Artefacts in Agile Software Development

Conference Paper · December 2015

DOI: 10.1007/978-3-319-26844-6_10

CITATIONS
10

READS
3,209

4 authors:

Gerard Wagenaar
Utrecht University
10 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Remko Helms
Open Universiteit Nederland
97 PUBLICATIONS **1,150** CITATIONS
SEE PROFILE

Daniela Damian
University of Victoria
120 PUBLICATIONS **4,184** CITATIONS
SEE PROFILE

Sjaak Brinkkemper
Utrecht University
432 PUBLICATIONS **8,479** CITATIONS
SEE PROFILE

Artefacts in Agile Software Development

Gerard Wagenaar[1], Remko Helms[2], Daniela Damian[3], Sjaak Brinkkemper[2]

[1] Avans University of Applied Science, Breda, the Netherlands
g.wagenaar@avans.nl
[2] Utrecht University, Utrecht, the Netherlands
{r.w.helms, s.brinkkemper}@uu.nl
[3] University of Victoria, Victoria, Canada
danielad@cs.uvic.ca

Abstract. 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.

Keywords: Agile software development · Artefacts · Case studies · Scrum

1 Introduction

The application of an agile software development method, for instance XP or Scrum, is common nowadays in delivering state-of-the-art software [6,23]. 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*" [4]. This emphasis on communication in software development does not come as a surprise. A field study on software design process for large systems [8] already indicated that developing large software systems must be treated, at least in part, as a learning, communication, and negotiation process.

Opposite, or complementary, to face-to-face 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*" [4], the use of artefacts is situational and a choice made by an agile team, dependent on the value it attaches to them. One agile method, DSDM [31], 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 [14]. And in co-located agile software

development projects the production of artefacts contributes to coordination mechanisms, such as synchronization and boundary spanning [33]. 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 [1,2,12]. 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 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.

2.1 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*" [11], "*An artefact is a deliverable that is produced, modified, or used by a sequence of tasks that have value to a role*" [19]. 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 [8], 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*" [3, *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 [32]. This result was partially confirmed in a case study on the impact of agile principles and practices on large-scale software development projects [17].

In a case study on knowledge management usage in agile software projects [35] 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*" [35, *p. 634*].

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

2.2 Artefact Models

As starting point for a Scrum artefact model we take the Scrum process framework, which distinguishes artefacts, people and events [26,27]. 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' (DoD) 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[1] [5] served as foundation for a Scrum artefact class diagram (SACD) [9]. 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.
- Architectural information, consisting of domain model and 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 [13]. This diagram was subsequently extended to a more generic model to abstract it from local, project-specific processes (Fig. 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 [16]. On the basis of experiences from practice the diagram was further refined [10].

[1] A Process-Deliverable Diagram describes both processes and deliverables in one diagram [34]; the interpretation of deliverable in a PDD is analogous to our artefact.

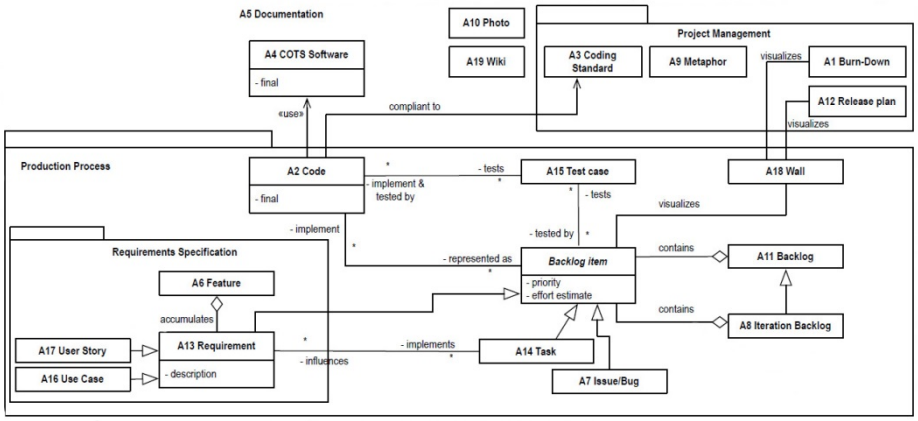

Fig. 1. Agile artefact class diagram (adapted from [16])

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 [24].

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’ (Fig. 1). A well-known distinction separates process from product artefacts [30]. 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 [16]. Other research also indicated the use of tools by agile teams, for instance story cards and the Wall [28,29]. In fact, using Scrum leads to an absolute need for tools [15]. 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.

2.4 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 (Fig. 2).

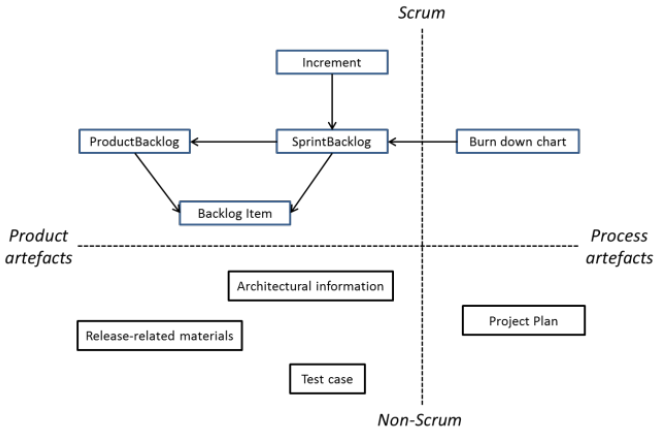

Fig. 2. Preliminary Scrum artefact model

In the upper half of Fig. 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 our preliminary 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 Fig. 2) are deliberately left out, because there, again, is a need for validation.

3 Case Study Design

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 (Fig. 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 [37]. For the case study we operationalized our goal into the following research questions:

RQ1 Which artefacts do Scrum teams produce and how are they related to each other?

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 [18]. According to the protocol organizations were required to use Scrum as software development method with a team of at least 5 members. We found 3 organizations willing to participate in our research, all having 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 [37]. The questionnaire contained 2 sections, apart from some questions on the interviewee's background. The first section contained questions related to the use of agile practices within the team and the second explicitly addressed (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 main 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. We used a combination of open and axial coding [21] on the basis of our preliminary Scrum artefact model, complemented with additional artefacts when necessary.

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

3.1 Validity

In terms of validity of our research method 4 criteria are widely used: construct validity, internal validity, external validity and reliability [37].

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 [37]. We applied all three: (1) each interviewee was pro-

vided with a summary for his or her approval and key informants commented on a draft case study report, (2) various team members were involved to complement viewpoints, and (3) there is a direct link from interviews (and other material) to conclusions by the use of the qualitative data analysis tool Nvivo[2].

Internal validity is mainly a concern for explanatory case studies [24,37]. Our case studies are exploratory, but we did apply pattern matching, one of the analytical techniques to enhance internal validity, by consistently coding material on the basis of our preliminary Scrum artefact model.

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 [37]. 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 [37] were both applied in our study to increase reliability.

4 Results

The results from the case study will be described next. We will first give an impression of (the structure of) each organization. We will then list the results pertaining to RQ1, divided into core Scrum artefacts and non-Scrum artefacts. At the same time, to answer RQ2, we will, for each artefact, indicate its usage as product or process artefact and/or, if applicable, the use of a supporting tool.

4.1 Organizations

We start with a summary describing some key characteristics of the organizations / teams (Table 1). Most numbers are indicative; the size of the Scrum team is not.

Table 1. Key characteristics case study organizations / Scrum teams

Organization	Domain	Size		Scrum experience	Installed base	Number of users
		Company	Scrum team			
Controller	Management of objects	120	5	2 years	1500	n/a[3]
Sunflower	Floral industry	15	6	2½ year	125	750
Local	Government taxing	5.000	10	1½ year	50	1000

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

[3] Controller has no direct insight in the number of users.

The organizations all deliver product software [35], where Controller and Local fabricate several software packages and Sunflower only one. The size of an organization relates to an organization as a whole. All other numbers refer to one software product, and its associated Scrum team. The installed base refers to the number of software packages in use with different customers (companies). Since one installation of a package may have any number of users, this number is significantly higher than the installed base.

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 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.

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[4], in use with Controller) or developed within the organization itself. All tools are also used to register progress information, from project planning to actual status.

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[5]) and Local (PVCS[6]) track information on versions and associated changes.

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

[4] A description can be found at: www.collab.net.
[5] A description can be found at: www.git-scm.com.
[6] A description can be found at: www.serena.com.

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.

Controller. Controller uses (exhaustive list):

- ***Test scripts*** which are drawn up by the testers of the Development Team in a so-called GWT-structure: **G**iven defines the initial situation, **W**hen defines actions, **T**hen predicts the final situation. Test scripts are typically product artefacts and are related to backlog items as well as to the source code.
- 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, accompanying the release.
- 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; it relates to both sprint backlog and release.

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; it relates to both sprint backlog and release. Both are product artefacts.

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. The design relates to backlog items on the one side and to source code on the other side.
- 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.

5 Discussion

In the previous section we have presented all artefacts in use with the 3 case study organizations. We will now turn to a discussion of the results. In first instance we will integrate the results of individual organizations to constitute our extended Scrum artefact model. We will then specifically discuss the use of non-Scrum artefacts and explicitly relate them to previous models. We end our discussion with a paragraph on the use of supporting tools.

5.1 Artefact Model

We first summarize the artefacts found in the case studies (Table 2).

Table 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 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 leftmost column in this way also introduces a vocabulary for the artefacts encountered in the case study organizations, since their naming differs (slightly) from one organization to another.

The table, as a synthesis of results of the study, now contains building blocks to redraw the preliminary model as an extended Scrum artefact model (Fig. 3).

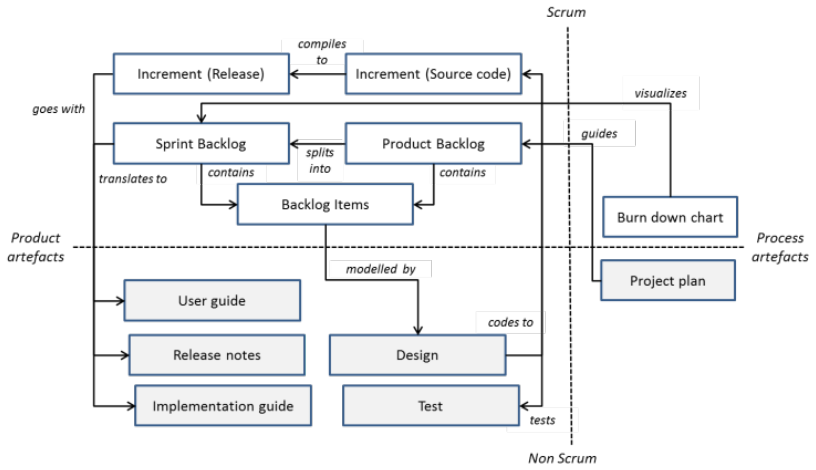

Fig. 3. Extended Scrum artefact model

In comparison with our preliminary model (Fig. 2) we note that all Scrum artefacts were already included. For the non-Scrum artefacts Project plan and Test were already identified, Design was not and Release-related material has been refined.

Both our preliminary and our extended artefact model are 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, for instance, in the AACD Backlog Item is a generalization of (all of) Requirement, Task and Issue/Bug, we restricted ourselves to Backlog Item.

5.2 Non-Scrum Artefacts: Design, Test and Release-related material

The extended Scrum model contains artefacts from both the Scrum artefact and the agile artefact class diagram. Both individual class diagrams do not cover the extended Scrum artefact model, but the class diagrams together make a fair prediction of most artefacts. However, 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 [22]. 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 [7]. 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 [17,32]. 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 [20].

5.3 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 [15]. 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, but the tools support Scrum rather than non-Scrum artefacts. We conclude that the co-located teams use integrative tools, perhaps as intensive as distributed teams.

6 Conclusions

Our paper presents a clear and empirical overview of the 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.

6.1 Limitations

Limitations are inherent when a case study is selected as research method. We studied three teams using Scrum. We have taken measures to reinforce validity, and therefor feel that some generalization of results is justified, but the sample remains small.

We limited ourselves in our study to artefacts which were within the sphere of influence of the teams themselves. We did not explicitly investigate or verify whether artefacts followed general organizational standards, derived from, for instance, ISO/IEC 15504.

6.2 Future work

Of course we would like to see our results confirmed with other organizations. To this extend we will continue working on our artefact model. We also feel that further research concerning the reasons for deciding, whether or not, to produce certain artefacts is motivated. To this extent the influence of context variables (for instance, organization or team size, Scrum maturity) on such decisions constitutes an interesting research topic.

We did not find extensive internal documentation, such as architectures, UML design or module descriptions. What still occupies us is the use of artefacts in knowledge management in the longer term; it would be interesting to consider artefacts in this broader perspective.

Acknowledgements. The results from this research would not have been possible without the generous cooperation from the three case study organizations. The (first) author also wants to express his gratitude to Avans University of Applied Sciences for facilitating and supporting this research.

References

1. Abrahamsson, P., Oza, N., & Siponen, M. (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). Springer.
2. 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, Portland (OR), USA, 3-5 May, 2003 (pp. 244–254). IEEE.
3. Ambler, S. (2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. John Wiley & Sons.
4. Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., et al. (2001). Agile Manifesto. Retrieved from http://agilemanifesto.org/
5. Blijleven, V. (2012). Scrum Development Process from a Method Engineering Perspective. Retrieved from http://foswiki.cs.uu.nl/foswiki/MethodEngineering/Scrumdevelopmentprocess20112012
6. 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). Springer.
7. 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, Linz, Austria, 9-11 June, 2003 (pp. 302–307). IEEE.
8. 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.
9. Dijkstra, O. (2013). Extending the Agile Development Discipline to Deployment - The Need For a Holistic Approach. Retrieved from http://dspace.library.uu.nl/handle/1874/279416.
10. 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, Sjanghai, China, 18-21 August, 2014 (pp. 1–5). IEEE.
11. Georgiadou, E. (2003). Software Process and Product Improvement: A Historical Perspective. Cybernetics and Systems Analysis, 39(1), 125–142.
12. 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, 21–25 July, 2013. System Dynamics Society.
13. Gröber, M. (2013). Investigation of the Usage of Artifacts in Agile Methods. Retrieved from http://www4.in.tum.de/~kuhrmann/studworks/mg-thesis.pdf.
14. Herbsleb, J. D., & Moitra, D. (2001). Global software development. IEEE Software, 18(2), 16–20.
15. 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, Limerick, Ireland, 13-16 July, 2009 (pp. 175–184). IEEE.
16. 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, Bari, Italy, 26-29 August, 2013 (pp. 11–20). IEEE.
17. 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. Retrieved from http://liu.diva-portal.org/smash/record.jsf?pid=diva2:608712.
18. Maimbo, H. (2005). Designing a Case Study Protocol for Application in IS research. In Proceedings of the 9th Asia Conference on Information Systems, Bangkok, Thailand, 7-10 July, 2005 (pp. 1281–1292). PACIS.
19. 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, Oslo, Norway, 3-8 October, 2010 (pp. 183–197). Springer.
20. Nonaka, I. (1994). A Dynamic Theory of Organizational Knowledge Creation. Organization Science, 5(1), 14–37.

21. 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.
22. Parnas, D. L., & Madey, J. (1995). Functional documents for computer systems. Science of Computer Programming, 25(1), 41–61.
23. 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, Lund, Sweden, 17-22 September, 2012 (pp. 139–148). ACM.
24. Royce, W. (1970). Managing the development of large software systems. Proceedings of IEEE WESCON (pp. 1–9).
25. Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131–164.
26. Schwaber, K., & Beedle, M. (2002). Agile Software Development with Scrum (1st ed.). Prentice Hall.
27. Schwaber, K., & Sutherland, J. (2013). The Scrum guide – The Definitive Guide to Scrum: The Rules of the Game. Retrieved from http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-US.
28. 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.
29. 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.
30. Sommerville, I. (2001). Software Documentation (revised version). Retrieved from https://ifs.host.cs.st-andrews.ac.uk/Research/Publications/Papers-PDF/2000-04/Documentation.pdf.
31. Stapleton, J. (1997). DSDM, Dynamic Systems Development Method: The Method in Practice. Addison Wesley Publishing Company.
32. 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, Pisa, Italy 3-5 October, 2011 (pp. 159–166). ACM.
33. Strode, D. E., Huff, S. L., Hope, B. G., & Link, S. (2012). Coordination in co-located agile software development projects. *Journal of Systems and Software*, *85*(6), 1222–1238.
34. Weerd, I. van de, & Brinkkemper, S. (2008). Meta-Modeling for Situational Analysis and Design Methods. In M.R. Syed & S.N. Syed (eds.), Handbook of Research on Modern Systems Analysis and Design Technologies and Applications (pp. 35–54). IGI Global.
35. Xu, L., & Brinkkemper, S. (2007). Concepts of product software. European Journal of Information Systems, 16(5), 531–541.
36. 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, Milan, Italy, 6-10 May, 2009 (pp. 627–638). Springer.
37. Yin, R. K. (2009). Case Study Research: Design and Methods (4th ed.). Sage Publications, Inc.