validity of our research and indicates directions for further research.

5.2 Related Work on ASD Artefacts

In the search for hybrid agile/waterfall models the usage of artefacts has received little attention. Adversely, usage of artefacts in ASD alone has recently attracted research attention.

(Gröber, 2013) constructed an agile artefact class diagram with 19 artefacts as result of a systematic literature study on artefacts usage in ASD. The diagram was subsequently extended to a more generic model to abstract it from local, project-specific processes as well as refined on the basis of experiences from practice (Femmer et al., 2014; Kuhrmann et al., 2013). Categories in the diagram are: (1) Project management, (2) Requirements specification, (3) Production process, and (4) Final artefacts (Gröber, 2013). Final artefacts are equated to working software and include source code and Commercial-Of-The-Shelf (COTS) software. Requirement specification artefacts need to be implemented in order to build final artefacts: Feature, requirement, user story and use case. To transform

requirements to a final product, production process artefacts are suggested: Backlogs and their backlog items, including the wall artefact which is used to enact the process as it can be seen as a simplified physical representation of a backlog containing items in the form of index cards. Also parts of the test cases are considered as they drive the production of code. Finally project management artefacts are listed: A release plan, a burn-down chart, and a coding standard.

Classification criteria were used to cluster artefacts in groups. In building the classification a tag cloud was generated on the basis of one of the inclusion criteria of the literature study, namely a 'Classification' column in the criterion "*Study is classifying a number of artifacts with an arbitrary characteristic. (Possible characteristics could be e. g. textual artifacts, test artifacts)*" (Gröber, 2013, *p. 9*). The cloud revealed classification criteria in which artefacts can be grouped with the most used criteria being: code, test, design, requirements, physical, documentation, management, production, intermediate, final, internal and digital. These were the basis for the final classification.

This research lists ASD artefacts on the basis of a classification and thus provides an overview of which artefacts are used in ASD. Although some hints may be derived on the basis of the classification, neither its derivation procedure nor its contents addresses the rationale for usage of the artefacts.

Several studies investigated the usage of agile artefacts in specific domains, such as requirements communication or user centred design. (Liskin, 2015) lists three artefact categories: Container, Individual element, and Solution model. Containers are characterized by their value to hold everything together in one place, for instance a virtual project environment. Individual elements are either user-oriented or technical. Examples are a use case (user-oriented) and a system

requirement (technical element). Finally a solution model illustrates aspects of the future solution in a concrete model, such as a GUI mock-up or an abstract model, such as a data model. (Garcia, Silva da Silva, & Selbach Silveira, 2017) used a systematic mapping study to identify which artefacts are used in order to facilitate the communication in an agile user-centred design approach, but did not classify the resulting artefacts. Artefacts in both studies bear similarities to those in previously mentioned models, but are restricted to artefacts with regard to requirements and user-centred design respectively.

In a study on large-scale offshore software development programmes (Bass, 2016) identified 25 artefacts on five levels of abstraction: Programme governance, Product, Release, Sprint, and Feature. Program governance artefacts are created to coordinate cooperating agile development teams and mitigate risk of development programme failure by providing a layer of oversight and governance; they include risk assessment and several architectural artefacts. The other four levels consist of ever finer-grained artefacts used within a product itself, its releases, its constituting sprints and the features dealt with within a sprint. The research focused on artefacts across the development life cycle but only investigated large-scale agile development programmes. It is argued that the artefacts used in large-scale agile development programmes are a superset of those used in smaller projects.

This study was designed to focus on artefacts to subsequently shed light on the tailoring of agile methods in a large-scale software development programme context. The study's interest was in practitioner interactions with artefacts and not in the artefacts per se, where its classification mechanism was mainly derived from previous research (Femmer et al., 2014; Kuhrmann et al., 2013).

Teams using agile development shall, among others, identify documents to be produced by the process or project (ISO, 2011). Although ISO-standard 26515 primarily focusses on user documentation processes, it at the same time identifies documentation items that should be produced by agile projects. Such life cycle documentation should be produced in projects using agile development to communicate processes, requirements, and deliverables required of the teams working on the project. These documents may contain less detail than their counterparts in other software development methods. Examples include user stories, burn down charts, and test plans.

In summary, previous research enumerated artefacts in several models or lists by using different classifications. The classifications contain elements which could relate to rationales for using them, although none of the models was built with this purpose explicitly in mind. Implicit rationales for the usage of artefacts could be derived, but the question 'Why are you using this artefact?' was never answered by the ultimate source: the agile team itself.

5.3 Research Method

In this section we describe our research method, starting with our study protocol, and the subsequent data collection, as well as the analysis procedure. To investigate our research question we used a holistic multiple-case study with as unit of analysis the agile team and its artefacts; this approach is an accustomed way to investigate phenomena in a context where events cannot be controlled and where the focus is on contemporary events (Yin, 2013). Since research results are scarce with regard to our research question, our aim is theory building rather than theory testing; the former is useful in early stages of research on a topic or when a fresh perspective is needed, while the latter is useful in later stages of knowledge (Eisenhardt, 1989).

5.3.1 Case Study Protocol

Although a case study allows flexible research, this does not mean that planning is unnecessary (Wohlin et al., 2012). A Case Study Protocol (CSP) is the container for the design decisions on the case study as well as for field procedures for carrying through the study. Among a CSP's purposes are guiding data collection and reporting. As an element in guiding data collection, interview questions were drawn up and a reporting structure was also included in our CSP. Elements from the CSP will be used to accompany our description of the data collection process (section 5.3.2).

For our research we formed a research group in which two senior researchers, being two of the authors of this manuscript, coordinated and supervised the activities of the group. Other members of the group were graduate students in preparation for their Master's thesis. If "*no reason exists to preclude most elementary, secondary and university students from becoming critical researchers*" (Kincheloe & Steinberg, 1998, p. 2), then graduate students certainly qualify as members of a research group, provided that they are properly equipped. In our case they were supported by a CSP, i.e., interview guidelines and reporting structure.

5.3.2 Data Collection on Artefact Usage in Agile Teams

Data collection took place through the use of various single-site case studies following widely accepted guidelines for case studies (Yin, 2013). Due to the exploratory nature of the research, semi-structured interviews were used to allow interviewees to speak freely and to be able to ask follow-up questions (Kajornboon, 2005). Data collection took place by members of the research group, under supervision of the senior researchers.

Approximately 80 SPOs were approached by our group, of which 19 organizations with 19 software development teams were willing to participate in our research (Table 5.1). In case an organization had a portfolio of several products, the research focused on one agile team working on one of the products only. In order to qualify for our research, organizations were further required to have their own in-house software development unit, although part of its activities could be outsourced. Within the unit, at least one team had to apply an ASD method.

Table 5.1 - Characteristics of teams/organizations

Id	Domain	Size		Country	Interviewee(s)
		Organization	Team		
A	Software	Large	Medium	NL	Product owner
B	CRM	Small	Small	NL	Scrum master (lead develop architect)
C	CRM	Small	Small / Outsourced	NL / TR	Product owner
D	Industry	Large	Large	NL	Scrum master
E	Software	Small	Large	NL	Product owner (CTO)
F	HRM	Medium	Small	NL	Product owner (lead developer) Product owner
G	E-learning	Small	Small	UK	Product owner
H	Hospitality	Small	Small / Outsourced	NL / MK	Scrum master (project manager)
I	Finance	Large	Medium	NL	Scrum master (tester)
J	Health care	Medium	Small	NL	Lead software developer Senior functional developer
K	ERP	Medium	Small	NL	Developer Product owner Chief executive officer
L	Purchase	Medium	Small / Outsourced	NL / BG	Product owner Chief technical officer
M	Insurance	Large	Small	NL	Development director
N	HRM	Small	Medium	NL	Scrum master (lead developer)
O	Finance	Large	Small	CN	Product owner
P	Finance	Medium	Medium	NL	Scrum master (agile coach)
Q	Health care	Large	Medium	NL	Scrum master (business line manager)
R	Finance	Large	Medium	NL	Scrum master
S	Finance	Small	Small / Outsourced	NL / SP	Senior technical lead developer

Organizations/teams are letter coded with 'Id' for reasons of confidentiality. The size of an organization is considered to be small if it has less than 50 employees, medium between 50 and 250 employees, and large more than 250. For an agile team small indicates less than 7 members, medium between 8 and 11, and large 12 or more.

The organizations and their corresponding agile teams thus vary in business domains, in organization and team size, and in country, although the Netherlands pre-dominates. This is not a surprise, since the vast majority of the organizations we approached was based in the Netherlands in the first place. Our interviewees have in general 'leading' positions, such as product owner, Scrum master, or lead developer. This is mainly caused by our design, in which we favoured interviewing at least one representative with a broad view on the team and its activities.

The data collection procedure according to the CSP was the same for all 19 teams. In each organization interviews were held. Most of the time this was one interview, often with a product manager or owner. In some cases two interviewees were involved and/or two interviews were conducted. Interviews lasted, on average, one hour and were transcribed and/or summarized thereafter.

To allow for the results to be comparable over the single cases we established a common vocabulary in the description of artefacts by using a FLOW model (Stapel et al., 2009; Stapel & Schneider, 2012, 2014). It is a modelling technique containing several elements:

- Documented information is called solid information if it is (1) long term accessible, (2) repeatedly readable, and (3) comprehensive for third parties. In

contrast, undocumented or fluid information is information that violates any one of the above criteria.

- Transfer of documented/undocumented information originates from solid and fluid flows respectively.
- A role describes an actor being the producer or consumer of any form of information. A role can also be an individual person.

The FLOW notation was designed to explicitly visualize the concepts of fluid and solid information and flows (Schneider, Stapel, & Knauss, 2008). The notation is intended for a variety of users and thus reuses well-known concepts from existing notations. Although differences between FLOW and related notations may seem subtle at first, fluid flows are obviously more difficult to represent in other notations which were not tailor-made for them. The emphasis on information and its distinction between solid and fluid information makes a FLOW model suitable for the representation of artefacts.

As an example the model for organization R is provided (Figure 5.1). The model shows documents and people involved. Direct flow of information (fluid) is represented by faces and dashed arrows originating from the people they represent, for instance 'Customers' towards 'Requirements gathering'. Solid arrows originate from documents (solid information, as in, for instance, 'Maintenance backlog'). Rectangles stand for activities and they are treated as black boxes. In the diagram persons participating in an activity are shown. The internal flow of information within an activity, however, is hidden or unknown and not shown here.

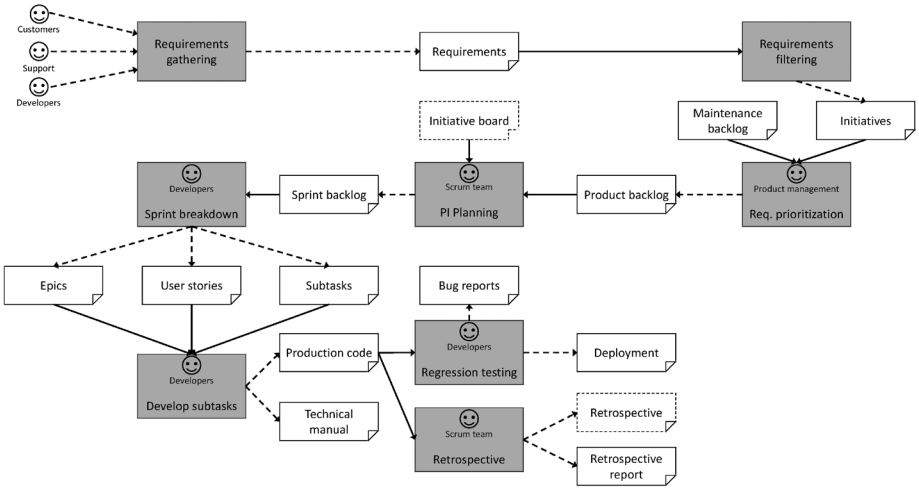

Figure 5.1 - Example of a FLOW model (team R)

Members of the research group transcribed and/or summarized the interviews before visualizing them in a FLOW model. To this extent the members collectively acquainted themselves with relevant FLOW literature and drawing up FLOW models, although for fictitious cases at first. Reporting instructions concerning the FLOW models were also part of the CSP. Besides a FLOW model itself, instructions in the CSP also required a description of solid and fluid information and the rationale for usage of the information. In first instance members of the research group combined results in a draft report. The senior researchers then provided interim feedback, which would sometimes lead to additional data collection. At approximately the same time the draft report was put at the disposal of interviewees to allow for comments with regard to correctness of transforming the interviews to FLOW models and the accompanying descriptions. Both sources of feedback were introduced to assure quality of the final report. An average (final) report comprised 24 pages.

5.3.3 Data Analysis

The 19 FLOW models together with their description of solid and fluid information and the rationale for the usage of the information formed our base data. We started data analysis by extracting solid information as artefacts from the models, and this resulted in 360 artefacts. The list was reduced in a two-step process: (1) a lexical analysis and (2) a semantic analysis (Jurafsky & Martin, 2008).

In the lexical analysis we removed distinctions in singular and plural forms, for instance 'User story' (listed 3 times) and 'User stories' (11 appearances). We removed adjectives, for instance mapped both '(Conceptual) user story' and 'User story' on 'User story'. This reduced the number of 360 to 200 artefacts. In our lexical analysis we primarily used the names of the artefacts and only occasionally and superficially used their descriptions.

In a further semantic analysis we did use the description of artefacts from the FLOW model to identify similarities and differences in artefacts. In this analysis we applied constant comparison (Glaser & Strauss, 1967). This comparison was done manually, proceeding from one artefact to another and working backwards whenever appropriate. For instance, we already identified an artefact 'Bug report' with description '*A bug report is a document that is used to report a bug, an unwanted and/or unintended malfunction in the source code*'. When we found 'Maintenance backlog' described as '*A maintenance backlog records data about the software during runtime, used to identify issues and points of improvement*', we mapped this backlog to the artefact 'Bug report'. As another example we mapped 'Specification' ("*… an extensive description of the requirement*") to 'Functional design' ("*… describes techniques or prescribed methods that need to be implemented in the user stories, as well as properties of the required input and output*"). In the process we deleted artefacts

when they could be equated to another artefact mentioned by the same team or we mapped artefacts when they could be equated to artefacts also mentioned by another team. In both cases we chose one of the applicable descriptions of the artefact or combined them to form a new blended description. This process also applies to the rationales provided for the artefacts.

5.4 Findings

5.4.1 Artefacts

Lexical and semantic analysis resulted in first instance in a list of 55 different artefacts (Table 5.2). An 'x' stands for usage of the artefact (row) by the agile team (column).

Table 5.2 - Overview of artefacts

Artefact	Agile team ID																		# occurrences
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
Acceptance criteria														x		x				2
Architecture standard	x						x						x							3
Bug report	x	x	x	x			x	x		x	x	x	x		x			x		12
Burndown chart	x	x	x	x		x						x			x		x			8
Business case					x									x			x			3
Business plan			x			x				x			x							4
Coding standard																			x	1
Contact																			x	1
Contract	x												x				x			3
Definition of done								x									x			2
Definition of ready								x												1
Deployment script												x								1
Epic	x				x	x						x					x	x		6
Functional design	x		x			x		x		x								x	x	7
Legislation/Regulation										x			x							2
Impact analysis			x					x												2
Implementation guide											x									1
ISO standards												x								1
Market analysis										x					x					2

Table 5.2 - Continued

Artefact	Agile team ID																			# occurrences
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
Market requirement					x	x		x	x	x	x	x	x	x	x	x				11
Mock-up							x					x					x			3
Product backlog	x	x	x			x	x	x	x		x	x	x	x	x	x	x	x	x	16
Product portfolio													x							1
Product requirement						x			x	x		x	x	x	x	x				8
Prototype																	x			1
Quotation					x															1
Release				x		x	x	x			x			x				x		7
Release checklist	x																x			2
Release log																			x	1
Release note	x	x		x		x	x			x	x	x		x		x	x		x	12
Release plan	x			x				x					x							4
Request for information						x														1
Requirement	x	x	x	x	x	x	x	x	x	x		x	x	x		x	x	x		2
Retrospective report									x									x		2
Risk assessment			x			x							x			x	x			5
Roadmap	x		x				x	x	x	x	x		x				x		x	10
Source code	x	x	x	x	x	x	x	x	x		x		x	x	x	x	x	x	x	17
Sprint backlog	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x		18

Table 5.2 - Continued

Artefact	Agile team ID																		# occurrences
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	
Status board		x	x			x	x	x											x	6
Task								x	x									x		3
Technical design			x											x						2
Technical documentation	x	x				x						x						x		5
Technical requirement	x							x				x								3
Technical roadmap										x										1
Template functional description																			x	1
Test case			x								x	x			x					4
Test criteria		x																		1
Test plan	x					x								x		x				4
Test report		x		x	x		x	x						x			x			7
Unit test report		x				x											x			3
Use case						x	x										x			3
User acceptance test	x	x				x											x			4
User manual										x	x				x				x	4
User story	x	x	x	x	x	x		x	x	x	x	x	x			x	x	x		15
Vision/Strategy			x					x					x			x				4

The artefacts that are mentioned most (top 5) are: Sprint backlog (18), Source code (17), Product backlog (16), User story (15), and Release note (12). This shows a pre-dominance of artefacts directly related to an ASD method.

5.4.2 Influence of Agile Team on Artefact Usage

All artefacts from Table 5.2 are used by at least one agile team in our case study. However, not all of them are used as a choice of an agile team itself. To distinguish between usage in general on the one hand and usage as result of an agile team's decision on the other hand, we use IEEE standard 1074-2006, which provides a process for creating a software project life cycle process (IEEE, 2006). In particular, the standard distinguishes the following phases in the software development-oriented processes:

- Pre-development process.
- Development process, to be further divided into requirements process, design process, and implementation process.
- Post-development process.

We present our list of artefacts according to this distinction (Table 5.3). As a consequence of this categorization we exclude pre-development artefacts from our consideration. This is in line with our research question which is related to the rationales of an agile team for artefacts usage. Artefacts that are not under an agile team's influence are therefore not contributing to an answer to our research question. The exclusion is neither a negation of the existence of such artefacts nor of their influence on some aspects of ASD by an agile team, but pre-development artefacts usage is decided upon outside its realm. Nevertheless, particularly a product owner (Schwaber & Sutherland, 2016) or product manager (Ebert & Brinkkemper, 2014) may, as an agile team member,

be partially involved in construction of pre-development artefacts. Yet these artefacts cannot be attributed to an effort of an agile team as they do in fact hardly directly contribute to the agile team's activities.

Our limitation is also reflected in rationales which are given for usage of the pre-development artefacts. As a first example we present some rationales for the usage of pre-development artefacts:

- "*Organization L needs to comply to ISO standards in order to have Service Level Agreements (SLAs) with customers and suppliers.*"
- "*The market analysis […] is required by internal stakeholders to have an overview of the product.*" (organization O)
- "*Without the contract, it is not possible to develop a software product for the customer at all.*" (organization Q)

Throughout this and the next section citations may have been slightly edited, as compared to the original statements, to improve readability.

As a second example we present some rationales for the use of 'Roadmap':

- "*Product roadmaps stimulates thinking over the immediate and future evolutions of the product.*" (organization G)
- "*The roadmap provides the team with a clear direction in which they should develop the product, in accordance with management.*" (organization I)
- "*The roadmap provides important information to the stakeholders about future plans.*" (organization S)

Especially the demarcation line between pre-development and development is a delicate one, because some artefacts, especially requirement artefacts,

seem to fit on either side of the line. Development includes a requirements process. Yet we classified some forms of requirements as belonging to the pre-development process, which are market requirement, product requirement and requirement in general. The organizations in our research were all SPOs and in this context market requirements are requirements as they are formulated by external stakeholders. Product requirements are market requirements which are adopted for inclusion in a future release of the software product. The term 'requirement' in general is in some cases used for market requirement, sometimes for product requirement or for both. We decided to categorize them as part of pre-development, since the involvement of an agile team in their formulation is limited, apart from some contribution by, mostly, a product owner. Also, requirements do still appear in (the requirement process within) development in the shape of epics and user stories.

Table 5.3 - Artefacts in software development-oriented processes

Pre-development	Development	Post-development
Business case	Acceptance criteria	Deployment script
Business plan	Architecture standard	Implementation guide
Contact	Bug report	Release
Contract	Burndown chart	Release checklist
Legislation/Regulation	Coding standard	Release log
ISO standards	Definition of done	Release note
Market analysis	Definition of ready	User manual
Market requirement	Epic	
Product portfolio	Functional design	
Product requirement	Impact analysis	
Quotation	Mock-up	
Release plan	Product backlog	
Requirement	Retrospective report	
Request for information	Source code	
Risk assessment	Sprint backlog	
Roadmap	Status board	
Vision/Strategy	Task	
	Technical design	
	Technical documentation	
	Technical requirement	
	Technical roadmap	
	Template functional description	
	Test case	
	Test criteria	
	Test plan	
	Test report	
	Unit test report	
	Use case	
	User acceptance test	
	User story	

The involvement of an agile team focusses on the development activities, with participation in post-development activities, where the latter depends heavily on the level of implementation of DevOps, Development-Operations (Dyck, Penners, & Lichter, 2015).

5.4.3 Rationales for Artefacts Usage

Having identified development and post-development artefacts as artefacts for which an agile team is involved in their usage, we now turn an exploration of their rationales. We will introduce five groups of rationales: Agility, Governance, Quality Assurance, Internal communication and External follow-up. Our choice of just those categories is inspired by the work of (Grant, 1996) and (Strode & Huff, 2014). Grant points to four mechanisms for integrating specialized knowledge:

- Rules and directives are (impersonal) approaches to coordination where rules may be viewed as standards which regulate the interactions between individuals.
- Sequencing is a means by which individuals can integrate their specialist knowledge by organizing activities in a time-patterned sequence.
- Routines may be simple sequences, their interesting feature is their ability to support complex patterns of interactions between individuals in the absence of rules, directives, or even significant verbal communication.
- Group problem solving and decision making supplements all the above mechanisms by recognizing that some tasks may require more personal and communication-intensive forms of integration.

(Strode & Huff, 2014) distinguished categories of artefacts: Synchronization and boundary spanning. Synchronisation artefacts are produced during

synchronisation activities that contain information used by all team members in accomplishing their work. These artefacts include, among others, working software, a product backlog, and stories. Boundary spanning artefacts are physical things produced to support boundary-spanning activities and enable coordination beyond the team and project boundaries. These could be a project management plan for a project management office or a Request for Change form for an IT support unit when additional servers were required. Some similarities with the product/process dimension of the previous research may be discovered. Although this research explicitly concerns reasons for artefacts usage, i.e. for synchronisation or boundary spanning, it at the same time lacks artefacts involved, other than in a context of examples. This research builds on a long tradition of the role of artefacts in communication, for instance in Computer Supported Cooperative Work, CSCW (Dix, 1994) or ASD (Sharp & Robinson, 2010).

We will now describe the rationales in more detail. In addition to the description we also indicate a group's relation to the base material and/or to the literature mentioned above.

5.4.3.1 Agility

When inspecting rationales for the development artefacts a first group arises for which we can identify an 'agility rationale'. We use this term to describe rationales which explain the usage of artefacts as a result of the adoption of an ASD in the first place. Sometimes the rationale for such an artefact was worded in a straightforward manner:

- "*User stories are needed to describe features when using SCRUM.*" (team E)

Others were more covert, but upon inspection they correspond well with other sources. For instance, an 'official' description of a product backlog taken from (Schwaber & Sutherland, 2013, *p. 12-13*) is: "*The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product*". This coincides well with rationales which are given for its use:

- "*The backlog is used to store the product requirements*." (team A)
- "*The product backlog serves as a way of keeping track what has been changed and implemented during a sprint; it is important, because it records everything that was implemented, including reasons and dates and is thus available for reference*." (team I)
- "*The product backlog keeps track of all the user stories associated with the product*." (team B)

Several artefacts share this mutual rationale. Burndown chart, Definition of done, Definition of ready, Epic, Product backlog, Retrospective (report), Source code (Increment), Sprint backlog, Status board, Task, and User story: They all come with the adoption of ASD. This is also the reason for referring to them as agile artefacts, as opposed to non-agile artefacts which are then in fact all other artefacts from Table 5.3, pre-development artefacts excluded.

The rationale for these artefacts arose in a natural way from our base material. In terms of Grant the rationale reflects a kind of rule/directive, where Strode would consider them as artefacts used for synchronisation.

5.4.3.2 Governance

Some artefacts are motivated by governance an agile team adopts itself. Of course some standards are imposed externally or organizationally, where the

latter refers to standards within the organization, but outside a team's influence. Examples of this are legislation/regulation or ISO standards, but they were considered to be pre-development artefacts and thus outside the influence of an agile team. Governance here applies only to rationales for which the team itself decides on their use. They include:

- "*Architecture standards describe requirements for the platform for which a feature is being developed, as well as provide guidelines for code standards and style. This is important since all code within the project needs to be consistent and the architecture of the projects needs to be standardized to facilitate future development.*" (team A)
- "*A definition of ready describes what a user story must adhere to, before development can start. It contains a checklist for a requirement and relates to what makes a good user story.*" (team H)
- "*A coding standard is used to write good code and to check if code is written properly.*" (team S)
- "*The template (functional description) helps the product manager in writing a functional description.*" (team S)

The governance rationale clusters acceptance criteria, architecture standard, coding standard, definitions of done & ready, and template functional description. From this enumeration it shows that some rationales may apply to multiple artefacts. For example, the definition of done was also motivated by the rationale that it comes with the adoption of an ASD.

This rationale was inspired by the notion of Grant's routines, although in its elaboration it is in fact a mixture between rules and routines. There are no predefined rules or directives, but some teams more or less decided to still formalize their routines in a rule-based way. With this viewpoint the mentioned

artefacts were unified under the governance rationale. For Strode they would again be synchronization artefacts.

5.4.3.3 Quality assurance

Testing is for a long time considered to be an important quality aspect of software quality assurance: "*The responsibilities of the quality assurance activity generally include: … Test Surveillance - Reporting of software problems, analysis of error causes and assurance of corrective action*" (B. W. Boehm, Brown, & Lipow, 1976, *p. 601*). This is still recognized by agile teams today as they motivate their use of test-related artefacts:

- "*These (unit) tests are performed to ensure the quality of the programmed code and eventually the products.*" (team B).
- "*The test plan ensures that all delivered software is of high quality and captures the intended functional behaviour.*" (team F)
- "*Test results ensure the correct- and completeness of a new piece of code. If a new piece is not correct and/or complete the result indicates what is wrong and the problem is easy to find for the developer.*" (team N)
- "*For documentation and proof purpose, bug reports are created to give an insight on how maintenance on the software is performed*." (team L)

Bug report, test case, test criteria, test plan, test report, unit test report, and user acceptance test all share a rationale in software quality assurance.

This rationale finds its origin mainly in the base material. It does not have a direct counterpart in neither Grant nor Strode.

5.4.3.4 Internal communication

As for testing, design is also for a long time considered to contribute to especially the efficiency of the software development process. Some (initial) steps to transform a risky development process into one that will provide the desired product: Program design comes first (step 1) and document the design thereafter (step 2) (Royce, 1970). This statement has not lost its value in agile times: "*Fundamentals of XP include starting a project with a simple design …*" (Beck, 1999) or explicit Functional Model and Design (& Build) iterations in DSDM (Dynamic System Development Method) (Stapleton, 1997). Of course, this applies especially to evolutionary design, as opposed to big design up front. However, in both cases: "*First keep in mind what you're drawing the diagrams for. The primary value is communication*" (Fowler, 2004). Agile teams value this type of communication:

- "*The design flow allows the developer to understand how a newly developed function is going to be tailored inside the product.*" (team A)
- "*For the user experience, visualizing the features (screens) of the software product is better than just a textual description.*" (team Q)
- "*The functional design improves communication and understanding of how new requirements fit into the overall software product.*" (team F)
- "*At organization H, a technical requirement describes how software should be designed from a technical standpoint; it is usually submitted by colleagues of the development team and they are gathered to improve the product specifically on technical aspects.*"

Functional design, impact analysis, mock-up, technical design, technical documentation, technical requirement, technical roadmap, and use case are several types of design that are used for communication purposes.

This rationale directly links to Grant's group problem solving as well as to Strode's boundary spanning, where in the latter case it is not crossing the boundary of the team as well expanding the range of an individual or a subgroup.

5.4.3.5 External follow-up

The rationale for the usage of post-development artefacts is somewhat different from the other categories. In fact their usage is not so much a decision of an agile team itself. These artefacts are especially appreciated by external parties, outside the agile team. This may be a customer, external to the organization as a whole, or another part of the organization, for instance Operations. However, the team certainly is involved with them, especially as their producer. This is reflected in the found rationales:

- "*Release notes are continuously added by the team and are a low effort approach to documenting each increment and release, because by adding small release notes to every commit, Visual Studio Team Services can generate an overview per increment or per release.*" (team K)
- "*Every product increment must be documented with a release note.*" (team L)
- "*User manual is required by external and internal stakeholders to know the new features of the product.*" (team O)

All post-development artefacts, Deployment script, Implementation guide, Release, Release checklist, Release log, Release note, and User manual are headed under this rationale.

This rationale followed from our base material, although a modest link with Strode's boundary spanning might be argued in the sense that these artefacts cross the border between the team and a part of the outside world.

5.5 Conclusions & Discussion

We first summarize our findings with regard to rationales for artefacts usage in agile teams. Figure 5.2 shows the usage of artefacts (boxes) categorized by the rationale for their usage (ellipses).

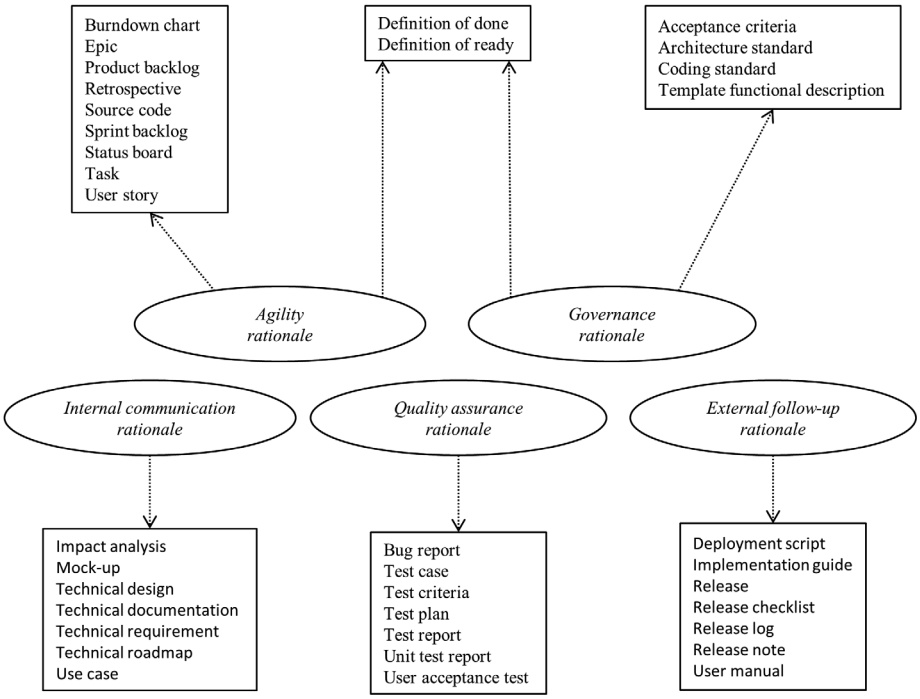

Figure 5.2 - Categorization of rationales for artefacts usage

We started our research from the observation that recent research reports that organizations blend waterfall and agile system development methods in all kinds of hybrid development varieties. At the same time, we noticed that recent research devotes attention to the usage artefacts in ASD. With the Agile Manifesto's valuation of working software over comprehensive documentation

we phrased our research question as: 'What are rationales for agile teams to use artefacts?'

5.5.4 Conclusions & Discussion

In our findings we consider our overview of 55 artefacts to be near to complete. When we compare the artefacts in Figure 5.2 with previous research on artefact models (Bass, 2016; Gröber, 2013; Wagenaar et al., 2015), we encounter many similarities. Differences occur, but seem to be more a result of choices with regard to granularity rather than fundamental differences.

ISO standard 26515 also addresses documentation items should be produced by projects using agile development to assist both the production of software and user documentation: project and sprint plan, requirements documents (user stories), test plan, risk statement, use case, descriptions of persona, burndown chart, task list, Scrum report, and lessons learned report. Comparing them to our findings only two of them are not directly recognizable from Figure 5.2: project plan and description of persona. With the project plan including the sprints to be developed and any milestones such as dates to provide the software and user documentation, it bears resembles to our release plan and/or the product backlog. The description of personas, as a form of user experience design, is not explicitly visible in our findings, but may be thought to be included in a functional and/or technical design. Probably not being consciously aware of the standard, because never mentioned explicitly, the teams nevertheless cover almost the entire list of documentation items form the ISO standard.

Proceeding from the artefacts in our case study we also retrieved rationales for agile teams' usage of artefacts and draw our conclusions on this basis.

Adoption of ASD leads to agile artefacts. The main rationale for using agile artefacts, for example, epics, a sprint backlog or a definition of done, is not surprising. This usage is directly motivated by the adoption an ASD method. An overwhelming majority of our teams uses artefacts for mainly this reason. Artefact usage as a result of this rationale dominates the list of all artefacts (Table 5.2); over one third of all artefacts (94 out of 254 occurrences) are used because of this rationale.

Team-internal communication leads to functional and technical design artefacts. Under the rationale of internal communication we found that many of our agile teams, on their own initiative, decided to use various additional, other than being inherent to an ASD method, artefacts with examples like functional designs, mock-ups, and technical designs. Teams introduce them to allow communication between members of a team, for instance from a user story on the product or sprint backlog (product owner) through a functional design (designer) to source code (developer). We conclude that agile teams, despite the Agile Manifesto's preference for face-to-face communication, have rationales for this usage of artefacts. This rationale is reflected in the use of boundary spanning artefacts (Strode & Huff, 2014), but not for coordination beyond the project team and its boundaries as for coordination between different disciplines, for instance requirement engineering and programming, within the team.

Quality assurance leads to test-related artefacts. Under the rationale of quality assurance we found agile teams to use quite some test-related artefacts, including, for instance, bug reports, test plans, and (unit) test reports. In fact, only two out of our nineteen teams did not use any of the artefacts motivated by this rationale (Table 5.2). Furthermore, the use of test-related artefacts is in fact easily combined with ASD, as in, for instance, Test-Driven Development (TDD);

"*Test-driven development is a set of techniques that any software engineer can follow, which encourages simple designs and test suites that inspire confidence*" (Beck, 2003, *p. xix*). However, another widely used method, Scrum, does not explicitly describe test artefacts. This rationale re-emphasizes the importance of basic principles which provide keys to a successful software effort (Boehm, 1983).

Agile teams impose governance on their own activities. Agile teams see the benefit of applying governance to their own activities. Architecture and coding standards are for instance constituted to ensure a team's conformity to team wide agreements. This rationale would also particularly be recognized by agile teams using Scrum as ASD: "*Self-organizing teams choose how best to accomplish their work*" (Schwaber & Sutherland, 2016, *p. 5*). Both this governance rationale as well as the internal communication rationale confirm items on the contemporary research agenda for large-scale agile software development: Architecture and Inter-team coordination (Dingsøyr & Moe, 2014).

External influences impose user-related material. It may be put up for dispute if user-related material really is constituted as a result of a decision of an agile team itself. However, an agile team certainly is involved in its production. And the more an agile team operates according to DevOps, the more it benefits from its own artefacts, as, for instance, a deployment script or a release checklist. And a release note, while primarily meant for customers, could also serve as a team's memory.

Usage of artefacts may or may not contradict the Agile Manifesto. Travelling light has often been taken as a motto to describe this dilemma. It could be argued that each artefact de-agilizes ASD. But whatever position is taken in this

dispute, knowing rationales for the usage of artefacts contributes to clarification on the issue.

5.5.5 Limitations

We explicitly consider our multiple case study as exploratory. This has two main reasons. Although we collected data for 19 teams we did not collect data from every individual team member. Rationales provided by our interviewees did not necessarily reflect the opinion of an entire team. Furthermore, as a second reason, deriving groups of rationales from individual rationales is not a process in which every single step is made on the basis of a one-by-one correspondence. We first had to combine descriptions and rationales for artefacts as artefacts (source artefacts) were mapped to others (destination artefacts), but still their descriptions and rationales had to be combined in one description c.q. rationale. Secondly, we also had to integrate descriptions and rationales for artefacts which were mentioned by several teams. Although we structured and documented this process as much as possible, its chain cannot be considered to be flawless.

Our exclusion of pre-development artefacts, especially the ones which are associated with requirements in one or another format, could be argued. In defence of our choice we note that teams that mentioned the artefacts market requirement, product requirement, and/or requirement also mention at least one of the artefacts epic, user story, product backlog, or sprint backlog. In this way rationales for agile requirement artefacts are still accounted for those teams.

5.5.6 Case Study Validity

Validity of a (multiple) case study in general depends on four criteria: construct validity, internal and external validity, and reliability (Yin, 2013).

5.5.6.6 Construct validity

Construct validity identifies operational measures for the concepts under study. To enhance construct validity (1) key informants should review draft case study reports, (2) multiple sources of evidence should be used, and (3) a chain of evidence should be established. We addressed all three: (1) For each interview summaries were drafted, including a FLOW model, and interviewee(s) were able to comment on them, (2) within some teams, but certainly not all, interviews were held with more than one interviewee so in those cases viewpoints could be complemented, and (3) we followed a strict procedure in proceeding from an interview protocol, via a FLOW model and its artefact descriptions, to the list of artefacts. Nevertheless, teams were not visited by the same members of the research group, so interpretation may have influenced especially the construction of FLOW models. Also, when only one interviewee was involved, personal opinions may have influenced our results.

5.5.6.7 Internal validity

Internal validity is considered mainly a concern for explanatory case studies. We do not claim our case study to be explanatory, it is rather exploratory. Still we considered internal validity in translating oral interviews via the FLOW models to an artefacts list. Clustering rationales, as described in Section 5.4, was, although structured, to a great extent exploratory.

5.5.6.8 External validity

External validity defines the domain to which a case study's findings can be generalized. The use of replication logic is listed as the main guarantee for this. Using various single case studies on the basis of a common procedure, as in our research, in general already contributes to external validity, and thus to generalizability of results. However, our organizations being SPOs may have

influenced the usage of artefacts, especially in the post-development process. Generalizability to non-SPOs is therefore limited. Even so, although our research included teams with different characteristics (domain, country, and size) we cannot claim them to be representative for every agile team. We also recognize a geographical bias with dominance of the Netherlands.

5.5.6.9 Reliability

Reliability should demonstrate that the study can be repeated. Using an interview guideline, instructions for FLOW modelling, and formatting results as well as the use of a case study repository contributed to reliability.

5.5.7 Future research

Our research was exploratory and we found directions to group rationales with respect to the use of artefacts. However, to allow for more robust conclusions, viewpoints from several team members should be taken into account. This would require in-depth interviews with members from one team, thus focusing on depth, rather than breadth. This would also strengthen the evidence for the current group of rationales.

As we now concluded face-to-face communication is certainly supplemented with artefacts. Still, another interesting question is: What is being discussed face-to-face and why? This should then go beyond 'standard' meetings as sprint planning or daily stand-up from Scrum.

Finally, our current study did not explicitly investigate characteristics nor context of teams and organizations. As in other areas, for instance the choice for a software development methodology in general (Vijayasarathy & Butler, 2016), context does of course matter in the rationales for artefacts usage. Incorporation

of such a context would also allow for more concrete definitions, templates or examples of artefacts.

PART III

Communication in agile teams beyond artefacts usage

CHAPTER 6

Formality of communication in agile teams

Agile Software Development (ASD) is often thought of as to predominantly use face-to-face communication. Research over the last years already led to adjustment of this conception. ASD also uses formal communication where it blends both traditional and agile artefacts. A next step should then be to explore the entire spectrum from informal to formal communication. This raised our research question: Which artefacts do agile teams use in between formal communication with artefacts and informal face-to-face communication? In an exploratory multiple single-case study we found 38 agile teams in as many organizations willing to participate in answering our question. The findings from our case study confirm the existence of what we called fuzzy artefacts, concepts which are not formally documented but are of such a pervasive nature that an agile team is well aware of their existence. We found in total 73 individual fuzzy artefacts in 38 agile teams. Most of them are related to requirements. Populating a product or sprint backlog with user stories equipped with priorities and efforts is another area where they were found. In this way we extended the typology of formal communication (artefacts) and informal communication (face-to-face) in ASD with an intermediate concept: fuzzy artefacts.

This work was originally published as:

Wagenaar G., Overbeek S., Brinkkemper S.: Fuzzy Artefacts: Formality of Communication in Agile Teams. Accepted for: International Conference on the Quality of Information and Communications Technology (QUATIC 2018), Coimbra, Portugal

6.1 Introduction

Methods for ASD have in common a profound focus on communication, supported by one of the principles of the agile manifesto: "*The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*" (Beck et al., 2001). Emphasis on communication in software development does not come as a surprise, since a field study on software design processes for large systems already indicated that developing large software systems must be treated, at least in part, as a learning, communication, and negotiation process (Curtis et al., 1988).

However, a structured, systematic literature review on the role of communication in ASD stated "*although communication is mentioned repeatedly as one of the most fundamental aspects of agile SD projects, our review shows that our current state of knowledge on the precise role of communication in agile SD and its impact on SD success is limited because previous results are scattered, inconclusive, as well as contradictory*" (Hummel, Rosenkranz, & Holten, 2013, *p. 349*).

Communication in software development can be divided into two types: formal and informal communication (Pikkarainen et al., 2008). Formal communication refers to explicit communication such as specification documents, and status review meetings; informal communication refers to explicit communication via conversations among the workers in the companies (Herbsleb & Mockus, 2003; Kraut & Streeter, 1995). Recent research shows that ASD in its formal communication blends both traditional and agile artefacts, where agile teams do use traditional artefacts (Bass, 2016; Gröber, 2013; Wagenaar et al., 2015, 2017).

Formal and informal communication are still often regarded as two end points on a measuring scale, where ASD does not only include the one, but also the

other (Pikkarainen et al., 2008). From two stakeholders wishing to communicate effectively the one values formal, written communication while the other values informal, verbal communication (Gregory, Barroca, Sharp, Deshpande, & Taylor, 2016). In software process improvement (SPI) the same dichotomy is used: "*Knowledge transfer in agile SPI is based on communication face-to-face … Traditional approaches consider that knowledge … is based on training and document use*" (Santana, Queiroz, Vasconcelos, & Gusmao, 2015, *p. 330*). The two resemble the distinction between explicit and tacit knowledge (Nonaka, 1994; Polanyi & Sen, 2009) represented by artefacts and face-to-face conversation respectively. In other words, ASD "*will focus mainly on knowledge management activities related to tacit knowledge, while the traditional development processes will need activities related to explicit knowledge*" (Bjørnson & Dingsøyr, 2008, *p. 1064*). However, the distinction between tacit and explicit is recognized not to be a black and white one: ""*tacitness*" … *is a matter of degree*" (Nelson & Winter, 1982, *p. 78*). (Cowan, David, & Foray, 2000) also supports the idea of this continuum between tacit and explicit.

This observation raises the question whether or not agile teams in their communication also experience tacitness as a matter of degree, especially in an intermediate appearance between formal communication (artefacts) and informal communication (face-to-face). We will coin this appearance 'fuzzy artefact' in this paper to emphasize its duality. The term is inspired by the notion of 'fuzzy sets': "a "class" with a continuum of grades of membership" (Zadeh, 1965, *p. 339*). We define a fuzzy artefact as an artefact which is not formally documented, but one that is still explicitly recognized by an agile team. In other words, when asked, agile team members mention fuzzy artefacts as playing a role in their ASD whilst at the same time acknowledging that there is no official record of them. Although fuzziness in general could include numerically expressing fuzziness

as a value, between 0 and 1 for instance, we restrict ourselves for the moment to a binary scale in establishing a proof of their existence: Do agile teams use fuzzy artefacts or not?

This notion of artefacts in between formal communication with artefacts and informal face-to-face communication in ASD has not been raised before. The confirmation of their existence would, from a theoretical point of view, be a useful contribution to the precise role of communication in ASD, especially in adding a third mark in a typology of, previously, formal and informal communication. Where it has already been shown that ASD blends 'traditional' and agile artefacts, it would now also become clear that agile teams do in fact use a spectrum of communication and not only face-to-face communication and formal artefacts. Such findings would show to practitioners that the use of other ways than face-to-face communication is a valid interpretation of an agile way of working.

We formulate our research question as:

> *Which fuzzy artefacts do agile teams use in between formal communication with artefacts and informal face-to-face communication?*

In order to answer this question, we set up a multiple single-case study and we found 38 agile teams in as many organizations willing to participate in our research.

The findings from our case study confirm the existence of fuzzy artefacts. Agile teams do use concepts which are not formally documented but are of such a pervasive nature that a team is well aware of their existence. We identified in total 73 individual fuzzy artefacts in 38 agile teams which could be classified under eleven different headings. We thus show that, in addition to already

knowing that agile teams use both formal and informal communication, they also communicate in a neither formal nor informal way with fuzzy artefacts. Tacitness thus indeed is a matter of degree.

The remainder of this paper is structured as follows. In Section 6.2 we present an exploratory multiple single-case study as our research method, including the procedure for data collection and analysis. Section 6.3 presents the fuzzy artefacts we discovered in 38 agile teams involved in our research. Discussion of the four most prominent categories in our findings takes place in section 6.4, including a review of related work. Section 6.5 presents our conclusions, discusses validity of our research and finally indicates directions for further research.

6.2 Research Method

We used an exploratory multiple single-case study to answer our research question; its unit of analysis was an agile team and its concepts between tacit and explicit knowledge. This approach is an accustomed way to investigate phenomena in a context where events cannot be controlled and where the focus is on contemporary events (Yin, 2013). Our aim is theory building rather than theory testing; the former is considered useful in early stages of research on a topic or when a fresh perspective is needed, while the latter is useful in later stages (Eisenhardt, 1989).

6.2.1 Case Study Protocol

Although a case study allows flexible research, this does not mean that planning is unnecessary (Wohlin et al., 2012). A CSP is the container for design decisions on the case study as well as for field procedures for carrying through the study. One of a CSP's purpose is guiding data collection. Our CSP guided data collection

by drawing up interview questions in advance. Another purpose, guiding reporting, was also addressed by prescribing a formal reporting structure.

For our research we formed a research group. In this group two senior researchers coordinated and supervised activities of other members of the group, which were graduate students in preparation for their Master's thesis. If "*no reason exists to preclude most elementary, secondary and university students from becoming critical researchers*" (Kincheloe & Steinberg, 1998, *p. 2*), then graduate students certainly qualify as members of a research group, provided that they are properly equipped. In our case they were supported by a CSP, i.e., interview guidelines and reporting structure.

6.2.2 Data Collection

Data collection took place through the use of various single-site case studies following widely accepted guidelines for case studies (Yin, 2013). Due to the exploratory nature of the research, semi-structured interviews were used to allow interviewees to speak freely and to be able to ask follow-up questions (Kajornboon, 2005). Data collection took place by members of the research group, under supervision of the senior researchers.

Over the period 2016 - 2017 over 100 organizations were approached. All organizations were SPOs, which means the organizations were responsible for the production of at least one software product, where a software product is a "*product whose primary component is software*" (Kittlaus & Clough, 2009, *p. 219*). We found 38 organizations with as many ASD teams willing to participate in our research. In case an organization had a portfolio of several products, the research focused on one agile team working on one of the products only. In order to qualify for our research, organizations were further required to have

their own in-house software development unit, although part of its activities could be outsourced.

The organizations and their corresponding agile teams varied in business domains, in organization and team size, and in country, although the Netherlands clearly predominated (Table 6.1). Organizations/teams are coded with 'Id' for reasons of confidentiality. The size of an organization is small if it has less than 50 employees, medium between 50 and 250 employees, and large more than 250. The country is the country of origin; in case two countries are mentioned this indicates that part of the agile team is outsourced to the second country. Our interviewees in general held 'leading' positions, such as product owner, Scrum master, or lead developer. This is mainly caused by our design, in which we favoured interviewing at least one representative with a broad view on the agile team and its activities.

Table 6.1 - Characteristics of case study organizations

Id	Domain	Organization size	Country	Interviewee(s)
A1	Government	Small	NL	Product owners (2 x)
A2	Software product	Small	NL	Product owner
A3	Government	Large	NL	Product owner
A4	Various	Small	NL	CEO
A5	Public	Medium	SI	CEO
A6	Government	Large	NL	Product owners (2 x)
A7	Marketing	Large	NL	Team leader & Product owner
A8	CRM	Medium	BR	Product owner
A9	Software product	Small	NL	Product owner
A10	Health care	Medium	NL	Product owners (2 x)
A11	Software product	Medium	NL	CEO
A12	Government	Medium	NL	Product owner
A13	Various	Medium	NL	Software department leader
A14	Software product	Medium	NL	Product owner

Table 6.1 - Continued

Id	Domain	Organization size	Country	Interviewee(s)
A15	Science	Small	NL	Product owner
A16	Retail	Large	NL	Product owner
A17	Financial	Large	NL	Product owner
A18	Health care	Small	NL	Business analyst
A19	Various	Small	NL	Product owner
B1	Software	Large	NL	Product owner
B2	CRM	Small	NL	Scrum master (lead develop architect)
B3	CRM	Small	NL / TR	Product owner
B4	Industry	Large	NL	Scrum master
B5	Software	Small	NL	Product owner (CTO)
B6	HRM	Medium	NL	Product owner (lead developer) & Product owner
B7	E-learning	Small	UK	Product owner
B8	Hospitality	Small	NL / MK	Scrum master (project manager)
B9	Finance	Large	NL	Scrum master (tester)
B10	Health care	Medium	NL	Lead software developer Senior functional developer
B11	ERP	Medium	NL	Developer & Product owner & Chief executive officer
B12	Purchase	Medium	NL / BG	Product owner & Chief technical officer
B13	Insurance	Large	NL	Development director
B14	HRM	Small	NL	Scrum master (lead developer)
B15	Finance	Large	CN	Product owner
B16	Finance	Medium	NL	Scrum master (agile coach)
B17	Health care	Large	NL	Scrum master (business line manager)
B18	Finance	Large	NL	Scrum master
B19	Finance	Small	NL / SP	Senior technical lead developer

The data collection procedure according to the CSP was the same for all organizations. In each organization interviews were held. Most of the time this was one interview, often with a product manager or owner. In some cases, two interviewees were involved and/or two interviews were conducted. Interviews lasted, on average, one hour and were transcribed and/or summarized thereafter. To allow for results to be comparable over single cases we established a common vocabulary by using a FLOW model (Stapel et al., 2009; Stapel & Schneider, 2012, 2014). Its emphasis on information and its distinction between solid and fluid information especially, makes a FLOW model a suitable representation technique for our research question. A FLOW model contains, among others:

- Documented information which is called solid information if it is (1) long term accessible, (2) repeatedly readable, and (3) comprehensive for third parties. In contrast, undocumented or fluid information is information that violates any one of the above criteria.
- Roles which describe actors as being the producer or consumer of any form of information. A role can also be an individual person.

An excerpt of a FLOW model, in this case for organization B18 (see Table 6.1), illustrates these notions (Figure 6.1). The activity 'Sprint review meeting' involves the roles Development (team), Product Manager (PM), and CEO and it produces unfinished backlog items as documented (solid) information and lessons learned as undocumented (fluid) information to the product board.

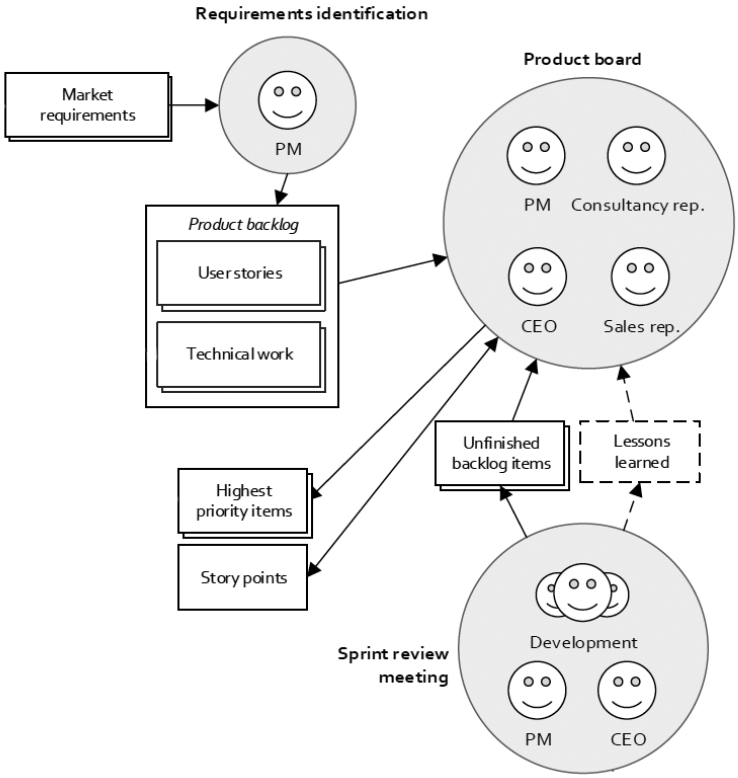

Figure 6.1 - Excerpt FLOW model

Members of the research group were acquainted with FLOW models and their vocabulary upfront, before proceeding with the interviews.

Reporting instructions required a FLOW model itself as a description of the ASD process within the organization. In addition, a description of solid and fluid information was mandatory. The senior researchers provided interim feedback to the other members of the research group, which would sometimes lead to additional data collection. Interim feedback was provided based on a draft report, also to assure quality of the final report. A draft report was also submitted to interviewees to ensure correctness, especially of the FLOW model

as an interpretation of the interview(s). An average (final) report comprised 20 pages.

6.2.3 Data Analysis

The FLOW models together with their description of solid and fluid information formed our base data. The characteristics of fluid information, being the opposite of at least one of (1) long term accessible, (2) repeatedly readable, and (3) comprehensive for third parties, justifies an equation of fluid information from the models to fuzzy artefacts. So, we started data analysis by extracting fluid information as fuzzy artefacts from the models, and this resulted in first instance in 82 items in 30 teams. Eight teams did not report fuzzy artefacts at all.

In a semantic analysis we used the accompanying descriptions of the FLOW model's documented or undocumented information to identify similarities and differences. In this analysis we applied constant comparison (Glaser & Strauss, 1967). This comparison was done manually, proceeding from one item to another and working backwards whenever appropriate. For instance, we first identified a fuzzy artefact 'Retrospective' which was described as "*… knowledge from looking back on the last sprint: what went right and what went wrong*". When we found 'Lessons learned' (see also Figure 6.1) described as "*… insights into the development process during the sprint, issues implementing product features or feedback on the current build that may be used to improve the software development process or software product in the future*", we unified the two under one heading, and named this fuzzy artefact 'Lessons learned' thereafter. In this process we mapped items when they could be equated to items also mentioned by another team. We equipped each fuzzy artefact with an appropriate heading (Lessons Learned) and description (These are insights into the development process that may be used to improve the software development process or software product in the future.) where

the description might be one of the previous ones or a new blended description based on a combination of the previous ones.

In this process we decided to rule out nine appearances of fluid information and to not consider them to be fuzzy artefacts. The main reason for this was a contradiction in the graphical FLOW model and the accompanying textual description. Examples of fluid information, from FLOW models, not taken into account were, for instance, product backlog and release note.

6.3 Findings

Thirty organizations reported 73 sources of fluid information, ranging from one to six per team; eight out of 38 organizations reported not to use any fluid information at all (Figure 6.2).

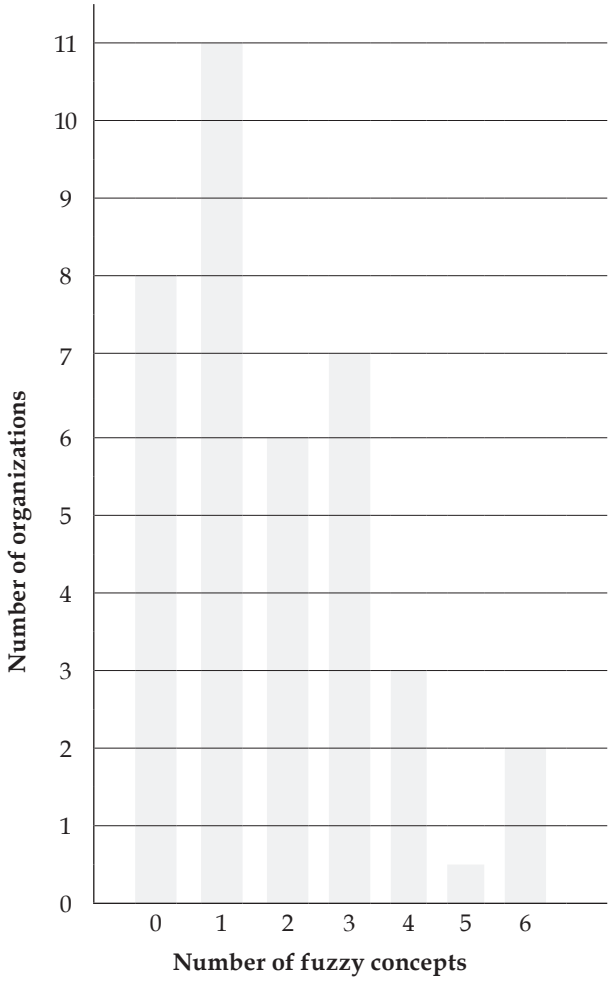

Figure 6.2 - Number of fuzzy artefacts per organization

Our semantic analysis identified fluid information in eleven fuzzy artefacts (Table 6.2).

Table 6.2 - Identified fuzzy artefacts

Identified fuzzy artefacts	Usage in number of teams
Wishes	22
User story planning	11
User story specification	8
Bugs	6
Go/No-Go decision	6
Quality constraint	4
Strategy	4
Technical knowledge	4
Customer feedback	3
Management wishes	3
Lessons learned	2
Total	*73*

Although wishes and customer feedback both originate from customers there is a distinction between the two that justifies two fuzzy artefacts. Customer feedback is provided directly as a result of demonstration of a part of the software, in agile/Scrum terms, a review. Wishes would refer to implicit requirements whose inclusion in the product is regarded as important by internal and/or external stakeholders. They are part of a continuous process in which new features or adaptation of the software are suggested to the SPO or its employees. A management wish is an informal request for a requirement, made directly by senior management, described in one case as "*This is an informal demand to change the content of the next release. It is made by the CEO, in case he does not fully agree with the release*".

User story specification is especially applicable when additional information is acquired, for instance, as "*the design team asks the software team to take into account technical functionalities of the user stories*". User story planning includes both assignment of priorities and effort, for instance in a planning poker procedure.

Go/No-Go decisions are decisions about, whether or not, (a part of) the ongoing project / sprint should be continued.

A quality constraint represents a criterion that restricts the software. An example was provided by organization A11's team, where "*undocumented quality requirements*" were identified with the addition "*… are mostly not formulated within documents but are directly communicated between employees*". This fuzzy artefact is certainly not formal, but is not completely tacit either, since the concept is recognized in the team beyond the level of a single occurrence of face-to-face communication.

Strategy is an intangible high-level vision for the product (and the organization) to guide the direction in which its development is headed; strategy is not written down, but embedded in the organization's culture.

Technical knowledge is held by the Development Team where it remembers technical issues or compromises risen during development. Examples of such knowledge include, among others, technical debt, good coding practices, or shared ideas on how to implement a backlog item. In all cases the agile teams mentioned their existence, but denied a formal record of them. As good coding practices were described, "*they concern practices used and discussed by the development team, in order to create maintainable software*", it is evident that they are informal, surely used and discussed, but not formal. They are fuzzy.

Lessons learned are defined as insights into the development process that may be used to improve the software development process or software product in the future. A sprint retrospective, although of course an informal, undocumented, one in this case, was mentioned as an example under this heading.

6.4 Discussion: Major Fuzzy Artefacts

In the identification of fuzzy artefacts wishes, including management wishes, user story planning and specification, Go/No-Go decisions, and bugs together account for approximately three quarters of the total number of fuzzy artefacts. We discuss these prominent artefacts.

6.4.1 Wishes and Management Wishes

The most frequent fuzzy artefact in our findings is wishes. This concept shows that requirements specification by a product owner, influenced by other stakeholders, is not a thoroughly structured process. This is not a surprise, since the software product's "*high release frequency … makes requirements organization highly complex, especially for large and complex software products that offer integration with other software*" (Fricker, 2012). Detailed definition of requirements in Software Product Management (SPM) is considered a three step process: (1) an SPM team translates concepts into a list of requirement definitions without going into a lot of detail, (2) a software development team elaborates these into requirements containing a detailed description of desired functionality, described in sufficient detail to work with, and (3) a development team ensures requirement clarity, so that each requirement is understood by all team members (Vlaanderen et al., 2011). Our wishes may be positioned in step one as the concepts, while at the same time emphasizing that they truly are informal concepts.

6.4.2 Elaborations of User Stories

While maturity models for SPM place great emphasis on requirements and their traceability (Bekkers, Weerd, Spruit, & Brinkkemper, 2010; Gomes, Pettersson, & Gorschek, n.d.) our current findings also show the structure of the requirements gathering process still may be improved. Over half of the teams uses fuzzy artefacts in this process. Furthermore, a small number of teams have wishes

entering their requirements process because of ad hoc intervention by senior management. Although such management wishes may be justified as far of their contents is concerned, they are not part of a mature, structured requirement process.

User stories may already be considered to be a shortcut for more formal documentation: "*Most organizations shun formal documentation of specifications. Instead, they use simple techniques such as user stories to define high-level requirements*" (Cao & Ramesh, 2008, *p. 63*). This is in line with an early description of a user story: "*Although Stories are written down on cards, these cards act as tokens within a release plan rather than descriptions of Stories; they represent customer requirements rather than document them*" (Davies, 2001, *p. 48*).

The fact that user stories need more detail was therefore to be expected. It is also known that agile teams draw up 'traditional' designs for this (Bass, 2016; Wagenaar et al., 2015). What is surprising in our current findings is the number of teams that deal informally with this, because it appears contradictory to the finding that several reference models that describe SPM practice exist (Fricker, 2012). Such models are meant just to contribute to structure, not in the least for the requirements process. In total 19 agile teams identified fuzzy artefacts as playing a role in either user story prioritization or user story planning.

6.4.3 Go/No-Go Decision

Intuitively a Go/No-Go decision is a formal mark in a decision process. Nevertheless, we encountered it as a fuzzy artefact in six agile teams. They concerned decisions about:

1. Acceptance by the CEO on the content present on a final scoping document used to populate a product backlog.

2. Implementation of a requirement by a product owner.

3. A design document by the customer.

4. Signoff to release the code after the testing done by the product owner and/or customer support by management (potentially CEO).

5. Acceptance by a consultant manager on whether or not the current product can be shipped to customers.

6. Acceptance by a customer on newly developed requirements and approval of deployment on a production environment.

Some of those decisions are internal decisions (1-2, 4-5) made either by a member of the agile team (product owner) or higher management. Some fuzziness here could perhaps be expected. However, decisions made by the customer, whether on a design document or concerning acceptance could be expected to be more formally backed up. A structured project management method like Prince2 considers Go/No-Go decisions to be a very formal milestone in a project (Bennett, 2017).

6.4.4 Bugs

"*Electronic repositories hold incomplete or incorrect data more often than not*" (Aranda & Venolia, 2009, *p. 306*). Therefore, it is inevitable that additional information on bugs is required occasionally. This is what we encountered in six teams. All teams indeed used an electronic repository; five of them Jira, one a structured storage of Requests for Change. Despite the repositories, the teams still considered bugs to be fuzzy artefacts, which might suggest that the tools do not provide adequate support and are not used in their full potentials.

6.4.5 Fuzziness versus Formality

Both agile artefacts and traditional artefacts have been identified in the software development of agile teams (Bass, 2016; Wagenaar et al., 2017). Some of them overlap with our fuzzy artefacts, for instance bugs or user story specification. This is in no way contradictory, since it only shows that some agile teams formally document artefacts, while others do use the concept, but do not explicitly register them.

Other fuzzy artefacts do not have a counterpart in formal artefacts, with a Go/No-Go decision as the most prominent example. Agile teams should perhaps consider formalizing especially these artefacts.

6.5 Conclusions

Our research question investigated the concept of fuzzy artefacts and the findings from our case study confirmed their existence. Agile teams do use concepts which are not formally documented but are of such a pervasive nature that a team is well aware of their existence. We found 73 fuzzy artefacts in 38 agile teams, which means that an average team used approximately two fuzzy artefacts, ranging from zero to six, in their ASD process. In this way we extended the typology of formal communication (artefacts) and informal communication (face-to-face) in ASD with an intermediate concept: fuzzy artefacts.

Most fuzzy artefacts are related to requirements. Notwithstanding the structured approach advocated by SPOs many wishes from external stakeholders (customers) and/or internal stakeholders enter the requirement engineering process secretly, but not unnoticed by the team. Teams also observe management intervening in the process in an informal, but structural manner.

Populating a product or sprint backlog with user stories equipped with priorities and efforts is another area where quite a number of fuzzy artefacts was found. In addition to the usual artefacts, such as product or sprint backlog, roughly half of the number of teams elaborated the user stories, thus enriching this process with informal communication.

Two other major fuzzy artefacts are Go/No-Go decisions and bugs.

Some fuzzy artefacts coincide with formal artefacts. However, there is a number of fuzzy artefacts that does not, and they constitute a new dot on a tacitness scale.

6.5.1 Validity

Validity of a case study depends on four criteria: construct validity, internal and external validity, and reliability (Yin, 2013).

6.5.1.1 Construct validity

Construct validity identifies operational measures for concepts under study. To enhance construct validity: (1) key informants should review draft case study reports, (2) multiple sources of evidence should be used, and (3) a chain of evidence should be established. We addressed all three:

- For each interview summaries were drafted, including a FLOW model, and interviewee(s) were able to comment on them.
- Within some teams, but certainly not all, interviews were held with more than one interviewee so in those cases viewpoints could be complemented. When only one interviewee was involved though, personal opinions or interpretations may have influenced our results.

- We followed a strict procedure in proceeding from an interview protocol, via a FLOW model and its descriptions, to a list of fuzzy artefacts. However, organizations were visited by different members of the research group, so interpretation may have influenced especially the construction of FLOW models, although our CSR was meant to minimize this risk.

6.5.1.2 Internal validity

Internal validity is considered mainly a concern for explanatory case studies. We do not claim our case study to be explanatory, it is rather exploratory. Still we considered internal validity in translating oral interviews via the FLOW models to fuzzy artefacts. Classifying fuzzy artefacts, as described in Section 6.2, was to a great extent exploratory, but in a structured way through our use of constant comparison.

6.5.1.3 External validity

External validity defines the domain to which a case study's findings can be generalized. The use of replication logic is listed as the main guarantee for this. Using various single case studies on the basis of a common procedure (CSP), as in our research, in general already contributes to external validity, and thus to generalizability of results. However, our organizations being SPOs may have influenced our findings, since an ongoing development process, which is characteristic for SPOs, tends to have a formalized structure. Generalizability to non-SPOs is then limited, but only in the sense that in non-SPOs an even greater number of fuzzy artefacts may be expected. Even so, although our research included teams with different characteristics (domain, country, and size) we cannot claim them to be representative for every agile team.

6.5.1.4 Reliability

Reliability should demonstrate that the study can be repeated. Using an interview guideline, instructions for FLOW modelling, and formatting results as well as the use of a case study repository contributed to reliability.

6.5.2 Future Research

Communication in agile teams remains a fruitful area of research. Although our research contributed to this area in showing the existence of fuzzy artefacts, it does not address their degree of fuzziness, compliant with Zadeh's original work (Zadeh, 1965). Exploring whether or not such a further classification could be achieved remains an issue for future research.

CHAPTER 7

Competencies outside agile team borders

According to the Scrum process framework a Scrum team should have all necessary competencies to accomplish its work. Fragmented and anecdotal evidence hints at Scrum teams still needing additional, external competencies. To contribute to theories on Scrum team composition and practitioner's concerns in staffing a Scrum team we investigated Scrum teams' cross-functionality: To whom do Scrum teams turn for additional competencies, which competencies are involved and how are Scrum teams aware of additional competencies they need? To this extent we analysed the communication in three Scrum teams during one of their Sprints. Our results show that additional competencies are called for, not only on an ad hoc basis, but also on a structural basis. To include those structural competencies the notion of an extended Scrum team is introduced.

This work was originally published as:

Wagenaar G., Overbeek S., Helms, R.: Competencies outside Agile Teams' Borders: The Extended Scrum Team. In: Position Papers of the Federated Conference on Computer Science and Information Systems (FedSYS), 36th IEEE Software Engineering Workshop (SEW-36), Gdansk, Poland, 11-14 September, 2016, 291-298

7.1 Introduction

The application of an agile software development method, for instance XP or Scrum, is nowadays common in delivering state-of-the-art software (Bustard et al., 2013; Rodríguez et al., 2012). Team members with high competence and expertise are a critical success factor in agile software development, especially with regard to timeliness and cost (Chow & Cao, 2008).

One of the guidelines accompanying Scrum as a framework for developing and sustaining complex (software) products states on team composition: "*Cross-functional teams have all competencies needed to accomplish the work without depending on others not part of the team*" (Schwaber & Sutherland, 2013, *p. 4*). Such a cross-functional team should, among others, have skills with regard to software analysis, design and coding (Schwaber & Beedle, 2002).

In general, although often implicit, membership of a team is considered to be full time during a Scrum project, certainly for members of the Development Team (Cervone, 2011; Hass, 2007). Emergent team members may support a team during software development (Damian, Marczak, & Kwan, 2007) or even stronger it is often not clear which members belong to a team and which do not (Mortensen & Hinds, 2002) or: "*team boundaries are often permeable*" (Ehrlich & Chang, 2006, *p. 157*). Part time membership of a Scrum team could therefore be an option to consider (Chau & Maurer, 2004; Schwaber & Beedle, 2002).

Fragmented and anecdotal evidence already hints at Scrum teams still needing additional, external competencies (Ferreira, Sharp, & Robinson, 2011; Martin, Biddle, & Noble, 2010; Sharp & Robinson, 2010), but this evidence was gathered as a by-product of more general research on communication in agile projects. Unlike this research we only focus on composition of Scrum teams, especially

their cross-functionality. We determine its boundaries and look for additional competencies not included in the team. To this extent only we analyse the communication within and outside team boundaries of Scrum teams' members to identify which additional competencies they require and on which basis. In this way our research on the one hand contributes to a better understanding of team composition in Scrum software development, and especially in the competencies included in the team, and those outside its boundaries, and on the other hand allows practitioners to mirror their way of working, and support them in the formation of a Scrum team.

The remainder of this paper is organised as follows. In section 7.2 we outline the theoretical background relating to our work, based on a literature review. In section 7.3 we present our research method. The results for three case studies are presented in section 7.4, followed by a discussion in section 7.5. Section 7.6 is the final section in which our conclusions are presented, with their limitations and future work.

7.2 Theoretical Background

In terms of socio-technical congruence (Cataldo, Herbsleb, & Carley, 2008) the Scrum process framework (Schwaber & Sutherland, 2013) describes the coordination requirements established by the dependencies among tasks. Upon inspection of the actual coordination activities a match between coordination requirements and activities may be established; such a match has proven to be beneficial to several aspects of software development, for instance reducing the resolution time of modification requests (Cataldo et al., 2008) or software build success (Kwan, Schröter, & Damian, 2011).

We apply socio-technical congruence specifically to the use of competencies in a Scrum team. The coordination requirements defined by the Scrum process

framework state cross-functionality. Should a perfect match with the actual activities exist, i.e. they can be carried out without external competencies, then a Scrum team is indeed cross-functional.

Competencies for agile software development have been divided into three major categories, forming a pyramid of agile competencies with engineering practices at the bottom via management practices to agile values at the top (Kropp & Meier, 2013) or, similarly, technical skills, people or soft skills, and attitudes (Bootla, Rojanapornpun, & Mongkolnam, 2015). Support for the two latter categories is a responsibility of a Scrum master, although an agile coach, operating outside a team's boundaries, is also often involved (Downey & Sutherland, 2013; Paasivaara & Lassenius, 2014). Since these are individual rather than team competencies, we use socio-technical congruence to focus on the former category: Engineering skills.

There are already indications that Scrum teams are not entirely cross-functional in this respect. No matter how tightly-knit agile teams are, they need to interact with roles outside the team (e.g. user experience designers, database administrators, system testers), because in practice it is infeasible for all individuals with relevant expertise to be part of the team (Sharp & Robinson, 2010). This is, for instance, confirmed in the communication between the disciplines of agile development and user experience design (Ferreira, Sharp, & Robinson, 2010).

Although internally distributing expertise in agile teams ultimately leads to successful cross-functional teams (Rejab, Noble, & Allan, 2014a), this appears to be an ideal situation. At least until then, but perhaps even on a more continuous basis, it is an unrealistic goal to aspire to include all competencies in a Scrum

development team itself, with its 3 to 9 members. Additional roles, contributing competencies to the Scrum team, have already been hinted at; they could be user experience designers, database administrators, system testers (Sharp & Robinson, 2010), acceptance testers, user interaction designer, or technical writers (Martin et al., 2010). Part time members could also be system or database administrators (Schwaber & Beedle, 2002).

Four ways to locate expertise in a Scrum team itself have been revealed:

1. Communicating frequently,
2. Working closely together,
3. Declaring self-identified expertise in order to let others know what a member can contribute to the team,
4. Using an expertise directory (Rejab, Noble, & Allan, 2014b).

It could be the case that these also apply to locating expertise outside a team's border, but this has not been established yet.

Coordinating expertise outside agile teams benefits from five factors:

1. Availability refers to the ability of external specialists to be present in agile teams when their expertise is needed,
2. Agile mind-set concerns dealing with external specialists who might be unfamiliar with agile methods and thus introducing problems for the team, for instance external specialists not being able to align work with the sprints,
3. Stability refers to keeping agile teams stable with a low rate of team members and external specialists turnover,

4. Knowledge retention involves capturing external specialists' knowledge and preserving the knowledge in agile teams,
5. Effective communication is defined as the activity of conveying sufficient information between agile teams and external specialists (Rejab, Noble, & Marshall, 2015).

These factors were established from responses from individuals, which were not necessarily joined in teams. Although important factors in coordinating expertise outside an agile team were identified, it does not answer the question how this expertise is identified in the first place or what expertise is involved.

Software development in general needs expertise finding to facilitate unplanned collaborative work among software developers, but: "*Who are these additional people, and why are they contributing, or why have they been contacted …?*" (Damian et al., 2007, *p. 89*). And these questions are equally applicable to the use of Scrum in software development.

To identify partners outside Scrum team boundaries, it is important to realize that both Product Owner and Scrum Master act as linking pins with the team's environment; the Product Owner in managing the Product Backlog in cooperation with stakeholders, the Scrum Master in serving the (outside) organization, for instance in helping employees and stakeholders understand and enact Scrum and empirical product development. Communication between the Product Owner and the Scrum Master on the one hand and partners like management and stakeholders on the other hand are thus already included in the Scrum process framework. Taking this into account and under the assumption that the Scrum team members attend all regular Scrum events we present an initial sketch of a Scrum team and its external partners (Figure 7.1); this sketch

can thus be considered as the common composition of a Scrum team (and its partners).

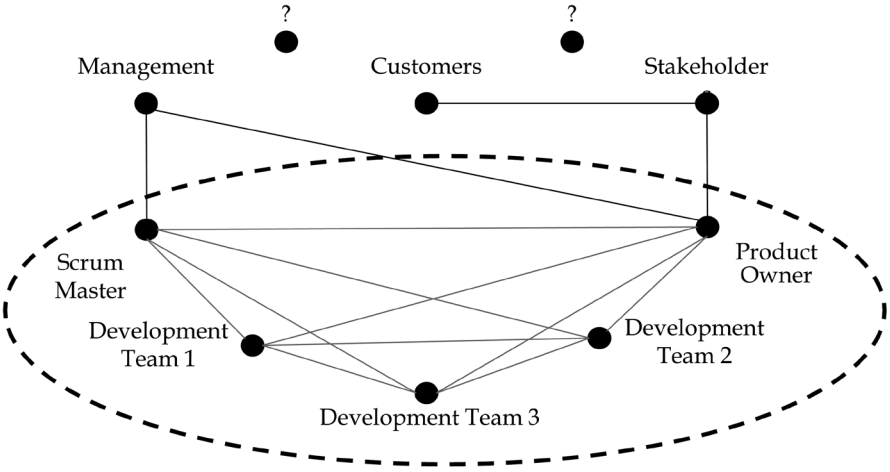

Figure 7.1 - Communication inside and outside Scrum teams

Question marks represent yet unidentified partners with additional competencies. The members Development Team 1 – 3 represent team members with competencies with regard to software analysis, design and coding (Schwaber & Beedle, 2002), where each node represents several team members, depending on the team size. We will use this model both to guide our data analysis as well as to compare its contents with our results.

7.3 Case Study Design

To investigate the cross-functionality of Scrum teams our research analyses the communication of Scrum teams outside their team boundaries to identify which additional competencies they require and on which basis. To allow for rich evidence we selected an exploratory comparative case study approach as our research method with as unit of analysis one sprint of Scrum software development. This approach is an accustomed way to investigate phenomena

in a context where events cannot be controlled and where the focus is on contemporary events (Yin, 2013). For this case study we drew up the following research questions:

RQ1 To whom do Scrum teams turn to for additional competencies and which competencies are involved?

RQ2 How are Scrum teams aware of what additional competencies are needed?

We used a protocol to guide the case study (Maimbo, 2005). This protocol contained:

- Its purpose, guidelines for data and document storage, and publication.
- A brief overview of the case research method.
- Detailed procedures for conducting each case, to ensure uniformity in the data collection process and consequently facilitate both within and cross case analyses.
- Research instruments.
- Guidelines for data analysis.

Given our goal and research questions, organizations were required to use Scrum as software development method with a team of at least 5 members; we chose this lower limit to allow us to find indeed cross-functional teams. We approached three organizations to participate in our research and all three were willing to do so; they all had a team size between 5 and 10 members. We name them Controller, Sunflower and Local for reasons of confidentiality.

7.3.1 Data Collection

The primary data collection method was semi-structured interviewing of team members. We also inspected available documents and/or the contents of information systems which, in some cases, were used to support the Scrum process. Our questionnaire addressed communication both between team members, exemplified in the use of Scrum practices, as well as communication between team members and non-team members. Some examples of questions of the latter category (from the protocol) are:

- With whom did you communicate (mainly) during the sprint? Please mention all people and include people who might have been outside the scope of the sprint (in first instance).
- What was the communication about?

Interviews lasted, on average, 60 minutes. In total approximately 12 hours with 13 interviewees were available. To achieve a representative sample a team's Scrum Master and Product Owner were always included and, depending on the size of the team, 2 to 4 developers, including designers and testers when these roles were explicitly assigned in a team. Six more additional interviews were held either before the other interviews started to provide general context, or afterwards, to clarify remaining issues.

Thirteen interviews were transcribed and coded, with a combination of open and axial coding (Paasivaara & Lassenius, 2003). We also used our preliminary sketch (Figure 7.1) to guide the coding; it guided the coding in the sense that external partners were either classified as manager, customer or stakeholder or an additional partner not yet included (previously a question mark). Summaries of interviews were consulted with the interviewees. For each organization the

results of the interviews and additional material were bundled in a case study report.

7.3.2 Validity

Validity of our research method depends on four widely used criteria: construct validity, internal and external validity, and reliability (Yin, 2013).

Construct validity identifies operational measures for the concepts under study. To enhance construct validity (1) key informants should review draft case study reports, (2) multiple sources of evidence should be used, and (3) a chain of evidence should be established (Yin, 2013). We applied all three: (1) Each interviewee was provided with a summary report of the interview and key interviewees commented on a draft case study report, (2) various team members were involved to complement viewpoints, and (3) interviews (and other materials) were linked to conclusions by using the tool NVivo; NVivo is a software package to aid qualitative data analysis (www.qsrinternational.com).

Internal validity is mainly a concern for explanatory case studies (Runeson & Höst, 2008; Yin, 2013). Our case study is exploratory, but we did apply pattern matching, one of the analytical techniques recommended to enhance internal validity, by consistently coding our base material.

External validity defines the domain to which a case study's findings can be generalized. The use of replication logic is listed as the main guarantee for this (Yin, 2013). Using a multiple-case study on the basis of a common questionnaire contributes to external validity, and thus to generalizability of results.

Reliability should demonstrate that the study can be repeated. The use of a case study protocol and the development of a case study database (Yin, 2013) were both applied in our study to increase reliability.

7.4 Results

In this section we describe the results from our case studies. We first give an impression of the organizations and the composition of their Scrum teams (Table 7.1). For Controller the role of Scrum Master coincides with one of the developers/testers; for Sunflower the role of Scrum Master coincides with one of the senior developers.

Table 7.1 - Description of organizations

Organization	Domain	Team composition			Scrum experience
		Product Owner	Development Team	Scrum Master	
Controller	Object management	1	2 developers 2 developers/testers	1	2 years
Sunflower	Floral industry	3	3 senior developers 1 junior developer	1	2½ years
Local	Government taxing	1	2 designers 4 developers 2 testers	1	1½ years

In the next three paragraphs we describe, for each organization in turn, the communication within, but mainly outside their Scrum teams' borders.

7.4.1 Controller

The Scrum team of Controller operates in a larger organizational context: the Development department with 35 employees (Figure 7.2).

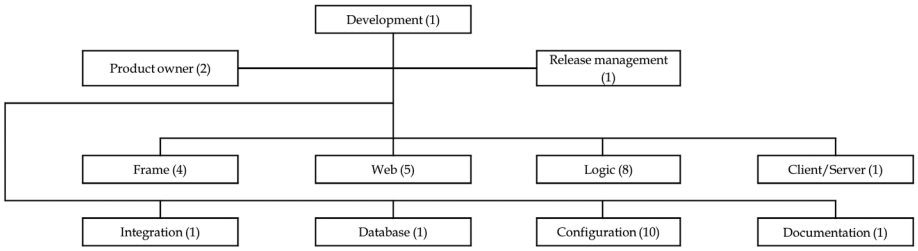

Figure 7.2 - Controller organization chart for Development department

The numbers in Figure 7.2 refer to the number of employees in a group. The Frame group is responsible for the building blocks (basic components) of the software; Web adds a graphical skin. Together these groups produce a kind of half-fabricate. The groups Logic and Configuration build end products on the basis of these (supporting) components; the Scrum team is staffed from these two groups and complemented with one of the two Product Owners. In this Logic takes care of the process flow in the software, whereas Configuration is concerned with visual aspects, screens, buttons, reports, et cetera. Configuration is also responsible for testing the software. All other groups are small; they support the four groups mentioned before.

The Scrum team communicates in the framework of Scrum events. Sprint Planning Meeting, Daily Scrum, Sprint Review Meeting and Sprint Retrospective are all regularly scheduled 'meetings'. But team members also communicate with non-team members. The figure below (Figure 7.3) demonstrates communication within and beyond the team's borders, where the customer is the only partner outside the organization Controller directly communicating with a team member.

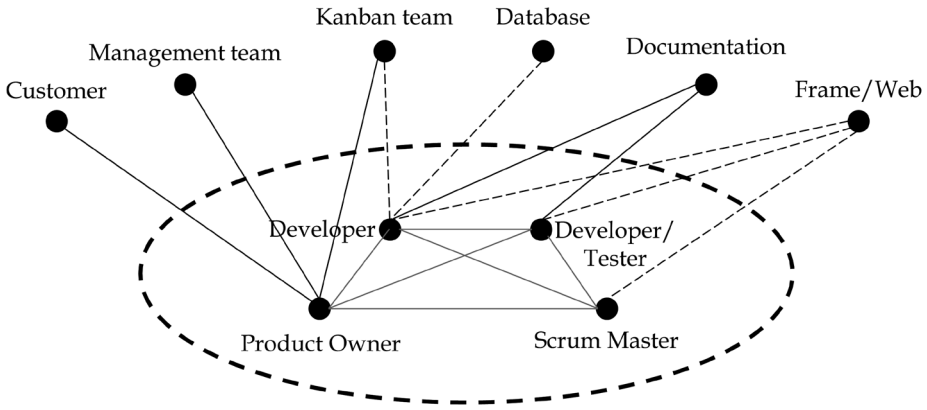

Figure 7.3 - Internal & external communication for Controller's Scrum team

The members of the Scrum team are shown in the lower half of Figure 7.3; there is agreement on the team composition. In Figure 7.3 only roles of team members are shown; the team in fact has 2 developers and 2 developers/testers (refer to Table 7.1). Apart from the Product Owner, all members are full time members.

External partners are shown in the upper half of Figure 7.3. Partners are only involved in the team's activities part time, where in vast majority part time means less than a small number of hours per Sprint. Solid lines represent structural communication between the team members, where we defined structural as communication not only in the Sprint under consideration, but also in a majority of Sprints; dotted lines indicate ad hoc communication taking place in this Sprint only, but not necessarily in other Sprints. Furthermore we include only partners who are directly communicating with at least one team member as the team is the focus of our research.

Considering the external partners we focused on the unidentified partners (Figure 7.1). Already identified partners for Controller then are Customer and Management team.

The Scrum team is only involved in the development of new parts of the software; maintenance is done by another team with Kanban. The Product Owner is responsible for coordination of the Kanban team; he - every employee is indicated as 'he', whether male or female - coordinates (the prioritization of) maintenance. It is his task to recognize overlapping activities of the Scrum and the Kanban team, with regard to components of the software, and to bring members of the two teams together whenever necessary. Although overlap does not occur every Sprint, communication is frequent whenever the two teams do work on the same piece of code to coordinate activities with regard to new code or adaptations to existing code.

The same applies for communication with the database specialist. He is consulted whenever sprint backlog items have impact on the database structure.

Documentation consists of a technical writer. He is responsible for the construction of a user guide and/or release notes in every Sprint and in cooperation with Development Team members. Communication is mainly on his initiative, where he has basic information available through a registration system.

Frame/Web, as producers of half-fabricates, are often contacted on details of the code they manufactured, although not in every Sprint.

The remaining groups in the department, Client/Server and Integration, were not mentioned to participate in the team's communication.

7.4.2 Sunflower

Sunflower belongs to the Small & Medium Enterprises (SME), more specifically a small company with a number of employees around 15; its organization chart is shown in Figure 7.4, the numbers referring to the number of employees.

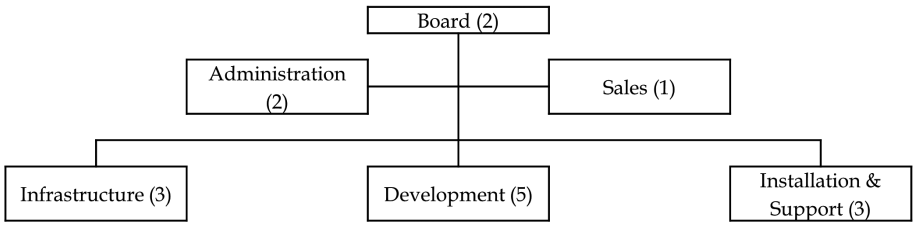

Figure 7.4 - Sunflower organization chart

Consultants, from Installation and Support, collectively function as Product Owner. The Development team consists of members of the group Development; the role of Scrum Master rests with a senior developer.

Members of the Scrum team communicate in the framework of Scrum. Sprint Planning Meeting and Daily Scrum are both regularly scheduled 'meetings', but they are skipped occasionally. Neither a Sprint Review Meeting nor a Sprint Retrospective is used by the team. Communication within and beyond the team's borders involves a restricted number of partners (Figure 7.5), where customer is the only partner outside Sunflower.

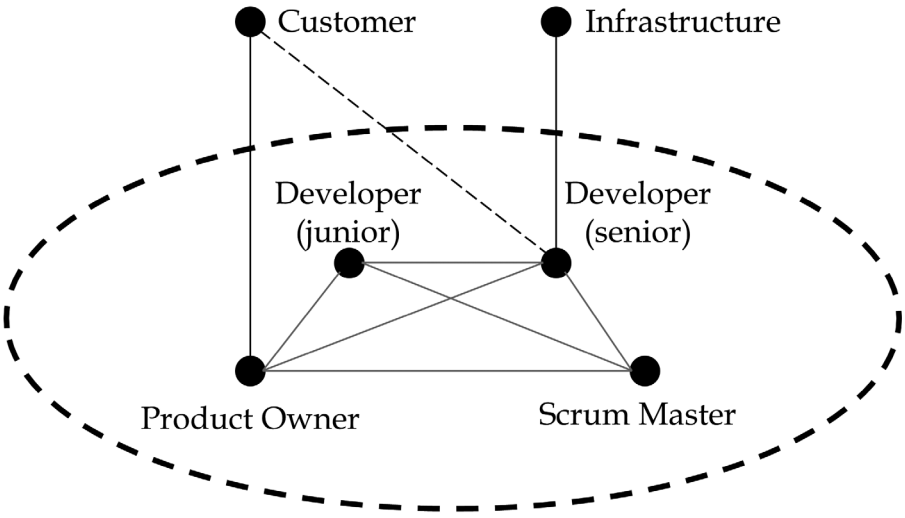

Figure 7.5 - Internal & external communication for Sunflower's Scrum team

A weekly meeting is scheduled with a representative of Infrastructure to prevent the Development Team from interfering with (scheduled) maintenance. However, whenever necessary, issues may also be taken up with Infrastructure immediately, not awaiting this meeting.

7.4.3 Local

The Scrum team of Logic operates in a larger organizational context: The Software Development department, consisting of:

- A Product group that is in charge of the implementation of software with new customers. Its developers convert customers from their previous software to Local's software; its consultants support customers with the set-up of the software and participate in courses for customers.
- Whenever a new customer approves of his software, it is taken into production at the customer and the responsibility within Local is transferred to the Support group. Support is the first point of contact for customers and functions

primarily as a helpdesk. Support has some consultants for customer support on site or, again, courses, but does not employ developers. Changes as a result of customer reports are transferred to Development.

- Development deals with all modifications to the software, whether new features or as a result of bug reports. The Scrum team is found within this group.

Analogously to the Controller team Local's Scrum team uses all of the regularly scheduled Scrum 'meetings'. The figure below (Figure 7.6) indicates communication within and beyond the team's borders.

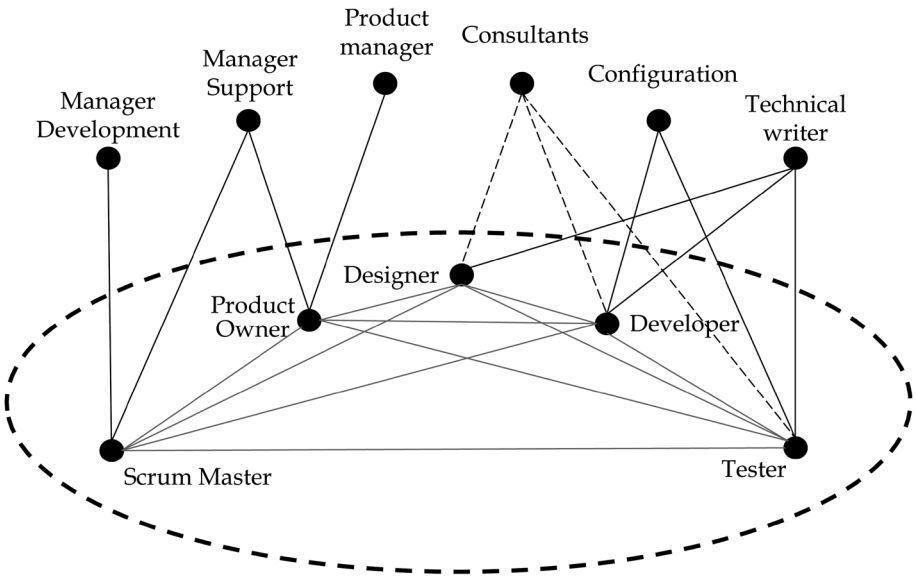

Figure 7.6 - Internal & external communication for Local's Scrum team

The managers (Development and Support) and the Product manager were already identified as stakeholders for customers. This implies that there is no direct communication between team members and partners outside of Local.

Configuration is in charge of the technical infrastructure. This partner, for instance, transfers software from a development stage to a testing stage to production (or vice versa). Involvement takes place in every Sprint.

The role of the technical writer is equal to the role of the documentarist in the Controller team.

7.4.4 Integrating Results

We have shown results for the three individual teams with regard to their communication. In integrating these results, and in line with our research questions, we now establish a common vocabulary by mapping the partners from the individual cases to a limited set, enumerate them and indicate whether the communication is structural or ad hoc (Table 7.2).

Table 7.2 - Summary of results

External partner	Controller	Sunflower	Local
Management	*Management team*	-	*Manager Support*
			Manager Development
Consultant	*Consultants*	-	*Product manager*
			Consultants
Customer	*Customer*	*Customer*	-
Developer	*Frame/Web*	*Developer*	-
	Kanban team		
Documentarist	*Documentation*	-	*Technical writer*
Database specialist	*Database*	-	-
Infrastructure specialist	-	*Infrastructure*	*Configuration*
		Structural communication	
		Ad hoc communication	

7.5 Discussion

It is important to note that, for none of the teams, there were differences of opinion on their composition. All Scrum teams were crystal clear about their membership. As an example, when interviewing members of Local's Scrum team all members agreed on having ten members on the team and every team member mentioned the same persons in the same roles (Table 7.1). Whether being involved full time or part time (some Scrum Masters and Product Owners), team membership, and thus also non membership, was incontrovertibly.

In the staffing of the teams it is straightforward that competencies of team members are in one or more of the 'traditional' phases of a software development life cycle: Analysis/design, programming, testing (Table 7.1, Figure 7.3, Figure 7.5 & Figure 7.6); these are indeed engineering skills (Bootla et al., 2015; Kropp & Meier, 2013).

When looking at the partners outside teams' boundaries we distinguish three categories:

1. In a first category we unite those partners who, from the viewpoint of socio-technical congruence, match coordination requirements with actual coordination activities; in fact they are the partners to be expected, as already identified in Figure 7.1.
2. In a second category we mention partners who do not match coordination requirements with actual coordination activities from the viewpoint of socio-technical congruence, but join the team through ad hoc communication.

3. The third category concerns partners who also do not match coordination requirements with actual coordination activities, but do so through structural communication.

This first category encompasses consultants, customers and management. None of these partners provides engineering skills. Instead they provide domain knowledge and their communication takes place via the Product Owner (with one small exception where the Scrum Master was also involved). This comes as no surprise, since the Product Owner is the linking pin with stakeholders as he is responsible for managing the Product Backlog, an ordered list of everything that might be needed in the product (Schwaber & Sutherland, 2013). Management also communicates on (project) progress with the Product Owner and/or the Scrum Master, because in Scrum there is no role of project manager. In fact none is needed; the traditional responsibilities of a project manager have been divided up and reassigned among the three Scrum roles (Deemer et al., 2010), especially the Product Owner and the Scrum Master (Yilmaz, O'Connor, & Clarke, 2012). Communication with management is thus to be expected; Scrum teams do not operate in a vacuum. In a previous study we already found that Scrum teams use project plans and progress information for control purposes (Wagenaar et al., 2015). And our current results show the structural character of this communication.

The second category consists of developers and database specialists, contributing engineering skills or competencies with regard to code/coding and database (structure) respectively. Often this communication is facilitated by the Scrum Master or arranged by the Product Owner; initiatives originate from these two roles, but often delegated to a developer working on a particular backlog item thereafter. The communication then is from one developer to another. This

category is in fact no different from non-Scrum software development: In e-mail discussions in software-engineering communication emergent people were included in a discussion as a result of an explicit request (Kwan & Damian, 2011).

From the viewpoint of socio-technical congruence our data show a clear mismatch between coordination requirements and actual coordination in the third category. Both documentarist and infrastructure specialist are structurally involved in the Scrum teams' activities. The documentarist is competent with regard to the production of user-related materials, such as a user guide. Communication is supported by data in a registration tool, such as (elaborated) user stories, design documents, test reports. The infrastructure specialist supports the Scrum team with regard to transition of code to a test or production facility; of course this is a part time activity; he also supports other teams. The incorporation of such a specialist in a Scrum team's activities coincides with the DevOps concept; this is an agile operations concept that uses agile techniques to link up departments - Development (Dev) and Operations (Ops) together, which traditionally operated in silos (Karunakaran, 2013).

No matter to which category partners belong, Scrum teams are well aware of their external partners and their competencies. Differences between the categories refer to whether or not the use of competencies was planned and whether it was structural or ad hoc. Revisiting the five factors for coordinating expertise outside agile teams - availability, agile mind-set, stability, knowledge retention, and effective communication – these factors are in fact already incorporated in the Scrum teams' communication with their external partners.

In fact we introduce the notion of an extended Scrum team, which includes partners structurally involved in a team's activities, because communication occurs Sprint after Sprint (Figure 7.7).

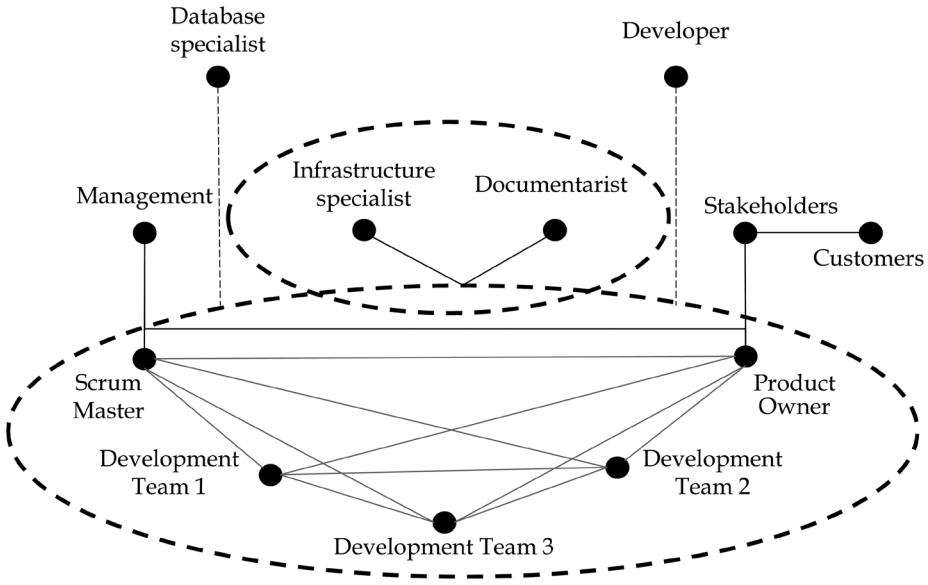

Figure 7.7 - The extended Scrum team

The extended Scrum team now incorporates an infrastructure specialist and a documentarist, adding technical and writing competencies. They work part time in the Scrum team, but they do not attend Scrum events, or at least not as a habit. The database specialist and the developer are (still) external partners. In other contexts it is conceivable that the database specialist is also a member of the extended Scrum team, because of his additional competencies.

7.6 Conclusions

Scrum team members agree on the boundaries of their team and are indeed cross-functional as far as 'traditional' phases of software development are

concerned. Analysis/design, programming and testing competencies are represented in the teams.

To whom then do Scrum teams turn for additional competencies, which competencies are involved and how are Scrum teams aware of additional competencies they need?

- When additional competencies are required with regard to engineering skills, already represented in the team, the team turns to partners within the own organization. The teams are already aware of their partners and consult them on an ad hoc basis. They include other developers and database specialists.
- Additional competencies are also found in partners who structurally contribute their competencies to the Scrum teams in every Sprint. These partners include management, documentarists and infrastructure specialists. Here management was already expected to do so with regard to domain expertise and progress information; this is indeed a socio-technical match between coordination requirements and actual coordination activities. Others were not; especially documentarists and infrastructure specialists structurally contributed, although not full time, to the Scrum teams; they could be considered to be members of an extended Scrum team.

7.6.1 Limitations

In our three case studies documentarists and infrastructure specialists are found to be members of an extended Scrum team; a database specialist was not. This particular result cannot be generalized to Scrum teams in general; depending on, for instance, domain area, software type, or specific features of a team, partners could be distributed in another way, with the infrastructure specialist outside a team's border and, for instance, a database specialist inside.

This argument also applies to partners who were not (even) identified in our case studies. What is generalizable, though, is the notion of the extended Scrum team, including partners not considered to be member of the team, but still structurally contributing to a team's activities. This allows practitioners to staff a Scrum team without pursuing overall cross-functionality in the team itself.

7.6.2 Future Work

Of course we would like to see our results confirmed with other organizations. We believe the extended Scrum team to be an important extension of the notion of a cross-functional Scrum team and we would like to observe more and/or other external partners to elaborate this notion. We are also eager to pursue situational factors, whether organizational or personal, that determine the inclusion of external competencies.

CHAPTER 8

Conclusion

The aim of this dissertation has been to investigate the role of artefacts in the communication in ASD teams. The research was formalized in the main research question:

What is the role of artefacts in the communication in teams involved in agile software development?

To answer this main question, the research was split into three parts:

- The first part focussed on the existence of artefacts as well as supporting communication and collaborative tools for backlog management and visualization.
- The second part addressed the role of artefacts in the communication in agile teams.
- In the third part communication mechanisms in agile teams partially based on the use of artefacts were investigated.

Each individual part was operationalized through multiple research questions. Combined, the answers to the research questions provide an answer to the main research question. The sub research questions are answered in Sections 8.1 – 8.3, structured according to the three parts of this dissertation. Section 8.4 presents an answer for the main research question. Section 8.5 contains the scientific and practical contributions of the research. The research's limitations and directions for future research are discussed in Section 8.6. Finally, Section 8.7 reflects on the research from a personal point of view.

8.1 Artefacts in ASD

The existence of artefacts in ASD, and the use of supporting tools led to three research questions:

A.1 Which artefacts do agile teams produce?

A.2 How can those artefacts be characterized to designate their use in agile team practices?

A.3 Which criteria do Scrum teams adopt when choosing computer-based support for their Scrum process?

8.1.1 Artefacts in agile teams

The hypothesis underlying questions A.1 and A.2 is that documentation has proven its value in traditional software development and that agile teams will at least carry over some of this value to their current practices. To answer the research questions case studies in three organizations were performed and their results were compared with existing artefact models.

This led to an agile artefact model (Figure 8.1), as an answer to research question A.1.

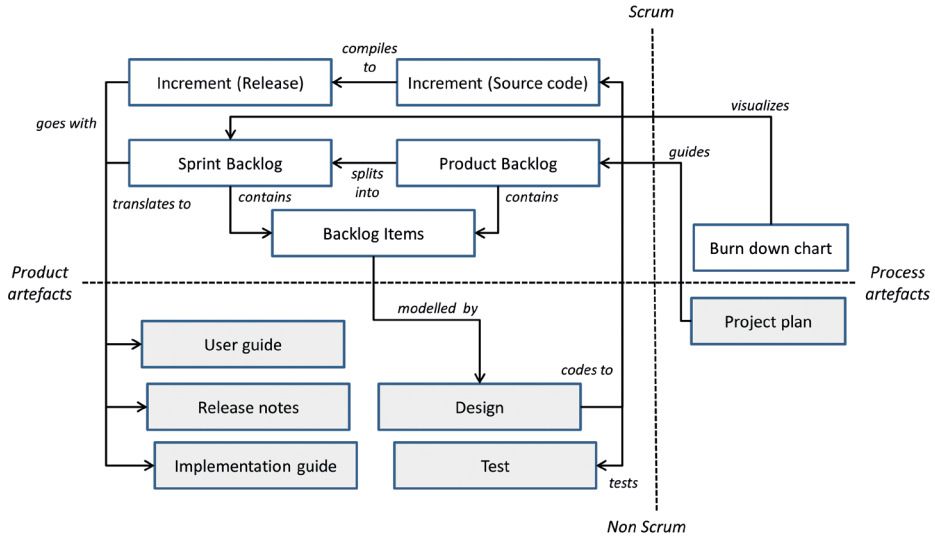

Figure 8.1 - Scrum artefact model (previously shown as Figure 2.3)

The model not only contains agile artefacts, but, in addition, various non-agile artefacts, most notably design documents, test plans, and user or release related materials. This model partially confirmed, especially for non-agile artefacts, other existing models (Dijkstra, 2013; Gröber, 2013). However, no existing individual agile artefact model included all artefacts depicted in Figure 8.1. Especially the design artefact enriched the models, because it identified a new artefact not yet included in any previous research. A later study confirmed its existence (Bass, 2016).

The Scrum artefact model also makes a clear distinction between product and process artefacts; one that was lacking in the existing ones (Dijkstra, 2013; Gröber, 2013). Finally we abstracted our model by separating function from form in denominating artefacts independent of their origin (Backlog Item instead of feature, bug, task) and separating supporting tools from artefacts themselves.

This abstraction allowed having a clear focus on artefacts themselves rather than their precise representation or support.

This part of the research developed a model with an explicit distinction between on the one hand product and process artefacts, and on the other hand agile and non-agile artefacts. This two-dimensional approach lacked in previous artefacts models (Dijkstra, 2013; Gröber, 2013). It shows that, despite agile's popularity, elements from previous software development methods linger on. The artefacts in the Scrum artefact model served on the one hand as a trigger for further exploring criteria used in the selection of supporting tools, as they are separated from artefacts themselves (Chapter 3), where on the other hand they served as a baseline for the research on the role of artefacts in agile team communication (as described in Chapters 4 and 5).

8.1.2 Criteria of Scrum teams in choosing computer-based support

Scrum teams extensively use tools to support their agile processes, but little attention has been given to the criteria a Scrum team applies in their selection. A greenfield approach was used to explore these criteria. Such an approach provides a relatively unbiased way to explore features a Scrum team thinks of as being important in its support. It at least assures that prior organizational or commercial influences are excluded. Twelve Scrum teams were asked to list criteria and assign weights in their decision processes. After having chosen and used a tool for three Sprints, teams evaluated the selected tools.

By using the Technology Acceptance Model (TAM) to structure findings, two major categories of criteria were identified: Perceived usefulness, alias criteria directly related to Scrum, and perceived ease of use. The TAM provides a sound

basis to classify criteria, as they fitted TAM's two categories. Most teams listed more or less the same criteria. Within the aforementioned categories specific criteria were distinguished, for instance burn down chart support or multi-platform aspects, the latter referring to the use of a tool on different platforms, such as web or mobile. The criteria Scrum teams adopt when choosing computer-based support for their Scrum process are summarized in Table 8.1; a further split into sub criteria is omitted here.

Table 8.1 - Overview of categories and criteria (summarized from Table 3.1)

Category Criterion	**#**
Perceived usefulness	1
• Supports backlog(s)	40
• Contains visualization task board	12
• Supports burn down chart	10
• Has Scrum compatibility	5
• Has additional features	1

Category Criterion	**#**
Perceived ease of use	-
• Ease of use	18
• Multiple platform support	14
• Acceptable price (or trial available)	10
• Security mechanism support	5
• Integration with other tools	3
• Communication support (e-mail, chat)	2

is the number of occurrences of a criterion.

This part of the research indicates that Scrum teams prefer perceived usefulness over perceived ease of use. In other words: Specific support of Scrum, especially its artefacts, are of greater value to a team than general tool considerations. Because of our approach, a greenfield one, the results would especially be useful for novel teams involved in an agile transformation.

8.2 Motivations for using artefacts

Having established the blend of traditional and agile artefacts whether supported by tools or not (in Part I), it became evident that agile teams do consider both agile and non-agile artefacts to be important. However, this did not yet shine light on reasons for using them. Two research questions (B.1 – B.2) were formulated to determine just those reasons.

8.2.3 Influence of maturity on usage of artefacts

In a first holistic view it was investigated whether or not organizational factors, i.e. maturity of a team's software management, has a relation to a team's use of artefacts. In agile product software development, SPOs as manufacturers of product software could be expected to use non-agile artefacts because of their needs with regard to, for instance, architecture and auxiliary materials. The underlying assumption is that artefacts usage is a quality consideration and relates to the quality of software product management (SPM) in an SPO. The resulting research question was formulated as:

B.1 To what extent does software product management maturity relate to the usage of artefacts in ASD?

In a multiple case study fourteen SPOs were visited where software product management maturity was rated and the usage of artefacts by an agile team listed. After applying statistical analysis, it was found that SPM maturity is

negatively correlated with a non-agile/all artefacts ratio. In other words, the more mature software product management is, the fewer non-agile artefacts are used in ASD.

This part of the research suggests that an organizational factor, SPM maturity, influences an agile team in, especially, its usage of non-agile artefacts. A possible explanation could be that a 'mature' SPO has organized its software product management already in such a way that additional documentation during ASD is hardly required, but further research is required to substantiate this claim. This would provide an answer to the question whether, especially non-agile, artefacts emerge from an agile team or are used for reasons which originate from outside the team. More general, how does an agile team reach a balance between agile and non-agile artefacts? Such rationales for the use of artefacts in agile teams are the subject of the next research question.

8.2.4 Rationales of agile teams for artefacts usage

Usage of artefacts may be the result of them being inherently included in an ASD method. Still, an agile team decides for itself on the usage of additional artefacts. To determine a team's rationales, agile teams were explicitly asked for them. The research question was phrased as follows:

B.2 What are rationales for agile teams to use artefacts?

The research method was a multiple case study. Nineteen agile teams were approached and interviews were held with prominent team members to discuss the use of artefacts and the motivation for using them. Fifty-five agile and non-agile artefacts were identified. With previous classifications in mind (Grant, 1996; Strode & Huff, 2014) five rationales accounted for artefacts usage:

1. Artefacts may simply come with ASD as 'prescribed', so adoption of an ASD method alone already leads to a number of, in this case, agile artefacts.

2. Agile teams impose governance on their own activities. Architecture and coding standards are for instance constituted to ensure a team's conformity to team wide agreements.

3. Artefacts are useful and/or necessary for quality reasons, This self-applied quality considerations lead, in general, to test-related artefacts.

4. Artefacts are useful and/or necessary for internal communication, which is then not face-to-face at all. In this category typical artefacts concern functional and technical design issues.

5. Artefacts are required by external parties need it. Examples of this are an user guide or a release note.

This part of the research shows that self-organizing agile teams do think about which artefacts to use in their communication process (and which not). Although it is doubtful whether rationales 1 and 5 should be attributed to the agile team's sphere of influence, because decisions regarding ASD introduction and demands from external parties may be assumed to be usually dealt with at other, higher, organizational levels. However, it remains that, for rationales 2 – 4, agile teams are self-organizing and explicitly choose to use non-agile documentation artefacts. Rationales 2 and 4 confirm items on the contemporary research agenda for large-scale agile software development: Architecture and inter-team coordination (Dingsøyr & Moe, 2014).

8.3 Beyond Using Artefacts

Artefacts, agile or traditional, are being used in ASD. But they together might not be the only communication means. Three research questions, C.1 – C.3 were formulated to discover other means of communication in agile teams.

8.3.5 Formality of communication in agile teams

ASD is often thought of as to predominantly use face-to-face communication. Research over the last years already led to adjustment of this conception. ASD also uses formal communication where it blends both traditional and agile artefacts. Research question C.1 investigates these intermediate appearances, named 'fuzzy' artefacts. They are artefacts not formally documented, but still explicitly recognized by an agile team. The research question was:

C.1 Which fuzzy artefacts do agile teams use in between formal communication with artefacts and informal face-to-face communication?

In an exploratory multiple single-case study thirty-eight agile teams in as many SPOs participated in answering the research question.

The findings confirm the existence of fuzzy artefacts, concepts which are not formally documented but are of such a pervasive nature that an agile team is well aware of their existence. The teams used in total seventy-three individual fuzzy artefacts (Table 8.2, summarized from Table 6.2).

Table 8.2 - Overview of fuzzy artefacts

Fuzzy artefacts	Usage in teams
Wishes	22
User story planning	11
User story specification	8
Bugs	6
Go/No-Go decision	6
Other	20
Total	*73*

Most fuzzy artefacts relate to requirements. Notwithstanding the structured approach advocated by SPOs, many wishes from external stakeholders (customers) and/or internal stakeholders enter the requirement engineering process informally, but do not go unnoticed from the point of view of the team, as they nominate them explicitly as fuzzy artefacts. Teams apparently observe management intervening in the requirements process in an informal, but structural, manner, as can be derived from the number of interventions in Table 8.2.

Populating a product or sprint backlog with user stories equipped with priorities and efforts is another area where they were found. In addition to the usual agile artefacts, such as product or sprint backlog, roughly half of the number of teams elaborated the user stories, thus enriching this process with informal communication. Two other prominent fuzzy artefacts are Go/No-Go decisions and bugs.

This part of the research shows that the former typology of formal communication (artefacts) and informal communication (face-to-face) does not suffice. Agile

teams highly and frequently use fuzzy artefacts as an intermediate concept in the scale formal - informal.

8.3.6 Competencies outside Agile Teams' Borders

According to agile thoughts an agile team should have all necessary competencies to accomplish its work. Fragmented and anecdotal evidence hints at agile teams still needing additional competencies. So, for internal communication on some topics, although supported by agile or non-agile, fuzzy or solid, artefacts, fails. The research questions to investigate agile teams' cross-functionality were formulated as follows:

C.2 To whom do Scrum teams turn to for additional competencies and which competencies are involved?

C.3 How are Scrum teams aware of what additional competencies are needed?

The results show that additional competencies are called for, not only on an ad hoc basis, but also on a structural basis. As a first result, agile team members do agree on the boundaries of their team and teams are indeed cross-functional as far as 'traditional' phases of software development are concerned: Analysis/ design, programming and testing competencies.

The answers to research questions C.2 and C.3, *to whom do Scrum teams turn for additional competencies?* and *which competencies are involved and how are they aware of additional competencies they need?*, is twofold.

When additional competencies are required with regard to engineering skills, already represented in the team, the team turns to partners within the own organization. There is no specific reason for or a restriction on the fact that

partners must be found within the organization; the teams in the research limit themselves. The teams are already aware of their partners and consult them on an ad hoc basis. They include, for the teams involved, other developers and database specialists.

Additional competencies are also found in partners who structurally contribute their competencies to the Scrum teams in every Sprint, such as management, documentarists and infrastructure specialists. Management was already expected to contribute with regard to domain expertise and progress information. Others were not; especially documentarists and infrastructure specialists structurally contributed, although not full time, to the Scrum teams; they could be considered members of an extended Scrum team (Figure 7.7).

This part of the research shows that complete cross-functionality of an agile team is impossible, because ad hoc consultations are inevitable. On the contrary, some competencies are that structurally needed that their suppliers may be part-time members of an agile teams, despite the fact that part-time participation is not encouraged in agile teams (Cervone, 2011; Hass, 2007). Considering the option of part-time tem membership, as done by Chau & Maurer (2004) and Schwaber & Beedle (2002), has in fact been left behind, as part-time membership of an agile team is already reality.

8.4 Answer to the Main Research Question

This dissertation included in its introduction the main research question:

What is the role of artefacts in the communication in teams involved in agile software development?

In the previous subsections (8.1 - 8.3) the main research question's related sub questions were answered and the ensemble of these answers provides the answer to the main research question.

1. ***Agile and non-agile artefacts play an important role in the communication within agile teams.***

This should not be a surprise as far as agile artefacts, artefacts that are inherent to an ASD, are involved, for instance a product or sprint backlog, or user stories. Agile teams use, in addition, non-agile artefacts, which are associated with traditional software development rather than ASD.

However, agile teams have sound reasons to use artefacts.

2. ***Usage of artefacts in agile teams springs from both own decisions of a self-organizing team as well as external influences.***

Three rationales lead an agile team to decide itself on the usage of artefacts: (1) Team-internal communication leads to functional and technical design artefacts, (2) quality assurance leads to test-related artefacts, and (3) agile teams impose governance, such as architecture or coding standards, on their own activities.

Three other rationales also influence an agile team, where it may or may not have a degree of influence on: (1) Adoption of ASD leads to agile artefacts, (2) external influences impose user-related artefacts, and (3) Fewer non-agile artefacts are used in organizations with more mature software product management.

Usage of agile and non-agile artefacts covers one end of the formal – informal spectrum of communication in agile teams; face-to-face communication the other.

3. ***Not all artefacts are formal artefacts; teams also use fuzzy artefacts, artefacts that are not formally documented, but still explicitly recognized by agile teams.***

The former typology of formal communication (artefacts) and informal communication (face-to-face) does not suffice in agile teams. Agile teams highly and frequently use fuzzy artefacts as an intermediate concept in the scale formal - informal.

8.5 Main Contributions

The contribution of this dissertation has two dimensions: a scientific one and a practitioner's one.

8.5.1 Scientific Contributions

In the introduction of this dissertation it was demonstrated that blending traditional and agile software development is in the heart of current research. The central theme of this dissertation, the role of artefacts in agile team's communication, deepens the understanding of this research area. Where the blending as such may be taken for granted, although not entirely completed, this dissertation adds a next level in describing how blending operationalizes in, in this case, blending agile and non-agile artefacts. With similar work from Gröber (2013) and Bass (2016) and the research from this dissertation, the overview of artefacts usage in the communication in agile teams approaches a mature stage. Coordination in agile teams still is an item in agile research, although the

emphasis is shifting toward large-scale agile and thus to intra-team coordination (Moe & Dingsøyr, 2017).

The current research goes beyond the sole modelling of artefacts and provides initial knowledge about factors that influence agile teams in their artefacts usage. Central in this dissertation is the role of artefacts in communication in agile teams. Although some preliminary work has been done, especially in the clustering of artefacts, motivation for artefacts' usage has not been a topic of interest. In this dissertation several viewpoints have been put forward:

- In first instance a Scrum artefact model has been drawn up, where artefacts have been classified in one dimension as either agile (Scrum) or non-agile (non-Scrum). In another dimension they were classified as either used for supporting the product, for instance a backlog, or supporting the process, for instance a burndown chart.
- As a second endeavour the relation between an organizational factor, maturity, and the usage of artefacts was established.

Finally, agile teams themselves have indicated reasons for their use of certain artefacts, which led to five rationales for artefacts' usage.

With this contribution the theoretical basis of the Agile Manifesto was elaborated, especially where its valuation of "*working software over comprehensive documentation*" is concerned. Usage of artefacts may or may not contradict the Agile Manifesto. Travelling light has often been taken as a motto to describe this dilemma. It could be argued that each artefact de-agilizes ASD. But whatever position is taken in this dispute, knowing rationales for the usage of artefacts contributes to clarification on the issue.

8.5.2 Contributions to Practice

Blending traditional and agile software development methods is also a concern for practitioners, whether as a choice for one of them or a combination of their elements (Table 8.3, adapted from (Scrum Alliance, 2015)).

Table 8.3 - Use of waterfall / agile (Scrum) software development

Statement	% of respondents
Scrum was very successful and that is all that we use now	20
Scrum was successfully introduced in addition to our Waterfall method	23
Scrum was used for some projects and Waterfall for the rest	40
Scrum or Waterfall Scrum was introduced and integrated into our Waterfall method	4
After a thorough evaluation of a project's type, requirements, and parameters, a decision is made to use either	9
We were not successful in introducing Scrum, so we stayed with our Waterfall method	4

Where "*the agile strategy is to defer the creation of all documents as late as possible, creating them just before you need them via a practice called "document late*" (S. W. Ambler, n.d.), practitioners, i.e. agile teams, have to find a way to decide when and which documentation artefacts are to be used. Apart from the timing of agile documents deference also leads to less documents, comparing agile to traditional software development (S. W. Ambler, n.d.).

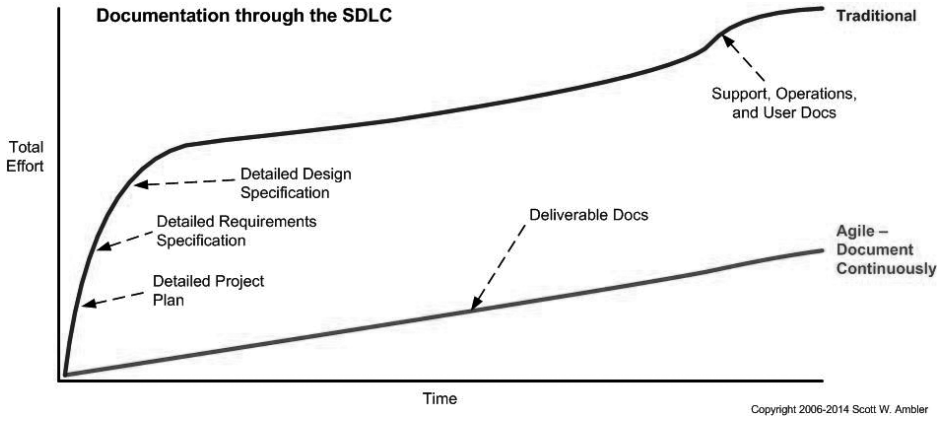

Figure 8.2 - Documentation through the software development life cycle

ASD has a connotation for prescribing not to use documentation artefacts. Although this is denied over and over, the thought sticks to it. The current research proves that agile teams themselves are mainly responsible for using quite many artefacts. Practitioners participating in an agile team may find themselves supported in pursuing adequate documentation, whether agile or, against the stream, non-agile.

8.6 Limitations and Future Research

The research described in this dissertation went off in the constructive worldview, combined with some elements of the pragmatic worldview (refer to Section 1.2.1). In these combined viewpoints, case study research is natural choice as a research method. Even stronger, "*Case study research is the most widely used qualitative research method in information systems research, and is well suited to understanding the interactions between information technology-related innovations and organizational contexts*" (Darke, Shanks, & Broadbent, 1998).

However, validity of a case study cannot be taken for granted. Its validity depends on four widely used criteria (Yin, 2013):

- Construct validity identifies operational measures for the concepts under study.
- Internal validity is mainly a concern for explanatory case studies (Runeson & Höst, 2008; Yin, 2013). Our case studies were in general exploratory.
- External validity defines the domain to which a case study's findings can be generalized. The use of replication logic is listed as the main guarantee for this (Yin, 2013).
- Reliability should demonstrate that the study can be repeated. The use of a case study protocol and the development of a case study database (Yin, 2013) are the main instruments to increase reliability.

All parts of the research were guided by a CSP. The respective CSP's and the criteria for validity were discussed within the context of each individual part of the current research (Chapters 2 - 7).

The most important limitation of the current research lies in external validity, especially in the choice of organizations involved in the research. First, participating organizations, in clear majority, held offices in the Netherlands. This restriction is primarily caused by the home base of the researchers involved. Although there is no a priori presumption why phenomena in the Netherlands should deviate from those elsewhere, the geographical base of our research is small and may have led to an unknown bias.

Secondly, most of the research was done in SPOs. An SPO produces a software product, as opposed to project, or tailor made, software (Xu & Brinkkemper, 2007). This focus has been maintained throughout the research, except for the research into the criteria for selecting a Scrum too, which was a case study with students. Again, the focus on SPOs may or may not have influenced the research

and its findings. It remains that part of the software business, especially where project software is concerned, is left outside of consideration.

Hodkinson & Hodkinson (2001) list eight limitations for case studies:

1. They contain too much data for easy analysis.
2. They are expensive, if attempted on a large scale.
3. They examine complexity which is difficult to represent simply.
4. They do not lend themselves to numerical representation.
5. They are not generalizable in the conventional sense.
6. They are strongest when researcher expertise and intuition are maximised, but this raises doubts about their "objectivity".
7. They are easy to dismiss, by those who do not like the messages that they contain.
8. They cannot answer a large number of relevant and appropriate research questions.

Comparing these limitations with the current set of case studies, it is easily seen that items 1 – 2 and 8 are not applicable. Our case studies were fairly small in their size and were not set up to answer a large number of research questions. Item 4 is only partially applicable in our research as some numerical calculations were applied in our research, for instance comparing and counting a number of artefacts over several different teams. Item 6 was, however partially, addressed by using a CSP for all parts of the research. However, those and the other items are legitimate concerns, which touch upon the core of using case studies. Applying precautionary measures as construct, internal and external

validity, and reliability addresses the items, but does not eliminate them. They precautionary measures were discussed for each of the elements of the research, but the mentioned treats are inherent limitations in case study research.

8.6.1 Future Directions

Some future directions were already identified in various parts of this research.

8.6.1.1 Motivations for using artefacts

The research addressed the motivation for using artefacts from two viewpoints:

- A relationship, although weak, between SPM maturity and the use of artefacts was established.
- Five rationales for using artefacts were derived from the viewpoint of one or two key members of agile teams.

This dissertation cannot claim to be the final answer to the usage of artefacts in communication in agile team.

The research so far proved a relationship between SPM maturity and artefacts usage. It is not only an ambition to improve this to a causal relationship, but also to identify other factors influencing agile teams in their choice for (non-agile) artefacts. Candidates here would be team composition (size, experience), project characteristics or explicit team decisions as opposed to maturity of an organization as a whole. This would provide an answer to the question whether, especially non-agile, artefacts emerge from an agile team or are used for reasons which originate from outside the team. More general, how does an agile team reach a balance between agile and non-agile artefacts?

The research also was exploratory and led to directions to group rationales with respect to the use of artefacts. However, to allow for more robust conclusions, viewpoints from several, if not all, team members should be taken into account. This would require in-depth interviews with members from one team, thus focusing on depth, rather than breadth. This would also strengthen the evidence for the current group of rationales.

As with the first point, the research did not explicitly investigate characteristics nor context of teams and organizations. As in other areas, context again does of course matter in the rationales for artefacts usage. Incorporation of such a context would also allow for more concrete definitions, templates or examples of artefacts.

8.6.1.2 Artefacts in agile software development

The overview of artefacts, agile or not, used in ASD team has reached a rather mature stage. However, confirmation of results with other organizations, whether or not as a replication study, would be welcome. Although the research also revealed the existence of fuzzy artefacts, it does not address their degree of fuzziness, compliant with Zadeh's original work (Zadeh, 1965). Exploring whether or not such a further classification could be achieved remains an issue for future research.

8.6.1.3 Diverging direction

Another future direction, also inspired by professional interest, is to think about this dissertation's implications for education. If agile software development methods blend artefacts, traditional and agile, what does this mean for our prospective agile team members, e.g. students. In a case study it was advised to balance "*ASD topics with occasional instances of more traditional ones during the*

classes and labs brings the necessary "reality check"" (Devedžić & Milenković, 2011). How to put this into a curriculum in one (agile) course or two separate courses is a topic for future research.

8.7 Final reflections

When I seriously started thinking about working towards a PhD, I was naïve. I did not fully realize when it was the last time I studied full-time. Yes, that was back in 1986. It is not that I did not keep up with developments with respect to the content. After all, the Agile Manifesto was not even written in 1986. No, what surprised me most was the rust on my research skills. This was an area where I really needed a reset.

In hindsight, it was an overambitious endeavour to aspire to complete this dissertation in four years, with slightly more than half of my working hours devoted to it. For the other half, I was just as involved in teaching as I ever was, at least for the last twenty years. Not finalizing the dissertation in four years sometimes felt as a failure. On the other hand, concluding it now, after four years of half-timework and almost three more years in a combination of a day's part-time work and stolen hours, is not that bad after all.

Where then did I spent my time on during the route to this dissertation? As mentioned before, on refreshing my research skills. Detailing the chain of evidence from the words spoken in an interview through grounded theory analysis to valid conclusions. Writing in an academic style, which in the beginning, and to be honest until the very last day, is not my usual or preferred way of writing; I guess I am better at more popular texts. Phrasing conclusions in such a way that they contain firm statements about brand-new theoretical insights, also of use to practitioners.

Where have I lost time on the route to this dissertation? I think I spent too much time on thinking before acting. I strived for a complete and thorough PhD research proposal. For my first research I had to set up a case study protocol, fitting in all its details. For too long, I neglected the products of science: conference papers and journal articles.

But finally, and most important, what was the fun along the way? I loved doing the actual research: interviewing agile team members myself, incorporating my own bachelor students as guinea pigs, preparing master students for doing (partly) my research outside. There was no greater joy than balancing base or derived materials, looking from different angles, going back and again forward, to finally come to results. Finally presenting parts of the research at a conference or in a journal, and the talks or discussions with colleagues are that satisfying (and secretively looking at the number of reads and citations afterwards).

I rest my pen …

Bibliography

Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010). Agility and Architecture: Can They Coexist? *IEEE Software*, *27*(2), 16–22. http://doi.org/10.1109/MS.2010.36

Abrahamsson, P., Oza, N., & Siponen, M. T. (2010). Agile Software Development Methods: A Comparative Review. In T. Dingsøyr, T. Dybå, & N. B. Moe (Eds.), *Agile Software Development: Current Research and Future Directions* (pp. 31–59). Berlin Heidelberg: Springer. http://doi.org/10.1007/978-3-642-12575-1_3

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New Directions on Agile Methods: A Comparative Analysis. In *Proceedings of the 25th International Conference on Software Engineering, May 3-5, 2003, Portland (OR), USA* (pp. 244–254). IEEE. http://doi.org/10.1007/978-3-642-12575-1_3

Agile is the new normal - Adopting Agile project management. (2017). Retrieved July 1, 2018, from softwaretestinggenius.com/docs/4aa5-7619.pdf

Allan, G. (2003). The use of grounded theory as a research method: warts & all. In A. Brown (Ed.), *Proceedings of the 2nd European Conference on Research Methodology for Business and Management Studies, 2003* (pp. 9–19). Management Centre International Limited.

Alomar, N., Almobarak, N., Alkoblan, S., Alhozaimy, S., & Alharbi, S. (2016). Usability Engineering of Agile Software Project Management Tools. In A. Marcus (Ed.), *Design, User Experience, and Usability: Design Thinking and Methods -Proceedings of the 5th International Conference, (DUXU 2016), Part of HCI International 2016, Toronto, Canada, 17–22 July, 2016 Part I* (pp. 197–208). Springer International Publishing. http://doi.org/10.1007/978-3-319-40409-7_20

Alyahya, S., Alqahtani, M., & Maddeh, M. (2016). Evaluation and improvements for agile planning tools. In *Proceedings of the IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA), Baltimore, USA, 8-10 June 2016* (pp. 217–224). IEEE. http://doi.org/10.1109/SERA.2016.7516149

Ambler, S. (2002). *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. John Wiley & Sons.

Ambler, S. W. (n.d.). Core Practices for Agile/Lean Documentation. Retrieved August 31, 2018, from http://www.agilemodeling.com/essays/agileDocumentationBestPractices.htm

Aranda, J., & Venolia, G. (2009). The secret life of bugs: Going past the errors and omissions in software repositories. In *Proceedings of the 31st International Conference on Software Engineering (ICSE '09), Vancouver, Canada, 16 - 24 May, 2009* (pp. 298–308). IEEE. http://doi.org/10.1109/ICSE.2009.5070530

Aymerich, B., Díaz-Oreiro, I., Guzmán, J. C., López, G., & Garbanzo, D. (2018). Software Development Practices in Costa Rica: A Survey. In T. Z. Ahram (Ed.), *Advances in Artificial Intelligence, Software and Systems Engineering - Joint Proceedings of the International Conference on Human Factors in Artificial Intelligence (AHFE 2018) and Social Computing, Software and Systems Engineering, The Human Side of Ser* (pp. 122–132). Cham: Springer. http://doi.org/10.1007/978-3-319-94229-2_13

Azizyan, G., Magarian, M. K., & Kajko-Mattson, M. (2011). Survey of agile tool usage and needs. In *Proceedings of the Agile Conference (Agile 2011), Salt Lake City (UT), USA, 7-13 August, 2011* (pp. 29–38). http://doi.org/10.1109/AGILE.2011.30

Bass, J. M. (2016). Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology*, *75*(C), 1–16. http://doi.org/10.1016/j.infsof.2016.03.001

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. *XP Series*. http://doi.org/10.1136/adc.2005.076794

Beck, K. (2003). *Test-driven development : by example*. Addison-Wesley.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Agile Manifesto. Retrieved July 1, 2018, from http://agilemanifesto.org/

Bekkers, W., Brinkkemper, S., van den Bemd, L., Mijnhardt, F., Wagner, C., & van de Weerd, I. (2012). Evaluating the Software Product Management Maturity Matrix. In *Proceedings of the 20th IEEE International Requirements Engineering Conference (RE 2012), Chicago (IL), USA, 24-28 September 2012* (pp. 51–60). IEEE. http://doi.org/10.1109/RE.2012.6345839

Bekkers, W., Spruit, M., Weerd, I. van de, Vliet, R. van, & Mahieu, A. (2010). A Situational Assessment Method for Software Product Management. In *Proceedings of the 18th European Conference on Information Systems (ECIS 2010), Pretoria, South Africa, 7 - 9 June, 2010* (p. Paper 22).

Bekkers, W., Weerd, I. van de, Spruit, M., & Brinkkemper, S. (2010). A Framework for Process Improvement in Software Product Management. In A. Riel, R. O'Connor, S. Tichkiewitch, & R. Messnarz (Eds.), *Systems, Software and Services Process Improvement - Proceedings of the 17th European Conference, EuroSPI 2010, Grenoble, France, 1-3 September, 2010.* (pp. 1–12). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-15666-3

Bennett, N. (2017). *Managing successful projects with PRINCE2*. TSO.

Bjørnson, F. O., & Dingsøyr, T. (2008). Knowledge management in software engineering: A systematic review of studied concepts, findings and research methods used. *Information and Software Technology*, *50*(11), 1055–1068. http://doi.org/10.1016/j.infsof.2008.03.006

Blijleven, V. (2012). Scrum Development Process from a Method Engineering Perspective. Retrieved from http://foswiki.cs.uu.nl/foswiki/MethodEngineering/Scrumdevelopmentprocess20112012

Boehm, B. W. (1983). Seven basic principles of software engineering. *Journal of Systems and Software*, *3*(1), 3–24. http://doi.org/10.1016/0164-1212(83)90003-1

Boehm, B. W. (2002). Get ready for agile methods, with care. *Computer*, *35*(1), 64–69. http://doi.org/10.1109/2.976920

Boehm, B. W., Brown, J. R., & Lipow, M. (1976). Quantitative evaluation of software quality. In *Proceedings of the 2nd International Conference on Software Engineering (ICSE), San Francisco (CA), USA, 13-15 October , 1976* (pp. 592–605). IEEE Computer Society Press.

Bootla, P., Rojanapornpun, O., & Mongkolnam, P. (2015). Necessary skills and attitudes for development team members in Scrum: Thai experts' and practitioners's perspectives. In *Proceedings of the 12th International Joint Conference on Computer Science and Software Engineering (JCSSE), Songkhla, Thailand, 22-24 July, 2015* (pp. 184–189). IEEE. http://doi.org/10.1109/JCSSE.2015.7219793

Breivold, H. P., Sundmark, D., Wallin, P., & Larsson, S. (2010). What Does Research Say about Agile and Architecture? In *2010 Fifth International Conference on Software Engineering Advances* (pp. 32–37). IEEE. http://doi.org/10.1109/ICSEA.2010.12

Bustamante, A. F., & Rincón, R. D. (2017). WYDIWYN – What You Define, Is What You Need: Defining Agile/Traditional Mixed Methodologies. In J. Mejia, M. Muñoz, Á. Rocha, Y. Quiñonez, & J. Calvo-Manzano (Eds.), *Trends and Applications in Software Engineering - Proceedings of the 6th International Conference on Software Process Improvement (CIMPS 2017), Zacatecas, Mexico, 18-20 October 2017* (pp. 35–44). Springer, Cham. http://doi.org/10.1007/978-3-319-69341-5_4

Bustard, D., Wilkie, G., & Greer, D. (2013). The Diffusion of Agile Software Development: Insights from a Regional Survey. In R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry, & M. Lang (Eds.), *Information Systems Development: Reflections, Challenges and New Directions* (pp. 219–230). New York, NY: Springer New York. http://doi.org/10.1007/978-1-4614-4951-5_18

Cao, L., & Ramesh, B. (2008). Agile Requirements Engineering Practices: An Empirical Study. *IEEE Software*, *25*(1), 60–67.

Cataldo, M., Herbsleb, J. D., & Carley, K. M. (2008). Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM 2008), Kaiserslautern, Germany, 9-10 October, 2008* (pp. 2–11). http://doi.org/10.1145/1414004.1414008

Cervone, H. F. (2011). Understanding agile project management methods using Scrum. *OCLC Systems & Services: International Digital Library Perspectives*, *27*(1), 18–22. http://doi.org/10.1108/10650751111106528

Chau, T., & Maurer, F. (2004). Knowledge sharing in agile software teams. In W. Lenski (Ed.), *Logic versus Approximation: Essays Dedicated to Michael M. Richter on the Occasion of his 65th Birthday (Lecture Notes in Computer Science 3075)* (Vol. 3075, pp. 173–183). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-25967-1_12

Chau, T., Maurer, F., & Melnik, G. (2003). Knowledge sharing: agile methods vs. Tayloristic methods. In *Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WET ICE 2003, Linz, Austria, June 9-11, 2003* (pp. 302–307). IEEE. http://doi.org/10.1109/ENABL.2003.1231427

Chetankumar, P., & Ramachandran, M. (2009). Agile Maturity Model (AMM): A Software Process Improvement framework for Agile Software Development Practices. *International Journal of Software Engineering*, *2*(1), 3–28.

Chow, T., & Cao, D.-B. (2008). A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, *81*(6), 961–971. http://doi.org/10.1016/j.jss.2007.08.020

Clarke, P., & O'Connor, R. V. (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Software Technology*, *54*(5), 433–447. http://doi.org/10.1016/j.infsof.2011.12.003

CMMI Product Team. (2010). *CMMI for Development, Version 1.3*.

Cockburn, A. (2002). *Agile Software Development*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Cohn, M. (2004). *User Stories Applied For Agile Software Development*. Addison-Wesley.

Coplien, J. O., & Harrison, N. B. (2005). *Organizational Patterns of Agile Software Development*. Pearson Prentice Hall. http://doi.org/0131467409

Corbin, J., & Strauss, A. (2015). *Basics of Qualitative Research - Techniques and Procedures for Developing Grounded Theory* (4th editio). Thousand Oaks (CA), USA: Sage Publications. http://doi.org/10.4135/9781452230153

Cowan, R., David, P. A., & Foray, D. (2000). The explicit economics of knowledge codification and tacitness. *Industrial and Corporate Change*, *9*(2), 211–253. http://doi.org/10.1093/icc/9.2.211

Creswell, J. W., & Creswell, J. D. (2017). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches* (5th ed.). Thousand Oaks (CA): Sage Publications. http://doi.org/10.1007/s13398-014-0173-7.2

Crowder, J. A., & Friess, S. (2015). Productivity Tools for the Modern Team. In - (Ed.), *Agile Project Management: Managing for Success* (pp. 43–48). Springer International Publishing. http://doi.org/10.1007/978-3-319-09018-4_4

Curtis, B., Krasner, H., & Iscoe, N. (1988). A Field Study of the Software Design Process for Large Systems. *Communications of the ACM*, *31*(11), 1268–1287. http://doi.org/10.1145/50087.50089

Damian, D., Marczak, S., & Kwan, I. (2007). Collaboration Patterns and the Impact of Distance on Awareness in Requirements-Centred Social Networks. *15th IEEE International Requirements Engineering Conference RE 2007*, 59–68. http://doi.org/10.1109/RE.2007.51

Darke, P., Shanks, G., & Broadbent, M. (1998). Successfully completing case study research: combining rigour, relevance and pragmatism. *Information Systems Journal*, *8*(4), 273–289. http://doi.org/10.1046/j.1365-2575.1998.00040.x

Davies, R. (2001). The Power of Stories. In *Extreme Programming and Agile Methods — XP/Agile Universe 2001*.

Davis, F. D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly*, *13*(2), 319–340.

Davis, F. D., Bagozzi, R. P., & Warshaw, P. R. (1989). User Acceptance Of Computer Technology: A Comparison Of Two Theoretical Models. *Management Science*, *35*(8), 982–1003.

Deemer, P., Benefield, G., Larman, C., & Vodde, B. (2010). *The Scrum primer - A Lightweight Guide to the Theory and Practice of Scrum (Version 2.0)*.

Devedžić, V., & Milenković, S. R. (2011). Teaching Agile Software Development: A Case Study. *IEEE Transactions on Education*, *54*(2), 273–278. http://doi.org/10.1109/TE.2010.2052104

Dijkstra, O. (2013). *Extending the Agile Development Discipline to Deployment - The Need For a Holistic Approach*. Utrecht University.

Dingsøyr, T., Bjørnson, F. O., Moe, N. B., Rolland, K., & Seim, E. A. (2018). Rethinking coordination in large-scale software development. In *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '18), Gothenburg, Sweden – 27 May, 2018* (pp. 91–92). New York: ACM Press. http://doi.org/10.1145/3195836.3195850

Dingsøyr, T., & Moe, N. B. (2014). Towards Principles of Large-Scale Agile Development - A Summary of the Workshop at XP2014 and a Revised Research Agenda. In T. Dingsoyr, N. B. Moe, R. Tonelli, S. Counsell, C. Gencel, & K. Petersen (Eds.), *Revised Selected Papers of the XP2014) International Workshops, Rome, Italy, 26-30 May 2014* (pp. 1–8). Springer, Cham. http://doi.org/10.1007/978-3-319-14358-3_1

Dix, A. (1994). Computer Supported Cooperative Work: A Framework. In D. Rosenberg & C. Hutchison (Eds.), *Design Issues in CSCW* (pp. 9–26). Springer, London. http://doi.org/10.1007/978-1-4471-2029-2_2

Downey, S., & Sutherland, J. (2013). Scrum Metrics for Hyperproductive Teams: How They Fly like Fighter Aircraft. In *Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS), Wailea (HI), USA, 7-10 January, 2012* (pp. 4870–4878). IEEE. http://doi.org/10.1109/HICSS.2013.471

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9–10), 833–859. http://doi.org/10.1016/j.infsof.2008.01.006

Dyck, A., Penners, R., & Lichter, H. (2015). Towards definitions for release engineering and DevOps. In *Proceedings of the 3rd International Workshop on Release Engineering (RELENG), Florence, Italy, 16 - 24 May, 2015* (pp. 3–3). IEEE Press.

Dzamashvili Fogelström, N., Gorschek, T., Svahnberg, M., & Olsson, P. (2010). The Impact of Agile Principles on Market-Driven Software Product Development. *Journal of Software Maintenance and Evolution Research and Practice*, *22*(1), 53–80. http://doi.org/10.1002/smr.453

Ebert, C. (2007). The impacts of software product management. *Journal of Systems and Software*, *80*(6), 850–861. http://doi.org/10.1016/j.jss.2006.09.017

Ebert, C., & Brinkkemper, S. (2014). Software product management – An industry evaluation. *Journal of Systems and Software*, *95*(0), 10–18. http://doi.org/http://dx.doi.org/10.1016/j.jss.2013.12.042

Ehrlich, K., & Chang, K. (2006). Leveraging expertise in global software teams: Going outside boundaries. In *Proceedings of the International Conference on Global Software Engineering (ICGSE '06), Florianopolis, Brazil, 16-19 October, 2006* (pp. 149–158). IEEE. http://doi.org/10.1109/ICGSE.2006.261228

Eisenhardt, K. M. (1989). Building Theories from Case Study Research. *Academy of Management Review*, *14*(4), 532–550. http://doi.org/10.5465/AMR.1989.4308385

Engum, E. A., Racheva, Z., & Daneva, M. (2009). Sprint Planning with a Digital Aid Tool: Lessons Learnt. In *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA '09), Patras, Greece, 27-29 August, 2009* (pp. 259–262). IEEE. http://doi.org/10.1109/SEAA.2009.68

Felderer, M., Winkler, D., & Biffl, S. (2017). Hybrid Software and System Development in Practice: Initial Results from Austria. In M. Felderer, D. M. Fernández, B. Turhan, M. Kalinowsk, If. Sarro, & D. Winkler (Eds.), *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbruck, Austria, 29 November – 1 December, 2017* (pp. 435–442). Cham: Springer. http://doi.org/10.1007/978-3-319-69926-4_33

Femmer, H., Kuhrmann, M., Stimmer, J., & Junge, J. (2014). Experiences from the Design of an Artifact Model for Distributed Agile Project Management. In *Proceedings of the IEEE 9th International Conference on Global Software Engineering (ICGSE 2014), Sjanghai, China, 18-21 August, 2014* (pp. 1–5). Sjanghai (China): IEEE. http://doi.org/10.1109/ICGSE.2014.9

Ferreira, J., Sharp, H., & Robinson, H. (2010). Values and Assumptions Shaping Agile Development and User Experience Design in Practice. In A. Martin, X. Wang, & E. Whitworth (Eds.), *Agile Processes in Software Engineering and Extreme Programming - Proceedings of the 11th International Conference on Agile Software Development, XP 2010, Trondheim, Norway, 1-4 June, 2010* (pp. 178–183). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-13054-0_15

Ferreira, J., Sharp, H., & Robinson, H. (2011). User experience design and agile development: managing cooperation through articulation work. *Software: Practice and Experience*, *41*(9), 963–974. http://doi.org/10.1002/spe.1012

Fowler, M. (2004). Is Design Dead? Retrieved October 25, 2017, from https://www.martinfowler.com/articles/designDead.html#PlannedAndEvolutionaryDesign

Fricker, S. A. (2012). Software Product Management. In A. Maedche, A. Botzenhardt, & L. Neer (Eds.), *Software for People - Fundamentals, Trends and Best Practices* (pp. 53–81). Springer Berlin Heidelberg.

Garcia, A., Silva da Silva, T., & Selbach Silveira, M. (2017). Artifacts for Agile User-Centered Design: A Systematic Mapping. In *Proceedings of the Hawaii International Conference on System Sciences (HICSS-50), Waikoloa Village, Hawaii, 4-7 January , 2017.*

Georgiadou, E. (2003). Software Process and Product Improvement: A Historical Perspective. *Cybernetics and Systems Analysis*, *39*(1), 125–142. http://doi.org/10.1023/A:1023833428613

Gill, A. Q., Henderson-Sellers, B., & Niazi, M. (2016). Scaling for agility: A reference model for hybrid traditional-agile software development methodologies. *Information Systems Frontiers*, 1–27. http://doi.org/10.1007/s10796-016-9672-8

Glaiel, F., Moulton, A., & Madnick, S. (2013). Agile project dynamics: A system dynamics investigation of agile software development methods. In *Proceedings of the 31st International Conference of the System Dynamics Society, Cambridge (MA) , USA, July 21–25, 2013.*

Glaser, B. G. (1992). *Basics of Grounded Theory Analysis: Emergence vs. Forcing.* Mill Valley (CA), USA: Sociology Press.

Glaser, B. G., & Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*. Chicago: Aldine.

Gomes, A., Pettersson, A., & Gorschek, T. (n.d.). *Market-Driven Requirements Engineering Process Model, version 1.0*.

Grant, R. M. (1996). Toward a knowledge-based theory of the firm. *Strategic Management Journal*, *17*(S2), 109–122. http://doi.org/10.1002/smj.4250171110

Gregory, P., Barroca, L., Sharp, H., Deshpande, A., & Taylor, K. (2016). The challenges that challenge: Engaging with agile practitioners' concerns. *Information and Software Technology*, *77*, 92–104. http://doi.org/10.1016/j.infsof.2016.03.003

Gröber, M. (2013). *Investigation of the Usage of Artifacts in Agile Methods*. Technischen Universität München.

Guba, E. G. (1990). The Alternative Paradigm Dialog. In E. G. Guba (Ed.), *The paradigm dialog* (pp. 17–30). Newbury Park (CA): Sage Publications. http://doi.org/10.1080/1357527032000140352

Gupta, R. M. (2011). *Project Management*. PHI Learning Pvt. Ltd.

Hass, K. B. (2007). The Blending of Traditional and Agile Project Management. *PM World Today*, *IX*(V), 1–8.

Hellmann, T. D., Sharma, A., Ferreira, J., & Maurer, F. (2012). Agile Testing: Past, Present, and Future - Charting a Systematic Map of Testing in Agile Software Development. In *Proceedngs of Agile 2012, Dallas (X), USA, 13-17 August 2012* (pp. 55–63). IEEE. http://doi.org/10.1109/Agile.2012.8

Herbsleb, J. D., & Mockus, A. (2003). An empirical study of speed and communication in globally distributed software development. *IEEE Transactions on Software Engineering*, *29*(6), 481–494. http://doi.org/10.1109/TSE.2003.1205177

Herbsleb, J. D., & Moitra, D. (2001). Global software development. *IEEE Software*, *18*(2), 16–20. http://doi.org/10.1109/52.914732

Hodkinson, P., & Hodkinson, H. (2001). The strengths and limitations of case study research. In *Proceedings of the Learning and Skills Development Agency (LSDA) Conference, Cambridge, United Kingdom, 2001*.

Hossain, E., & Babar, M. A. (2009). Risk identification and mitigation processes for using scrum in global software development: A conceptual framework. In *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC '09), Penang, maaysia, 1-3 December, 2009* (pp. 457–464).

Hossain, E., Babar, M. A., & Paik, H. (2009). Using Scrum in Global Software Development: A Systematic Literature Review. In *Proceedings of the Fourth IEEE International Conference on Global Software Engineering (ICGSE), Limerick, Ireland, 13-16 July, 2009* (pp. 175–184). IEEE. http://doi.org/10.1109/ICGSE.2009.25

Hummel, M. (2014). State-of-the-Art: A Systematic Literature Review on Agile Information Systems Development. In *Proceedings of the 47th Hawaii International Conference on System Sciences (HICSS), Waikoloa (HI,) USA, 6-9 January, 2014* (pp. 4712–4721). IEEE. http://doi.org/10.1109/HICSS.2014.579

Hummel, M., Rosenkranz, C., & Holten, R. (2013). The Role of Communication in Agile Systems Development. *Business & Information Systems Engineering*, *5*(5), 343–355. http://doi.org/10.1007/s12599-013-0282-4

IEEE. (2006). *Standard for Developing a Software Project Life Cycle Process. IEEE Standard 1074-2006.*

ISO/IEC 33002:2015. (2015).

ISO. (2011). ISO/IEC/IEEE 26515:2011 Systems and software engineering - Developing user documentation in an agile environment.

Jurafsky, D., & Martin, J. H. (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition* (2nd editio). Prentice Hall.

Kajornboon, A. B. (2005). Using Interviews as Research Instruments. *E-Journal for Researching Teachers (EJRT)*, *2*(1).

Karunakaran, S. (2013). Impact of Cloud Adoption on Agile Software Development. In Z. Mahmood & S. Saeed (Eds.), *Software Engineering Frameworks for the Cloud Computing Paradigm* (pp. 213–234). Springer London. http://doi.org/10.1007/978-1-4471-5031-2_10

Kincheloe, J. L., & Steinberg, S. R. (1998). Students as Researchers: Critical Visions, Emancipatory Insights - Shirley R. Steinberg, Joe L. Kincheloe. In Shirley R. Steinberg & Joe L. Kincheloe (Eds.), *Students as Researchers: Creating Classrooms that Matter* (pp. 2–19). London (UK)/Bristol (USA): Farmer Press.

Kittlaus, H.-B., & Clough, P. N. (2009). *Software Product Management and Pricing: Key Success Factors for Software Organizations*. Springer-Verlag Berlin Heidelberg.

Klünder, J., Hohl, P., Fazal-Baqaie, M., Krusche, S., Küpper, S., Linssen, O., & Prause, C. R. (2017). HELENA Study: Reasons for Combining Agile and Traditional Software Development Approaches in German Companies. In M. Felderer, D. M. Fernández, B. Turhan, M. Kalinowski, F. Sarro, & D. Winkler (Eds.), *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES, 2017), Innsbruck, Austria, 29 November –1 December, 2017* (pp. 428–434). Springer, Cham. http://doi.org/10.1007/978-3-319-69926-4_32

Kraut, R. E., & Streeter, L. A. (1995). Coordination in software development. *Communications of the ACM*, *38*(3), 69–81. http://doi.org/10.1145/203330.203345

Kropp, M., & Meier, A. (2013). Teaching Agile Software Development at University Level: Values, Management, and Craftsmanship. In *Proceedings of the IEEE 26th Conference on Software Engineering Education and Training (CSEE&T), San Francisco (CA), USA, 19-21 May 2013* (pp. 179–188). IEEE. http://doi.org/10.1109/CSEET.2013.6595249

Kruchten, P. (2000). *The Rational Unified Process: An Introduction* (3rd editio). Addison-Wesley Professional.

Kruchten, P. (2010). Software Architecture and Agile Software Development - A Clash of Two Cultures? In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE '10, Cape Town, South Africa, May 2-8, 2010* (Vol. 2, pp. 497–498). New York, New York, USA: ACM Press. http://doi.org/10.1145/1810295.1810448

Kuhrmann, M., Diebold, P., MacDonell, S., & Münch, J. (2017). 2nd Workshop on Hybrid Development Approaches in Software Systems Development. In M. Felderer, D. M. Fernández, B. Turhan, M. Kalinowski, F. Sarro, & D. Winkler (Eds.), *Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbrück, Austria, 29 November - 1 December, 2017* (pp. 397–403). Springer International Publishing. http://doi.org/10.1007/978-3-319-69926-4_28

Kuhrmann, M., Diebold, P., Munch, J., Tell, P., Trektere, K., Mc Caffery, F., … Prause, C. (2018). Hybrid Software Development Approaches in Practice: A European Perspective. *IEEE Software*, Advance online publication. http://doi.org/10.1109/MS.2018.110161245

Kuhrmann, M., Méndez Fernández, D., & Gröber, M. (2013). Towards Artifact Models as Process Interfaces in Distributed Software Projects. In *Proceedings of the IEEE 8th International Conference on Global Software Engineering (ICGSE), Bari, Italy, 26-29 August, 2013* (pp. 11–20). Bari (Italy): IEEE. http://doi.org/10.1109/ICGSE.2013.11

Kuhrmann, M., Münch, J., Diebold, P., Linssen, O., & Prause, C. (2016). On the Use of Hybrid Development Approaches in Software and Systems Development: Construction and Test of the HELENA Survey. In *Proceedings of the Annual Special Interest Group Meeting Projektmanagement und Vorgehensmodelle 2016, Paderborn, Germany* (pp. 59–68). Bonn: Gesellschaft für Informatik.

Kuhrmann, M., Münch, J., Tell, P., & Diebold, P. (2018). Summary of the 1st International Workshop on Hybrid Development Approaches in Software Systems Development. *ACM SIGSOFT Software Engineering Notes*, *42*(4), 18–20. http://doi.org/10.1145/3149485.3149519

Kwan, I., & Damian, D. (2011). The Hidden Experts in Software-Engineering Communication (NIER Track). In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11), Waikiki, Honolulu (HI), USA, 21-28 May, 2011* (pp. 800–803). ACM. http://doi.org/10.1145/1985793.1985906

Kwan, I., Schröter, A., & Damian, D. (2011). Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project. *IEEE Transactions on Software Engineering*, *37*(3), 307–324. http://doi.org/10.1109/TSE.2011.29

Lagerberg, L., & Skude, T. (2013). *The impact of agile principles and practices on large-scale software development projects : A multiple-case study of two software development projects at Ericsson*. Linköping University (Sweden).

Leppänen, M. (2013). A Comparative Analysis of Agile Maturity Models. In R. Pooley, J. Coady, C. Schneider, H. Linger, C. Barry, & M. Lang (Eds.), *Information Systems Development - Reflections, Challenges and New Directions* (pp. 329–343). New York, NY: Springer New York. http://doi.org/10.1007/978-1-4614-4951-5_27

Lienitz, P. (2017). The pros and cons of Waterfall Software Development | DCSL Software Ltd. Retrieved November 4, 2018, from https://www.dcslsoftware.com/pros-cons-waterfall-software-development/

Liskin, O. (2015). How Artifacts Support and Impede Requirements Communication. In S. A. Fricker & K. Schneider (Eds.), *Requirements Engineering: Foundation for Software Quality - Proceedings of the 21st International Working Conference (REFSQ 2015), Essen, Germany, 23-26 March, 2015* (pp. 132–147). Springer International Publishing. http://doi.org/10.1007/978-3-319-16101-3_9

Maimbo, H. (2005). Designing a Case Study Protocol for Application in IS research. In *Proceedings of the 9th Asia Conference on Information Systems (PACIS 2005), Bangkok, Thailand, 7-10 July, 2005* (pp. 1281–1292). University of Hong Kong: PACIS.

Maniuk, I. (2016). Implementation of Waterfall Methodology. Retrieved November 4, 2018, from https://hygger.io/blog/implementation-of-waterfall-methodology/

Martin, A., Biddle, R., & Noble, J. (2010). An Ideal Customer: A Grounded Theory of Requirements Elicitation, Communication and Acceptance on Agile Projects. In T. Dingsøyr, T. Dybå, & N. B. Moe (Eds.), *Agile Software Development: Current Research and Future Directions* (pp. 111–141). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-12575-1_6

Melo, C. de O., Santos, V., Katayama, E., Corbucci, H., Prikladnicki, R., Goldman, A., & Kon, F. (2013). The evolution of agile software development in Brazil. *Journal of the Brazilian Computer Society*, *19*(4), 523–552. http://doi.org/10.1007/s13173-013-0114-x

Méndez Fernández, D., Penzenstadler, B., Kuhrmann, M., & Broy, M. (2010). A Meta Model for Artefact-Orientation: Fundamentals and Lessons Learned in Requirements Engineering. In D. C. Petriu, N. Rouquette, & H. Oystein (Eds.), *Model Driven Engineering Languages and Systems - Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems, MODELS 2010, Oslo, Norway, October 3-8, 2010* (pp. 183–197). Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-16129-2_14

Mertens, D. M. (2010). Transformative Mixed Methods Research. *Qualitative Inquiry*, *16*(6), 469–474. http://doi.org/10.1177/1077800410364612

Moe, N. B., & Dingsøyr, T. (2017). Emerging research themes and updated research agenda for large-scale agile development. In *Proceedings of the XP2017 Scientific Workshops on - XP '17* (pp. 1–4). New York, New York, USA: ACM Press. http://doi.org/10.1145/3120459.3120474

Møller, L. S., Nyboe, F. B., Jørgensen, T. B., & Broe, J. J. (2009). A Scrum tool for improving Project Management. In I. Wouters, Fl. K. Man, R. Tieben, S. Offermans, & H. Nagtzaam (Eds.), *Flirting with the Future: Prototypes visions by the next generation - Proceedings of the fifth Student Interaction Design Research Conference, Eindhoven, the Netherlands, 15-17 April, 2009* (pp. 30–32).

Mortensen, M., & Hinds, P. (2002). Fuzzy Teams: Boundary Disagreement in Distributed and Collocated Teams. In P. J. Hinds & S. Kiesler (Eds.), *Distributed Work* (pp. 283–308). MIT Press.

Nakatumba-Nabende, J., Kanagwa, B., Hebig, R., Heldal, R., & Knauss, E. (2017). Hybrid Software and Systems Development in Practice: Perspectives from Sweden and Uganda. In M. Felderer, D. M. Fernández, B. Turhan, M. Kalinowsk, If. Sarro, & D. Winkler (Eds.), *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbruck, Austria, 29 November – 1 December, 2017* (pp. 413–419). Cham: Springer. http://doi.org/10.1007/978-3-319-69926-4_30

Nelson, R. R., & Winter, S. G. (1982). *An evolutionary theory of economic change*. Belknap Press of Harvard University Press.

Nonaka, I. (1994). A Dynamic Theory of Organizational Knowledge Creation. *Organization Science*, *5*(1), 14–37. http://doi.org/10.1287/orsc.5.1.14

Paasivaara, M., & Lassenius, C. (2003). Collaboration practices in global inter-organizational software development projects. *Software Process: Improvement and Practice*, *8*(4), 183–199. http://doi.org/10.1002/spip.187

Paasivaara, M., & Lassenius, C. (2014). Communities of practice in a large distributed agile software development organization - Case Ericsson. *Information and Software Technology*, *56*(12), 1556–1577. http://doi.org/10.1016/j.infsof.2014.06.008

Paez, N., Fontdevila, D., & Oliveros, A. (2017). HELENA Study: Initial Observations of Software Development Practices in Argentina. In M. Felderer, D. M. Fernández, B. Turhan, M. Kalinowsk, If. Sarro, & D. Winkler (Eds.), *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbruck, Austria, 29 November – 1 December, 2017* (pp. 443–449). Cham: Springer. http://doi.org/10.1007/978-3-319-69926-4_34

Parnas, D. L., & Madey, J. (1995). Functional documents for computer systems. *Science of Computer Programming*, *25*(1), 41–61. http://doi.org/10.1016/0167-6423(95)96871-J

Paulk, M. C. (2002). *Agile Methodologies and Process Discipline*.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (1993). *Capability Maturity Model for Software (Version 1.1)*. Pittsburgh (PA), USA.

Petersen, K., & Gencel, C. (2013). Worldviews, Research Methods, and their Relationship to Validity in Empirical Software Engineering Research. In *Proceedngs of the Joint Conference of the 23rd International Workshop on Software Measurement (IWSM) and the 8th International Conference on Software Process and Product Measurement (Mensura), Ankara, Turkey, 23-26 October, 2013* (pp. 81–89). IEEE. http://doi.org/10.1109/IWSM-Mensura.2013.22

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., & Still, J. (2008). The impact of agile practices on communication in software development. *Empirical Software Engineering*, *13*(3), 303–337. http://doi.org/10.1007/s10664-008-9065-9

Polanyi, M., & Sen, A. (2009). *The tacit dimension*. University of Chicago Press.

Prathan, D. (2018). History: Some pictures and PDFs of the Agile Manifesto meeting on 2001. Retrieved January 23, 2019, from https://siamchamnankit.co.th/history-some-pictures-and-pdfs-of-the-agile-manifesto-meeting-on-2001-a33c40bcc2b

Rejab, M. M., Noble, J., & Allan, G. (2014a). Distributing Expertise in Agile Software Development Projects. In *Proceedings of the Agile Conference (AGILE '14), Kissimmee (FL), USA, 28 July - 1 August, 2014* (pp. 33–36). IEEE. http://doi.org/10.1109/AGILE.2014.16

Rejab, M. M., Noble, J., & Allan, G. (2014b). Locating Expertise in Agile Software Development Projects. In G. Cantone & M. Marchesi (Eds.), *Agile Processes in Software Engineering and Extreme Programming - Proceedings of the 15th International Conference XP 2014, Rome, Italy, 26-30 May, 2014* (Vol. LNBIP 179, pp. 260–268). Cham, Heidelberg, New York, Dordrecht, London: Springer International Publishing. http://doi.org/10.1007/978-3-319-06862-6_19

Rejab, M. M., Noble, J., & Marshall, S. (2015). Coordinating Expertise Outside Agile Teams. In C. Lassenius, T. Dingsøyr, & M. Paasivaara (Eds.), *Agile Processes in Software Engineering and Extreme Programming - Proceedings of the 16th International Conference XP 2015, Helsinki, Finland, 25-29 May, 2015* (Vol. LNBIP 212, pp. 141–153). Cham, Heidelberg, New York, Dordrecht, London: Springer International Publishing. http://doi.org/10.1007/978-3-319-18612-2_12

Rodríguez, P., Markkula, J., Oivo, M., & Turula, K. (2012). Survey on Agile and Lean Usage in Finnish Software Industry. In *Proceedings of the ACM-IEEE 12th International Symposium on Empirical Software Engineering and Measurement (ESEM), Lund, Sweden, 17-22 September, 2012* (pp. 139–148). New York, NY, USA: ACM. http://doi.org/10.1145/2372251.2372275

Royce, W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*, *26*, 1–9.

Rubin, E., & Rubin, H. (2011). Supporting agile software development through active documentation. *Requirements Engineering*, *16*(2), 117–132. http://doi.org/10.1007/s00766-010-0113-9

Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164. http://doi.org/10.1007/s10664-008-9102-8

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, *35*(3), 8. http://doi.org/10.1145/1764810.1764814

Santana, C., Queiroz, F., Vasconcelos, A., & Gusmao, C. (2015). Software Process Improvement in Agile Software Development: A Systematic Literature Review. In *Proceedings of the 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2015), Funchal (Madeira), Portugal, 26-28 August, 2015* (pp. 325–332). IEEE. http://doi.org/10.1109/SEAA.2015.82

Schatz, B., & Abdelshafi, I. (2005). Primavera Gets Agile: A Successful Transition to Agile Development. *IEEE Software*, *22*(3), 36–42. http://doi.org/10.1109/MS.2005.74

Schneider, K., Stapel, K., & Knauss, E. (2008). Beyond Documents: Visualizing Informal Communication. In *Proceedings of Requirements Engineering Visualization (REV '08), Barcelona, Spain, 8 September, 2008* (pp. 31–40). IEEE. http://doi.org/10.1109/REV.2008.1

Schwaber, K. (1997). SCRUM Development Process. In J. Sutherland, D. Patel, C. Casanave, G. Hollowell, & J. Miller (Eds.), *Business Object Design and Implementation - OOPSLA '95 Workshop Proceedings, 16 October 1995, Austin (TX), USA* (pp. 117–134). London: Springer. http://doi.org/10.1007/978-1-4471-0947-1_11

Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum* (1st ed.). Upper Saddle River (NJ, USA): Prentice Hall.

Schwaber, K., & Sutherland, J. (2013). *The Scrum guide – The Definitive Guide to Scrum: The Rules of the Game*.

Schwaber, K., & Sutherland, J. (2016). *The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game*.

Schweigert, T., Vohwinkel, D., Korsaa, M., Nevalainen, R., & Biro, M. (2013). Agile Maturity Model: A Synopsis as a First Step to Synthesis. In F. McCaffery, R. V. O'Connor, & R. Messnarz (Eds.), *Systems, Software and Services Process Improvement* (Vol. 364, pp. 214–227). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-39179-8

Scott, E., Pfahl, D., Hebig, R., Heldal, R., & Knauss, E. (2017). Initial Results of the HELENA Survey Conducted in Estonia with Comparison to Results from Sweden and Worldwide. In M. Felderer, D. M. Fernández, B. Turhan, M. Kalinowsk, If. Sarro, & D. Winkler (Eds.), *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbruck, Austria, 29 November – 1 December, 2017* (pp. 404–412). Cham: Springer. http://doi.org/10.1007/978-3-319-69926-4_29

Selleri Silva, F., Soares, F. S. F., Peres, A. L., Azevedo, I. M. de, Vasconcelos, A. P. L. F., Kamei, F. K., & Meira, S. R. de L. (2015). Using CMMI together with agile software development: A systematic review. *Information and Software Technology*, *58*, 20–43. http://doi.org/http://dx.doi.org/10.1016/j.infsof.2014.09.012

Sharp, H., & Robinson, H. (2010). Three 'C's of Agile Practice: Collaboration, Coordination and Communication. In T. Dingsøyr, T. Dybå, & N. B. Moe (Eds.), *Agile Software Development* (pp. 61–85). Springer. http://doi.org/10.1007/978-3-642-12575-1_4

Sharp, H., Robinson, H., & Petre, M. (2009). The role of physical artefacts in agile software development: Two complementary perspectives. *Interacting with Computers*, *21*(1–2), 108–116. http://doi.org/10.1016/j.intcom.2008.10.006

Sommerville, I. (2001). Software Documentation (revised version). In *Software Engineering* (4th ed.). Addison Wesley.

Stankovic, D., Nikolic, V., Djordjevic, M., & Cao, D.-B. (2013). A survey study of critical success factors in agile software projects in former Yugoslavia IT companies. *Journal of Systems and Software*, *86*(6), 1663–1678. http://doi.org/10.1016/j.jss.2013.02.027

Stapel, K., Knauss, E., & Schneider, K. (2009). Using FLOW to Improve Communication of Requirements in Globally Distributed Software Projects. In *Proceedings of the Workshop on Collaboration and Intercultural Issues on Requirements: Communication, Understanding and Softskills (CIRCUS), Atlanta (GA), USA, 31 August 2009* (pp. 5–14). IEEE. http://doi.org/10.1109/CIRCUS.2009.6

Stapel, K., & Schneider, K. (2012). *FLOW-Methode - Methodenbeschreibung zur Anwendung von FLOW* (Software Engineering).

Stapel, K., & Schneider, K. (2014). Managing knowledge on communication and information flow in global software projects. *Expert Systems*, *31*(3), 234–252. http://doi.org/10.1111/j.1468-0394.2012.00649.x

Stapleton, J. (1997). *DSDM, Dynamic Systems Development Method: The Method in Practice*. Addison Wesley Publishing Company.

Stettina, C. J., & Heijstek, W. (2011). Necessary and neglected? An empirical study of internal documentation in agile software development teams. In *Proceedings of the 29th ACM International Conference on Design of Communication (SIGDOC 2011), Pisa, Italy 3-5 October, 2011* (pp. 159–166).

Strode, D. E., & Huff, S. L. (2014). A Coordination Perspective on Agile Software Development. In *Modern Techniques for Successful IT Project management* (pp. 1–28).

Taheri, M., & Sadjad, S. M. (2015). A Feature-Based Tool-Selection Classification for Agile Software Development. In *Proceedings of the 27th International Conference on Software Engineering and Knowledge Engineering (SEKE 2015), Pittsburgh, USA, 6-8 July, 2015*. http://doi.org/10.18293/SEKE2015-234

Tell, P., Pfeiffer, R.-H., & Schultz, U. P. (2017). HELENA Stage 2—Danish Overview. In M. Felderer, D. M. Fernández, B. Turhan, M. Kalinowsk, If. Sarro, & D. Winkler (Eds.), *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbruck, Austria, 29 November – 1 December, 2017* (pp. 420–427). Cham: Springer. http://doi.org/10.1007/978-3-319-69926-4_31

Theocharis, G., Kuhrmann, M., Münch, J., & Diebold, P. (2015). Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. In *Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015), Bolzano, Italy, 2-4 December, 2015* (pp. 149–166). Springer-Verlag New York, Inc. http://doi.org/10.1007/978-3-319-26844-6_11

Turk, D., France. Robert, & Rumpe, B. (2005). Assumptions Underlying Agile Software-Development Processes. *Journal of Database Management*, *16*(4), 62–87. http://doi.org/10.4018/jdm.2005100104

Turner, W. S., Langerhorst, R. P., Hice, G. F., Eilers, H. B., Remmerde, E., & Uijttenbroek, A. A. (1987). *SDM, System Developmet Methodology*. Rijswijk: Pandata.

Uy, E., & Rosendahl, R. (2008). Migrating from SharePoint to a Better Scrum Tool. In *Proceedings of Agile Conference (AGILE'08), Toronto (ON), Canada, 4-8 August, 2008* (pp. 506–512). http://doi.org/10.1109/Agile.2008.69

Venkatesh, V., Brown, S. A., & Bala, H. (2013). *Bridging the qualitative-quantitative divide: Guidelines for conducting mixed methods research in information systems. MIS Quarterly: Management Information Systems* (Vol. 37). Society for Management Information Systems.

Venkatesh, V., & Davis, F. D. (2000). A theoretical extension of the technology acceptance model: Four longitudinal field studies. *Management Science*, *46*(2), 186–204.

Venkatesh, V., Morris, M. G., Davis, G. B., & Davis, F. D. (2003). User Acceptance of Information Technology: Toward a Unified View. *MIS Quarterly*, *27*(3), 425–478.

VersionOne. (2017). *11th Annual State of Agile Report*.

Vijayasarathy, L. R., & Butler, C. W. (2016). Choice of Software Development Methodologies: Do Organizational, Project, and Team Characteristics Matter? *IEEE Software*, *33*(5), 86–94. http://doi.org/10.1109/MS.2015.26

Vlaanderen, K., Jansen, S., Brinkkemper, S., & Jaspers, E. (2011). The agile requirements refinery: Applying SCRUM principles to software product management. *Information and Software Technology*, *53*(1), 58–70. http://doi.org/http://dx.doi.org/10.1016/j.infsof.2010.08.004

Voigt, S., Garrel, J. von, Müller, J., & Wirth, D. (2016). A Study of Documentation in Agile Software Projects. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16), Ciudad Real, Spain, 8-9 September, 2016* (p. Article no. 4). New York, New York, USA: ACM Press. http://doi.org/10.1145/2961111.2962616

Wagenaar, G., Helms, R., Damian, D., & Brinkkemper, S. (2015). Artefacts in Agile Software Development. In P. Abrahamsson, L. Corral, M. Oivo, & B. Russo (Eds.), *Product-Focused Software Process Improvement - Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015), Bolzano, Italy, December 2-4, 2015* (Vol. LNCS 9459, pp. 133–148). Springer International Publishing. http://doi.org/10.1007/978-3-319-26844-6_10

Wagenaar, G., Overbeek, S., Lucassen, G., Brinkkemper, S., & Schneider, K. (2017). Influence of Software Product Management Maturity on Usage of Artefacts in Agile Software Development. In *Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbrück, Austria, 30 November - 2 December, 2017* (pp. 19–27). Springer, Cham. http://doi.org/10.1007/978-3-319-69926-4_2

West, D. (2011). *Water-scrum-fall is the reality of agile for most organizations today. Forrester Research*.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-29044-2

Xu, L., & Brinkkemper, S. (2007). Concepts of product software. *European Journal of Information Systems*, *16*(5), 531–541.

Yang, C., Liang, P., & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, *111*, 157–184. http://doi.org/10.1016/J.JSS.2015.09.028

Yanzer Cabral, A., Blois Ribeiro, M., Lemke, A. P., Silva, M. T., Cristal, M., & Franco, C. (2009). A case study of Knowledge Management usage in agile software projects. In J. Filipe & J. Cordeiro (Eds.), *Enterprise Information Systems - Proceedings of the 11th International Conference on Enterprise Information Systems, ICEIS 2009, Milan, Italy, May 6-10, 2009* (Vol. 24, pp. 627–638). Springer Berlin Heidelberg.

Yilmaz, M., O'Connor, R. V., & Clarke, P. (2012). A Systematic Approach to the Comparison of Roles in the Software Development Processes. In A. Mas, A. Mesquida, T. Rout, R. V. O'Connor, & A. Dorling (Eds.), *Software Process Improvement and Capability Determination - Proceedings of the 12th International Conference on Process Improvement and Capability dEtermination in Software, Systems Engineering and Service Management (SPICE 2012), Palma, Spain, 29-31 May,* (Vol. 290, pp. 198–209). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-30439-2_18

Yin, R. K. (2013). *Case Study Research: Design and Methods (Applied Social Research Methods Series - volume 5). Case study research design and methods* (5th ed., Vol. 34). Thousand Oaks, California, USA: Sage Publications, Inc.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353. http://doi.org/10.1016/S0019-9958(65)90241-X

Published Work

Wagenaar, G., Helms, R., Damian, D., & Brinkkemper, S. (2015). Artefacts in Agile Software Development. In P. Abrahamsson, L. Corral, M. Oivo, & B. Russo (Eds.), *Product-Focused Software Process Improvement - Proceedings of the 16th International Conference on Product-Focused Software Process Improvement (PROFES 2015), Bolzano, Italy, December 2-4, 2015* (Vol. LNCS 9459, pp. 133–148). Springer International Publishing. http://doi.org/10.1007/978-3-319-26844-6_10

Wagenaar, G., Overbeek, S., & Helms, R. (2016). Competencies outside Agile Teams' Borders: The Extended Scrum Team. In *Position Papers of the Federated Conference on Computer Science and Information Systems (FedSYS), 36th IEEE Software Engineering Workshop (SEW-36), Gdansk, Poland, 11 - 14 September 2016, 291 – 298*

Wagenaar, G., Overbeek, S., & Helms, R. (2017). Describing Criteria for Selecting a Scrum Tool Using the Technology Acceptance Model. In Nguyen N., Tojo S., Nguyen L., Trawiński B. (Eds.), *Proceedings of the 9th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2017), Kanazawa, Japan, 3 - 5 April 2017,* 811 – 821 (Springer, Cham)

Wagenaar, G., Overbeek, S., Lucassen, G., Brinkkemper, S., & Schneider, K. (2017). Influence of Software Product Management Maturity on Usage of Artefacts in Agile Software Development. In *Proceedings of the 18th International Conference on Product-Focused Software Process Improvement (PROFES 2017), Innsbrück, Austria, 30 November - 2 December, 2017* (pp. 19–27). Springer, Cham. http://doi.org/10.1007/978-3-319-69926-4_2

Wagenaar, G., Overbeek, S., Lucassen, G., Brinkkemper, S., & Schneider, K. (2018). Working software over comprehensive documentation – Rationales of agile teams for artefacts usage. *Journal of Software Engineering Research and Development 6.1 : 7*

Wagenaar, G., Overbeek, S., & Brinkkemper, S. (2018). Fuzzy Artefacts: Formality of Communication in Agile Teams. To be published in *Proceedings of the International Conference on the Quality of Information and Communications Technology (QUATIC 2018), Coimbra, Portugal, 4 – 7 September 2018*

Summary

Agile software development (ASD) with methods as XP, Scrum or Kanban, has become a de facto standard in software development over the last years. ASD methods share a preference for face-to-face communication rather than formal comprehensive documentation. However, old habits die hard and in an evolutionary transformation from waterfall to agile software development documentation artefacts do not go extinct like dinosaurs; they just fade away. As agile teams are independent in their internal processes they can decide on, whether or not, to use documentation artefacts. Their decisions thus constitute an elaboration of the Agile Manifesto's second statement: "Working software over comprehensive documentation". Therefore, the main research question in this PhD dissertation is:

What is the role of artefacts in communication in agile software development teams?

The hypothesis underlying this research is that documentation has proven its value in traditional software development and that agile teams will at least carry over some of this value to their current practices.

In first instance the use of artefacts in agile teams was investigated. Had models already been proposed for this and if so, which artefacts did they identify? We conducted a case study in three agile teams in which we interviewed team members about their use of artefacts. The study's conclusions demonstrated the basic validity of our hypothesis. The teams in our case study did use artefacts and we thus partially confirmed previous findings and complemented them with additional artefacts, both traditional and agile. We noticed in this case study also the tool support of the agile teams, despite the fact that the teams were co-

located. This led to some further research into the choice of agile teams in their tool support. In another case study with twelve student teams we investigated possible criteria agile teams use to choose computer-based support for their software development process as communication and collaboration tools are crucial in modern agile teams. One of the conclusions drawn from this research was that the number of criteria concerning the use of agile artefacts, such as a visualization board, backlogs, or a burndown chart, far outnumbered other criteria. In contradiction, this did not always lead to a choice for an agile specific tool.

Having established the blend of traditional and agile artefacts it is evident that agile teams do consider them both to be important. However, this does not shine light on reasons for using them. In a holistic view we investigated whether or not there is a relation between maturity of an agile team's software development process and its use of artefacts. Evidence was found for maturity to be negatively correlated with the non-agile/all artefacts ratio. In a follow-up study we explicitly investigated rationales for the use of artefacts. In fourteen agile teams interviews were held with prominent team members to discuss the use of artefacts and the motivation for using them. Agile teams stated five groups of rationales, among which one for agile artefacts and four for traditional artefacts, for instance to promote internal team communication.

Formal and informal communication in ASD are still often regarded as two end points on a measuring scale, where ASD nowadays includes both. The two resemble the distinction between explicit and tacit knowledge and where the distinction between those two is recognized not to be a black and white one, this raised the question whether or not agile teams also experience intermediate appearances between formal communication (artefacts) and

informal communication (face-to-face). We coined this appearance a 'fuzzy' artefact. This is an artefact which is not formally documented, but one that is still explicitly recognized by an agile team. The findings from our case study confirmed the existence of fuzzy artefacts and teams use them, for instance, in the requirements process, the elaboration of user stories and in taking Go/No-Go decisions.

In a final case study we studied how agile teams proceed when for some subjects internal communication, with the aid of artefacts, fails. Who do agile teams address when additional competencies are needed and how are they aware of them? We found that agile team members agree on the boundaries of their team and are indeed cross-functional as far as 'traditional' phases of software development, for instance, analysis/design, programming, and testing are concerned. Additional competencies are obtained either on an ad hoc basis or are found in external partners who structurally contribute to an agile team every sprint. They could be considered to be members of an extended agile team.

The overall answer to our research question thus is: Yes, artefacts play an important role in the communication within agile teams. This is no surprise as far as agile artefacts, artefacts that are inherent to an ASD, are involved, for instance a product or sprint backlog, or user stories. Agile teams use, in addition, non-agile artefacts, which are associated with traditional software development rather than ASD. However, they have sound reasons to do so. Two examples: (1) Team-internal communication benefits from functional and technical design artefacts, (2) Quality assurance leads to test-related artefacts. In general, agile teams are able to explain why they are using the artefacts they use. Artefacts do not replace face-to-face communication, but complement it. They also take different shapes as in fuzzy artefacts, artefacts that are not formally

documented, but still explicitly recognized by agile teams. Communication does not end within the agile team itself. Sometimes, whether on an ad hoc or on a structural basis, an agile team has to be extended with external partners.

Samenvatting

Agile SysteemOntwikkeling (ASO) met methoden als XP, Scrum of Kanban, is de laatste jaren een de facto standaard geworden in de ontwikkeling van software. ASO-methoden hebben alle een voorkeur voor persoonlijke communicatie boven formele uitgebreide documentatie. Oude gewoonten sterven echter langzaam uit en in een evolutionaire transformatie van waterval naar agile softwareontwikkeling zijn documentatie-artefacten niet in één klap verdwenen; ze vervagen. Agile teams geven zelf hun interne processen vorm; ze beslissen zelf of ze al dan niet documentatie-artefacten gebruiken. Deze beslissingen vormen als het ware een uitwerking van de tweede uitspraak van het Agile Manifesto: "*Werkende software boven uitgebreide documentatie*". Daarom is de belangrijkste onderzoeksvraag in dit proefschrift:

Wat is de rol van artefacten in communicatie in agile softwareontwikkelteams?

De hypothese die aan dit onderzoek ten grondslag ligt, is dat documentatie haar waarde bewezen heeft in traditionele softwareontwikkeling en dat agile teams deze waarde in elk geval gedeeltelijk zullen overdragen op hun huidige praktijken.

In eerste instantie werd het gebruik van artefacten in agile teams onderzocht. Zijn hier al modellen voor opgesteld en zo ja, welke artefacten identificeerden deze modellen dan? We voerden een case study uit in drie agile teams, waarin we teamleden interviewden over hun gebruik van artefacten. De conclusies van de studie toonden de validiteit van onze hypothese aan. De teams in onze case study gebruikten artefacten en daarmee konden we eerdere bevindingen gedeeltelijk bevestigen, maar ook aanvullen met nieuwe artefacten, zowel traditionele als agile. In deze case study zagen we ook het belang van ondersteunende tools

binnen de agile teams, ondanks het feit dat de teams zich binnen één ruimte (gebouw) bevonden.

Deze observatie leidde tot verder onderzoek naar de keuze van agile teams in hun ondersteuning door tools. In een nieuwe case study met twaalf studententeams onderzochten we criteria die agile teams gebruiken om computertools te kiezen voor hun agile systeemontwikkelingsproces. Bekend was reeds het cruciale belang van dergelijke communicatie- en samenwerkingstools. Eén van de conclusies van het onderzoek was dat criteria met betrekking tot het gebruik van agile artefacten, zoals een Scrum-bord, een backlog of een burndown-grafiek, vele malen belangrijker worden gevonden dan andere criteria. Dit leidde overigens niet altijd tot een keuze voor een agile specifieke tool.

Gezien de gebruikte mix van traditionele en agile artefacten, is het duidelijk dat agile teams beide belangrijk vinden. Dit geeft echter nog geen inzicht in de reden(en) om ze te gebruiken. Met in eerste instantie een holistische blik onderzochten we of er een relatie is tussen de volwassenheid van het softwareontwikkelingsproces van een agile team en het gebruik van artefacten. Er werd enig bewijs gevonden dat de volwassenheid negatief gecorreleerd is met de verhouding tussen niet-agile en alle artefacten. In een vervolgstudie hebben we expliciet beweegredenen voor het gebruik van artefacten onderzocht. In veertien agile teams werden interviews gehouden met prominente teamleden om het gebruik van artefacten en de motivatie om ze te gebruiken, te bespreken. Agile teams noemden vijf groepen van beweegredenen, waaronder één voor agile artefacten en vier voor traditionele artefacten, bijvoorbeeld om interne communicatie in het team te bevorderen.

Formele en informele communicatie in ASD worden nog vaak beschouwd als twee eindpunten op een meetschaal. De twee eindpunten lijken op het onderscheid tussen expliciete en impliciete kennis, waarbij onderkend wordt dat het onderscheid tussen beide geen zwart-wit onderscheid is. Dit wierp de vraag op of agile teams intermediaire verschijningsvormen ervaren tussen formele communicatie (artefacten) en informele communicatie (direct). Inderdaad werd vastgesteld dat deze intermediaire artefacten gebruikt werden; deze zijn benoemd als 'fuzzy' artefacten. Zo'n fuzzy artefact wordt niet formeel gedocumenteerd, maar agile teamleden erkennen wel expliciet het bestaan ervan. De bevindingen uit de case study bevestigden het bestaan; teams gebruiken ze bijvoorbeeld in het requirements engineering proces, de uitwerking van user stories en het nemen van Go / No-Go-beslissingen.

In een laatste case study is onderzocht hoe agile teams te werk gaan als de interne communicatie, met behulp van artefacten, niet werkt. Tot wie richten agile teams zich als aanvullende competenties nodig zijn en hoe weten ze die te vinden? Agile teamleden blijken het eens zijn over de grenzen van hun team – wie is onderdeel van het team en wie niet? - en teams zijn inderdaad multifunctioneel in de 'traditionele' fasen van softwareontwikkeling, zoals analyse/ontwerp, programmeren en testen. Bijkomende competenties worden verkregen op ad-hoc basis of worden gevonden in externe partners, maar dan wel partners die elke sprint weer structureel bijdragen aan het team. Ze kunnen beschouwd worden als lid van een uitgebreid agile team.

Het algemene antwoord op de onderzoeksvraag luidt dan ook: Ja, artefacten spelen, nog steeds, een belangrijke rol in de communicatie binnen agile teams. Dit is geen verrassing als het gaat om agile artefacten, artefacten die inherent zijn aan een ASO, bijvoorbeeld een product- of sprint-backlog of user stories.

Agile teams gebruiken daarbovenop volop non-agile artefacten, die eerder geassocieerd worden met traditionele softwareontwikkeling. Ze hebben daar echter goede redenen voor. Twee voorbeelden: (1) Team-interne communicatie profiteert van functionele en technische ontwerpartefacten, (2) Kwaliteitsborging leidt tot testgerelateerde artefacten. Over het algemeen kunnen agile teams goed uitleggen waarom ze de artefacten gebruiken, die ze gebruiken. Artefacten vervangen face-to-face communicatie niet, maar vullen het aan. Ze nemen ook verschillende vormen aan, zoals in fuzzy artefacten, artefacten die niet formeel zijn gedocumenteerd, maar nog steeds expliciet worden herkend door agile teams.

Curriculum Vitae

Gerard Wagenaar was born on March 29th, 1960, in Purmerend, the Netherlands. Between 1981 and 1986, he studied at the University of Amsterdam to obtain a Master's degree in Computer Science, with a specialization in Artificial Intelligence. His Master's thesis explored the feasibility of developing knowledge based systems for varying domains.

Having finalized his studies, he joined the Netherlands Organization for Applied Scientific Research (TNO). He was involved in software development in TNO's department of Industrial Safety, among others the development of an emergency response management systems for accidents at sea. He fulfilled various roles from analyst to developer, consultant, project manager.

In the early nineties he was stationed with the International Association for the promotion of cooperation with scientists from the New Independent States of the Former Soviet Union (INTAS) in Bruxelles to act as scientific coordinator for project which were executed with funds from the Association.

Returning to the Netherlands Gerard worked in several places within the software industry such as team leader software development in a graphical company, and test coordinator in an insurance company.

In 1998 Gerard started his career in education when he joined the former University of Applied Sciences West-Brabant (now Avans University of Applied Sciences). He became lecturer within the Business IT & Management department and later on within the Software Engineering department. Research interests and topics of teaching include software engineering in general with a focus on

software development lifecycle models, and business intelligence with topics such as machine learning and big data.

From 2012 onwards Gerard combined his position at Avans with a PhD research at Utrecht University. Following his research interests he investigated the role of artefacts, both agile and non-agile, in agile software development methods. During the PhD research he presented various papers at international conferences and also acted as reviewer.

SIKS Dissertation Series

2014		
01	Nicola Barile (UU)	Studies in Learning Monotone Models from Data
02	Fiona Tuliyano (RUN)	Combining System Dynamics with a Domain Modeling Method
03	Sergio Raul Duarte Torres (UT)	Information Retrieval for Children: Search Behavior and Solutions
04	Hanna Jochmann-Mannak (UT)	Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation
05	Jurriaan van Reijsen (UU)	Knowledge Perspectives on Advancing Dynamic Capability
06	Damian Tamburri (VU)	Supporting Networked Software Development
07	Arya Adriansyah (TUE)	Aligning Observed and Modeled Behavior
08	Samur Araujo (TUD)	Data Integration over Distributed and Heterogeneous Data Endpoints
09	Philip Jackson (UvT)	Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language
10	Ivan Salvador Razo Zapata (VU)	Service Value Networks
11	Janneke van der Zwaan (TUD)	An Empathic Virtual Buddy for Social Support
12	Willem van Willigen (VU)	Look Ma, No Hands: Aspects of Autonomous Vehicle Control
13	Arlette van Wissen (VU)	Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains
14	Yangyang Shi (TUD)	Language Models With Meta-information
15	Natalya Mogles (VU)	Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare
16	Krystyna Milian (VU)	Supporting trial recruitment and design by automatically interpreting eligibility criteria
17	Kathrin Dentler (VU)	Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability
18	Mattijs Ghijsen (UVA)	Methods and Models for the Design and Study of Dynamic Agent Organizations

19	Vinicius Ramos (TUE)	Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support
20	Mena Habib (UT)	Named Entity Extraction and Disambiguation for Informal Text: The Missing Link
21	Kassidy Clark (TUD)	Negotiation and Monitoring in Open Environments
22	Marieke Peeters (UU)	Personalized Educational Games - Developing agent-supported scenario-based training
23	Eleftherios Sidirourgos (UvA/ CWI)	Space Efficient Indexes for the Big Data Era
24	Davide Ceolin (VU)	Trusting Semi-structured Web Data
25	Martijn Lappenschaar (RUN)	New network models for the analysis of disease interaction
26	Tim Baarslag (TUD)	What to Bid and When to Stop
27	Rui Jorge Almeida (EUR)	Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty
28	Anna Chmielowiec (VU)	Decentralized k-Clique Matching
29	Jaap Kabbedijk (UU)	Variability in Multi-Tenant Enterprise Software
30	Peter de Cock (UvT)	Anticipating Criminal Behaviour
31	Leo van Moergestel (UU)	Agent Technology in Agile Multiparallel Manufacturing and Product Support
32	Naser Ayat (UvA)	On Entity Resolution in Probabilistic Data
33	Tesfa Tegegne (RUN)	Service Discovery in eHealth
34	Christina Manteli(VU)	The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems
35	Joost van Ooijen (UU)	Cognitive Agents in Virtual Worlds: A Middleware Design Approach
36	Joos Buijs (TUE)	Flexible Evolutionary Algorithms for Mining Structured Process Models
37	Maral Dadvar (UT)	Experts and Machines United Against Cyberbullying
38	Danny Plass-Oude Bos (UT)	Making brain-computer interfaces better: improving usability through post-processing
39	Jasmina Maric (UvT)	Web Communities, Immigration, and Social Capital
40	Walter Omona (RUN)	A Framework for Knowledge Management Using ICT in Higher Education
41	Frederic Hogenboom (EUR)	Automated Detection of Financial Events in News Text

42	Carsten Eijckhof (CWI/TUD)	Contextual Multidimensional Relevance Models
43	Kevin Vlaanderen (UU)	Supporting Process Improvement using Method Increments
44	Paulien Meesters (UvT)	Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden
45	Birgit Schmitz (OUN)	Mobile Games for Learning: A Pattern-Based Approach
46	Ke Tao (TUD)	Social Web Data Analytics: Relevance, Redundancy, Diversity
47	Shangsong Liang (UVA)	Fusion and Diversification in Information Retrieval
2015		
01	Niels Netten (UvA)	Machine Learning for Relevance of Information in Crisis Response
02	Faiza Bukhsh (UvT)	Smart auditing: Innovative Compliance Checking in Customs Controls
03	Twan van Laarhoven (RUN)	Machine learning for network data
04	Howard Spoelstra (OUN)	Collaborations in Open Learning Environments
05	Christoph Bösch(UT)	Cryptographically Enforced Search Pattern Hiding
06	Farideh Heidari (TUD)	Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes
07	Maria-Hendrike Peetz(UvA)	Time-Aware Online Reputation Analysis
08	Jie Jiang (TUD)	Organizational Compliance: An agent-based model for designing and evaluating organizational interactions
09	Randy Klaassen(UT)	HCI Perspectives on Behavior Change Support Systems
10	Henry Hermans (OUN)	OpenU: design of an integrated system to support lifelong learning
11	Yongming Luo(TUE)	Designing algorithms for big graph datasets: A study of computing bisimulation and joins
12	Julie M. Birkholz (VU)	Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks
13	Giuseppe Procaccianti(VU)	Energy-Efficient Software
14	Bart van Straalen (UT)	A cognitive approach to modeling bad news conversations

15	Klaas Andries de Graaf (VU)	Ontology-based Software Architecture Documentation
16	Changyun Wei (UT)	Cognitive Coordination for Cooperative Multi-Robot Teamwork
17	André van Cleeff (UT)	Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs
18	Holger Pirk (CWI)	Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories
19	Bernardo Tabuenca (OUN)	Ubiquitous Technology for Lifelong Learners
20	Loïs Vanhée (UU)	Using Culture and Values to Support Flexible Coordination
21	Sibren Fetter (OUN)	Using Peer-Support to Expand and Stabilize Online Learning
22	Zhemin Zhu(UT)	Co-occurrence Rate Networks
23	Luit Gazendam (VU)	Cataloguer Support in Cultural Heritage
24	Richard Berendsen (UVA)	Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation
25	Steven Woudenberg (UU)	Bayesian Tools for Early Disease Detection
26	Alexander Hogenboom (EUR)	Sentiment Analysis of Text Guided by Semantics and Structure
27	Sándor Hóman (CWI)	Updating compressed colomn-stores
28	Janet Bagorogoza(TiU)	Knowledge Management and High Performance: The Uganda Financial Institutions Model for HPO
29	Hendrik Baier (UM)	Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains
30	Kiavash Bahreini(OU)	Real-ti On the robustness of Power Grids me Multimodal Emotion Recognition in E-Learning
31	Yakup Koç (TUD)	On the robustness of Power Grids
32	Jerome Gard(UL)	Corporate Venture Management in SMEs
33	Frederik Schadd (TUD)	Ontology Mapping with Auxiliary Resources
34	Victor de Graaf(UT)	Gesocial Recommender Systems
35	Jungxao Xu (TUD)	Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction
2016		
01	Syed Saiden Abbas (RUN)	Recognition of Shapes by Humans and Machines

02	Michiel Christiaan Meulendijk (UU)	Optimizing medication reviews through decision support: prescribing a better pill to swallow
03	Maya Sappelli (RUN)	Knowledge Work in Context: User Centered Knowledge Worker Support
04	Laurens Rietveld (VU)	Pub Trusting Crowdsourced Information on Cultural Artefactslishing and Consuming Linked Data
05	Evgeny Sherkhonov (UVA)	Expanded Acyclic Queries: Containment and an Application in Expla A Link to the Past: Constructing Historical Social Networks from Unstructured Data ining Missing Answers
06	Michel Wilson (TUD)	Robust scheduling in an uncertain environment
07	Jeroen de Man (VU)	Measuring and modeling negative emotions for virtual training
08	Matje van de Camp (TiU)	A Link to the Past: Constructing Historical Social Networks from Unstructured Data
09	Archana Nottamkandath (VU)	Trusting Crowdsourced Information on Cultural Artefacts
10	George Karafotias (VUA)	Parameter Control for Evolutionary Algorithms
11	Anne Schuth (UVA)	Search Engines that Learn from Their Users
12	Max Knobbout (UU)	Logics for Modelling and Verifying Normative Multi-Agent Systems
13	Nana Baah Gyan (VU)	The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach
14	Ravi Khadka (UU)	Revisiting Legacy Software System Modernization
15	Steffen Michels (RUN)	Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments
16	Guangliang Li (UVA)	Socially Intelligent Autonomous Agents that Learn from Human Reward
17	Berend Weel (VU)	Towards Embodied Evolution of Robot Organisms
18	Albert Meroño Peñuela (VU)	Refining Statistical Data on the Web
19	Julia Efremova (Tu/e)	Mining Social Structures from Genealogical Data
20	Daan Odijk (UVA)	Context & Semantics in News & Web Search
21	Alejandro Moreno Célleri (UT)	From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground

22	Grace Lewis (VU)	Software Architecture Strategies for Cyber-Foraging Systems
23	Fei Cai (UVA)	Query Auto Completion in Information Retrieval
24	Brend Wanders (UT)	Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach
25	Julia Kiseleva (TU/e)	Using Contextual Information to Understand Searching and Browsing Behavior
26	Dilhan Thilakarathne (VU)	In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains
27	Wen Li (TUD)	Understanding Geo-spatial Information on Social Media
28	Mingxin Zhang (TUD)	Large-scale Agent-based Social Simulation - A study on epidemic prediction and control
29	Nicolas Höning (TUD)	Peak reduction in decentralised electricity systems - Markets and prices for flexible planning
30	Ruud Mattheij (UvT)	The Eyes Have It
31	Mohammad Khelghati (UT)	Deep web content monitoring
32	Eelco Vriezekolk (UT)	Assessing Telecommunication Service Availability Risks for Crisis Organisations
33	Peter Bloem (UVA)	Single Sample Statistics, exercises in learning from just one example
34	Dennis Schunselaar (TUE)	Configurable Process Trees: Elicitation, Analysis, and Enactment
35	Zhaochun Ren (UVA)	Monitoring Social Media: Summarization, Classification and Recommendation
36	Daphne Karreman (UT)	Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies
37	Giovanni Sileno (UvA)	Aligning Law and Action - a conceptual and computational inquiry
38	Andrea Minuto (UT)	Materials That Matter - Smart Materials meet Art & Interaction Design
39	Merijn Bruijnes (UT)	Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect
40	Christian Detweiler (TUD)	Accounting for Values in Design
41	Thomas King (TUD)	Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance

42	Spyros Martzoukos (UVA)	Combinatorial and Compositional Aspects of Bilingual Aligned Corpora
43	Saskia Koldijk (RUN)	Context-Aware Support for Stress Self-Management: From Theory to Practice
44	Thibault Sellam (UVA)	Automatic Assistants for Database Exploration
45	Bram van de Laar (UT)	Experiencing Brain-Computer Interface Control
46	Jorge Gallego Perez (UT)	Robots to Make you Happy
47	Christina Weber (UL)	Real-time foresight - Preparedness for dynamic innovation networks
48	Tanja Buttler (TUD)	Collecting Lessons Learned
49	Gleb Polevoy (TUD)	Participation and Interaction in Projects. A Game-Theoretic Analysis
50	Yan Wang (UVT)	The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains
2017		
01	Jan-Jaap Oerlemans (UL)	Investigating Cybercrime
02	Sjoerd Timmer (UU)	Designing and Understanding Forensic Bayesian Networks using Argumentation
03	Daniël Harold Telgen (UU)	Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines
04	Mrunal Gawade (CWI)	Multi-Core Parallelism in a Column-Store
05	Mahdieh Shadi (UVA)	Collaboration Behavior
06	Damir Vandic (EUR)	Intelligent Information Systems for Web Product Search
07	Roel Bertens (UU)	Insight in Information: from Abstract to Anomaly
08	Rob Konijn (VU)	Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery
09	Dong Nguyen (UT)	Text as Social and Cultural Data: A Computational Perspective on Variation in Text
10	Robby van Delden (UT)	(Steering) Interactive Play Behavior

11	Florian Kunneman (RUN)	Modelling patterns of time and emotion in Twitter #anticipointment
12	Sander Leemans (TUE)	Robust Process Mining with Guarantees
13	Gijs Huisman (UT)	Social Touch Technology - Extending the reach of social touch through haptic technology
14	Shoshannah Tekofsky (UvT)	You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior
15	Peter Berck (RUN)	Memory-Based Text Correction
16	Aleksandr Chuklin (UVA)	Understanding and Modeling Users of Modern Search Engines
17	Daniel Dimov (UL)	Crowdsourced Online Dispute Resolution
18	Ridho Reinanda (UVA)	Entity Associations for Search
19	Jeroen Vuurens (TUD)	Proximity of Terms, Texts and Semantic Vectors in Information Retrieval
20	Mohammadbashir Sedighi (TUD)	Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility
21	Jeroen Linssen (UT)	Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)
22	Sara Magliacane (VU)	Logics for causal inference under uncertainty
23	David Graus (UVA)	Entities of Interest - Discovery in Digital Traces
24	Chang Wang (TUD)	Use of Affordances for Efficient Robot Learning
25	Veruska Zamborlini (VU)	Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search
26	Merel Jung (UT)	Socially intelligent robots that understand and respond to human touch
27	Michiel Joosse (UT)	Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors
28	John Klein (VU)	Architecture Practices for Complex Contexts
29	Adel Alhuraibi (UVT)	From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT
30	Wilma Latuny (UVT)	The Power of Facial Expressions
31	Ben Ruijl (UL)	Advances in computational methods for QFT calculations

32	Thaer Samar (RUN)	Access to and Retrievability of Content in Web Archives
33	Brigit van Loggem (OU)	Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity
34	Maren Scheffel (OUN)	The Evaluation Framework for Learning Analytics
35	Martine de Vos (VU)	Interpreting natural science spreadsheets
36	Yuanhao Guo (UL)	Shape Analysis for Phenotype Characterisation from High-throughput Imaging
37	Alejandro Montes García (TUE)	WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy
38	Alex Kayal (TUD)	Normative Social Applications
39	Sara Ahmadi (RUN)	Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR
40	Altaf Hussain Abro (VUA)	Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support (For applications in human-aware support systems)
41	Adnan Manzoor (VUA)	Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle
42	Elena Sokolova (RUN)	Causal discovery from mixed and missing data with applications on ADHD datasets
43	Maaike de Boer (RUN)	Semantic Mapping in Video Retrieval
44	Garm Lucassen (UU)	Understanding User Stories - Computational Linguistics in Agile Requirements Engineering
45	Bas Testerink (UU)	Decentralized Runtime Norm Enforcement
46	Jan Schneider (OU)	Sensor-based Learning Support
47	Yie Yang (TUD)	Crowd Knowledge Creation Acceleration
48	Angel Suarez (OU)	Collaborative inquiry-based learning
2018		
01	Han van der Aa (VUA)	Comparing and Aligning Process Representations
02	Felix Mannhardt (TUE)	Multi-perspective Process Mining
03	Steven Bosems (UT)	Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction

04	Jordan Janeiro (TUD)	Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks
05	Hugo Huurdeman (UVA)	Supporting the Complex Dynamics of the Information Seeking Process
06	Dan Ionita (UT)	Model-Driven Information Security Risk Assessment of Socio-Technical Systems
07	Jieting Luo (UU)	A formal account of opportunism in multi-agent systems
08	Rick Smetsers (RUN)	Advances in Model Learning for Software Systems
09	Xu Xie (TUD)	Data Assimilation in Discrete Event Simulations
10	Julienka Mollee (VUA)	Moving forward: supporting physical activity behavior change through intelligent technology
11	Mahdi Sargolzaei (UVA)	Enabling Framework for Service-oriented Collaborative Networks
12	Xixi Lu (TUE)	Using behavioral context in process mining
13	Seyed Amin Tabatabaei (VUA)	Using behavioral context in process mining: Exploring the added value of computational models for increasing the use of renewable energy in the residential sector
14	Bart Joosten (UVT)	Detecting Social Signals with Spatiotemporal Gabor Filters
15	Naser Davarzani (UM)	Biomarker discovery in heart failure
16	Jaebok Kim (UT)	Automatic recognition of engagement and emotion in a group of children
17	Jianpeng Zhang (TUE)	On Graph Sample Clustering
18	Henriette Nakad (UL)	De Notaris en Private Rechtspraak
19	Minh Duc Pham (VUA)	Emergent relational schemas for RDF
20	Manxia Liu (RUN)	Time and Bayesian Networks
21	Aad Slootmaker (OUN)	EMERGO: a generic platform for authoring and playing scenario-based serious games
22	Eric Fernandes de Mello Araújo (VUA)	Contagious: Modeling the Spread of Behaviours, Perceptions and Emotions in Social Networks
23	Kim Schouten (EUR)	Semantics-driven Aspect-Based Sentiment Analysis
24	Jered Vroon (UT)	Responsive Social Positioning Behaviour for Semi-Autonomous Telepresence Robots

25	Riste Gligorov (VUA)	Serious Games in Audio-Visual Collections
26	Roelof de Vries (UT)	Theory-Based And Tailor-Made: Motivational Messages for Behavior Change Technology
27	Maikel Leemans (TUE)	Hierarchical Process Mining for Scalable Software Analysis
28	Christian Willemse (UT)	Social Touch Technologies: How they feel and how they make you feel
29	Yu Gu (UVT)	Emotion Recognition from Mandarin Speech
30	Wouter Beek (VU)	The "K" in "semantic web" stands for "knowledge": scaling semantics to the web
2019		
01	Rob van Eijk (UL)	Comparing and Aligning Process Representations
02	Emmanuelle Beauxis- Aussalet (CWI, UU)	Statistics and Visualizations for Assessing Class Size Uncertainty
03	Eduardo Gonzalez Lopez de Murillas (TUE)	Process Mining on Databases: Extracting Event Data from Real Life Data Sources
04	Ridho Rahmadi (RUN)	Finding stable causal structures from clinical data
05	Sebastiaan van Zelst (TUE)	Process Mining with Streaming Data
06	Chris Dijkshoorn (VU)	Nichesourcing for Improving Access to Linked Cultural Heritage Datasets
07	Soude Fazeli (TUD)	Recommender Systems in Social Learning Platforms
08	Frits de Nijs (TUD)	Resource-constrained Multi-agent Markov Decision Processes
09	Fahimeh Alizadeh Moghaddam (UVA)	Self-adaptation for energy efficiency in software systems
10	Qing Chuan Ye (EUR)	Multi-objective Optimization Methods for Allocation and Prediction
11	Yue Zhao (TUD)	Learning Analytics Technology to Understand Learner Behavioral Engagement in MOOCs
12	Jacqueline Heinerman (VU)	Better Together
13	Guanliang Chen (TUD)	MOOC Analytics: Learner Modeling and Content Generation
14	Daniel Davis (TUD)	Large-Scale Learning Analytics: Modeling Learner Behavior & Improving Learning Outcomes in Massive Open Online Courses

15	Erwin Walraven (TUD)	Planning under Uncertainty in Constrained and Partially Observable Environments
16	Guangming Li (TU/e)	Process Mining based on Object-Centric Behavioral Constraint (OCBC) Models
17	Ali Hürriyetoğlu (RUN)	Extracting actionable information from microtexts