Understanding the Role of Requirements Artifacts in Kanban

Liskin, Olga

ACM
2014

Liskin , O , Schneider , K , Fagerholm , F & Münch , J 2014 , Understanding the Role of Requirements Artifacts in Kanban . in Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2014) . ACM , pp. 56-63 , International workshop on cooperative and human aspects of software engineering , Hyderabad , India , 02/06/2014 . https://doi.org/10.1145/2593702.2593707

http://hdl.handle.net/10138/153356
https://doi.org/10.1145/2593702.2593707

submittedVersion

Understanding the Role of Requirements Artifacts in Kanban

Olga Liskin, Kurt Schneider
Leibniz Universität Hannover
Welfengarten 1
30167 Hannover, Germany
+49 511 76219667
{olga.liskin,kurt.schneider}@inf.uni-hannover.de

Fabian Fagerholm, Jürgen Münch
Department of Computer Science, University of Helsinki
P.O. Box 68
FI-00014 University of Helsinki, Finland
+358 9 19151383
fabian.fagerholm@helsinki.fi,
juergen.muench@cs.helsinki.fi

ABSTRACT

User stories are a well-established way to record requirements in agile projects. They can be used as such to guide the daily work of developers or be split further into tasks, which usually represent more technical requirements. User stories and tasks guide communication and collaboration in software projects. However, there are several challenges with writing and using user stories in practice that are not well documented yet. Learning about these challenges could raise awareness for potential problems. Understanding how requirements artifacts are used for daily work could lead to better guidelines on writing stories that support daily work tasks. Moreover, user stories may not be appropriate to capture all kinds of requirements that are relevant for a project.

We explore how to utilize requirements artifacts effectively, what their benefits and challenges are, and how their scope granularity affects their utility. For this, we studied a software project carried out in the Software Factory at the Department of Computer Science, University of Helsinki. We investigated the requirements artifacts and then interviewed the developers and the customer about their experiences. Story and task cards have helped the participants throughout the project. However, despite having a Kanban board and rich communication within the team, some requirements were still too implicit, which also led to misunderstandings. This and other challenges revealed by the study can guide future in-depth research.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications – *methodologies.*

General Terms

Management, Documentation, Experimentation, Human Factors.

Keywords

Requirements Artifacts, User Stories, Kanban, Collaboration.

1. INTRODUCTION

Good requirements communication is crucial to the success of agile projects. Especially in agile projects, a big emphasis is laid on collaborating with the customer in order to implement his or her requirements as well as possible. For this purpose, agile methods put user stories into the center of the process [2],[4]. While there is agreement about the usefulness of the user story oriented approaches, there is little focused research on the exact benefits and challenges. This results in lacking guidelines for how to use user stories [19] and tasks and jeopardizes the stories' and tasks' effectiveness in projects. One aspect that we believe influences the handling of a requirements artifact is the amount of functionality it deals with at a time, described as *granularity* in this paper.

Granularity of a requirements artifact has many facets. It can be understood in terms of: (See [14])

- *Clarity/vagueness*. If a user story leaves out a lot of information, it is written vaguely.
- *Concreteness/abstraction*. A user story can describe the desired functionality as an abstract concept or sketch a concrete manifestation of this concept.
- *Scope*. A user story that implies a lot of system functionality (and accordingly, implementation work) has a large scope.

In order to change the scope of a user story, the desired functionality must be changed. For example, in order to reduce implementation work, some of the desired functionality must be removed or the story must be split into smaller ones. In contrast, the clarity or abstractness of a user story is varied by providing different information about the desired functionality that the customer has in mind, while the functionality remains the same

While all three aspects are important for the quality of a user story, we focus on scope granularity. If a user story's scope is too large, the team working with it might become less agile. If a story covers a large chunk of requirements, the developers implementing it get fewer chances to get customer feedback, check their progress, and adjust their plans. Also, the desired functionality might become vague and less tangible for the developers resulting in more unexpected issues. However, when stories are smaller, there are usually more of them.

We try to find out how developers and customers in agile projects work with requirements artifacts, which demands they have, and how the scope or the number of requirements artifacts affects their work. In order to approach these questions, we investigate a software project at the Software Factory at the Department of Computer Science, University of Helsinki. We observe the requirements artifacts and take a detailed look at the participants' experiences and challenges throughout the project. These help us approach our research questions and uncover interesting points for discussion. For example, we find that user stories that take one week or longer often are too vague and should be split into smaller stories.

The remainder of this paper is structured as follows. Section 2 describes literature that is related to our work. Section 3 presents the context of the study, namely the observed Software Factory project. In Section 4, we introduce the design and leading questions of our study. Section 5 presents the study results, which we then discuss in Section 6. Section 7 concludes the paper.

2. RELATED WORK

In an empirical study, Cao and Ramesh [3] investigate general requirements engineering practices in agile projects. Abdullah et al. [1] specifically address communication patterns in agile requirements engineering.

Sharp et al. investigate communication within agile teams [22] and in particular the role of physical artifacts like story cards and the Wall [23]. They observe real teams and use the distributed cognition approach in order to understand communication in agile teams. Petre et al. [21] compare the use of physical artifacts between agile and traditional products. While the other authors deal with team communication aspects, we specifically focus on requirements communication. Therefore, our work also addresses aspects of collaboration with the customer.

The effective handling of user stories has been in the focus of recent literature. Cohn [4] and Nawrocki [17] provide general guidelines on how user stories should be written. Wake has created the acronyms INVEST and SMART to manifest criteria for good user stories and tasks [25].

Kanban does not incorporate sprints and therefore estimation of user stories is optional. However, a user story's scope still influences the workflow so that estimation and splitting of user stories are relevant techniques that help control a story's scope. Further, the method used in the observed project is a prominent variant of Kanban called Scrumban [12] (see Section 3 for a more detailed explanation). Scrumban in turn does include sprints and thus relies on story and task estimations. Cohn [4], [5] and Leffingwell [13] provide insights on estimation and splitting of user stories. Gottesdiener [9] considers the INVEST criteria and advises on how to split user stories according to them. Miranda et al. [16] focus on improving estimation strategies and the estimations themselves. Fægri [7] investigates how estimation can be established as a team activity and observes barriers to team estimation in a specialist organizational environment. He reports how poor planning and too optimistic estimates affected higher work pressure. Haugen, Mahniç et al. and Tamrakar et al. [10] [15] [24] examine whether introduction of planning poker improves the team's ability to estimate user stories.

Oza et al. [18] investigate the impacts of Kanban on Communication and Collaboration. In a questionnaire study, they investigate frequency and importance of communication and collaboration and discuss patterns. Ikonen et al. [11] present an empirical investigation of the impact of Kanban on project work and also look into the visualization and communication aspects. They present a framework for understanding project work, which consists of nine literature-based aspects. They find out that visualization and communication were supported by Kanban and had positive effects on the project.

Petersen and Wohlin [20] apply cumulative flow diagrams in lean projects to visualize the flow and define new measures related to these. They evaluate the measures in an industrial case study.

3. STUDY CONTEXT

The study was carried out in the Software Factory laboratory at the Department of Computer Science, University of Helsinki. The Software Factory consists of a physical facility and a framework procedure for selecting and conducting projects in which teams of senior computer science students collaborate with industry partners on software development projects that have real business relevance [6]. In this study, the partner company was Tellybean, Inc., a startup based in Helsinki, Finland. Tellybean produces a novel video calling service targeted for a late-adopter market. Users of the service are assumed to have a low level of technical proficiency, and thus there is a heavy emphasis on simplicity and ease of use in the service. The service is delivered as a hardware-software product that connects to the user's existing TV set.

The student team initially consisted of seven master's-level students, of which two exited the project during the first few weeks due to scheduling conflicts with other courses. The remaining five-person team worked for a total of seven weeks in close collaboration with Tellybean's representatives, mainly the lead architect and designer of the technical platform. The project started with an initial meeting during which the overall service vision was presented and the team worked closely with the customer to form an initial understanding of the goals of the project. During the remainder of the project, the team worked in a self-organised manner, with regular customer meetings, daily team meetings, and direct contact to the customer representative as needed. The team was supported by three coaches, who helped the team with project management, customer communication, and quality assurance. However, the coaches' role was supportive and they were instructed to let the team solve their own problems rather than providing solutions.

The team used a variant of the Kanban task scheduling system, a method which originates in Lean and Agile software development [11]. The variant used in this project is called *Scrumban*, and is a combination of Scrum project management practices and a Kanban board [12]. Practices from the Scrum methodology included daily team meetings and week-long sprints with an end demo for the customer and a retrospective session where the team reflected on their progress in the previous sprint. The Kanban board consists of columns, which represent task states, into which task cards are placed to signify the stage in which each task is currently. Markers representing team members are placed on the task cards to indicate who is working on which task. The objective is to keep an even flow of tasks and to gain a visual overview of the current state of the project.

4. STUDY DESIGN

In this study, we used direct observation, surveying and interviews to gain information on the requirements artifacts that were used in the project described in the previous section. Qualitative analysis was used to gain insights into the role of the artifacts. In this section, we present the detailed research questions and our procedures for data collection and analysis.

4.1 Research Questions

Our objective was to examine the role of requirements artifacts in Kanban by studying the customer's and developers' perspectives in a real project. We formed the following research questions in order to focus the study. They guide us in our interviews.

RQ1: How are requirements artifacts (stories and tasks) used to facilitate team communication and collaboration?

Story and task cards contain information about the requirements and present them in a visible way on the board. We want to find out, how the participants used these requirements artifacts for communication and collaboration. Also, we wanted to see, for which communication purposes the cards were used (user requirements, work items, etc.).

RQ2: Does the granularity of requirements artifacts affect communication and collaboration?

Granularity (in the meaning of scope) greatly influences the work that is attached to a story or task card. Changing the overall granularity of requirements artifacts therefore could influence the whole workflow within a Kanban project. Story and task cards can be on a high level and contain a large amount of requirements and functionality. Alternatively, they can be divided into smaller items – with the consequence that each item might become more tangible, but at the same time there are more of them. We want to find out, whether differences can also manifest themselves in the communication and collaboration.

Further, we sought to link effects on collaboration to concrete granularity values. As a weaker form, we wanted to at least isolate granularity ranges that proved to be beneficial or problematic.

RQ3: How are user stories, tasks, and actual implementation work mapped to each other?

Splitting a user story into smaller stories or tasks is an often-used but not strongly documented technique. Especially if granularity of user stories and tasks influences development, knowing how to split an artifact well is crucial.

We want to understand how artifacts in this project are split and which effects this has. Requirements can be split vertically or horizontally. User stories can be split into smaller stories or into tasks. Actual implementation work could consist of further subtasks that are implicit. All these are interesting things we want to understand.

4.2 Data Gathering and Analysis

We recorded all user story and task cards that were written by the students. Each task belonged to a user story. After each sprint, the students filled out a questionnaire where they entered each story card they had worked on and the time that they worked on it. Further, the students stated whether the duration had corresponded with what they had implicitly expected or whether the implementation of the task took longer or shorter time.

At the end of the project, we interviewed all participants. We used semi-structured interviews. Based on our research questions, we formed leading topics for the interviews. We conducted 5 interviews in total, one with the customer (C) and 4 with the developers (D1 – D5). Two of the developers (D3 and D4) had to be interviewed together because of their personal schedules. On average, an interview took 30 minutes.

We recorded all interviews, transcribed them and coded all relevant statements. Then, we grouped all coded statements based on the interview topics, i.e. all statements about a certain topic were grouped together.

5. RESULTS

In this section, we describe our findings from the interviews and from the collected story and task cards. First, we describe the process and notable events. Then, in the following sections, we address the research questions.

Figure 1 summarizes the relevant themes in our results and their connections.

5.1 The Actual Process

5.1.1 Process and Planning

Figure 2 summarizes the process, requirements artifacts, and events throughout the project. The height of the requirements artifacts on the vertical axis depicts their duration within the sprint. The striped artifacts were planned for but not completed within Sprint 1. Each sprint started on a Wednesday and took one week. Between two sprints, the developers and the customer had a meeting ("customer meeting"). The developers presented the work of the previous sprint and then talked about the next week. The customer told the team what was the next functionality he wanted and then they talked about the technical details of what had to be done for this.

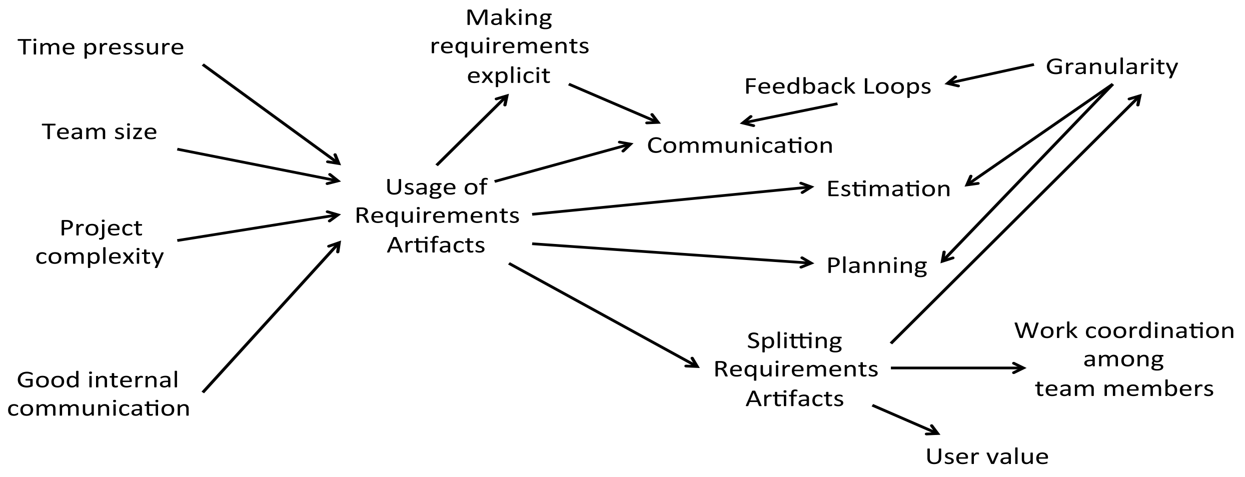
Time pressure
Team size
Project complexity
Good internal communication
Making requirements explicit
Usage of Requirements Artifacts
Communication
Feedback Loops
Granularity
Estimation
Planning
Splitting Requirements Artifacts
Work coordination among team members
User value

Figure 1: Themes that emerged in the interviews

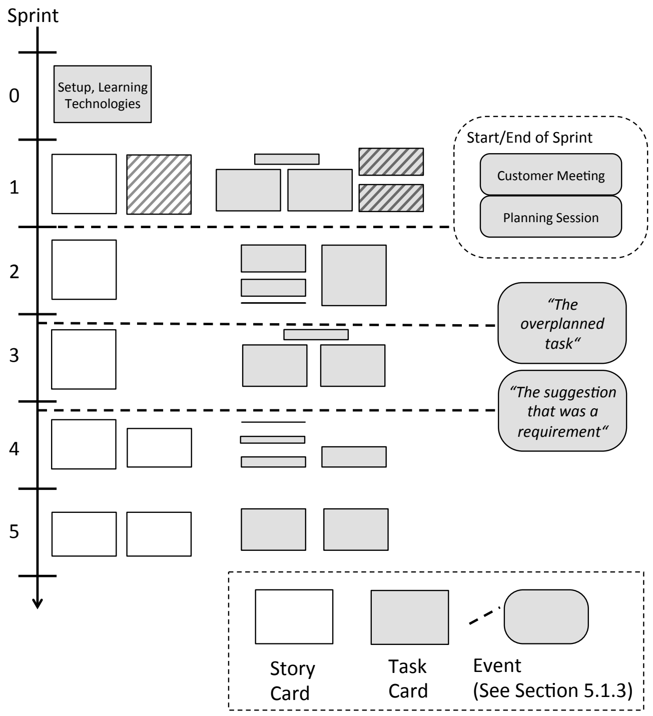

Figure 2: Project overview with sprints, requirements artifacts and notable events. The height of artifacts depicts their duration in the sprint.

After the customer had left, the team did a planning session. There, they wrote the actual story and task cards for the sprint. Usually, a sprint contained one story card, which illustrated the goal for the sprint. Sometimes, the team had talked with the customer on the task level and wrote a story card that summarized all the tasks mentioned by the customer.

Mostly, a story was divided into 2-3 tasks, which often mainly facilitated work distribution among team members. They divided the story into the client-side and server-side implementations, and, where applicable, the definition of the protocol. The user stories and task cards for the current sprint were placed on a physical Kanban board in the development space.

There was no explicit backlog. Instead, the members had rather a fluid backlog in their heads. There existed a slide from the first meeting with the customer, which contained a vision together with all essential and dispensable features.

5.1.2 Creation of a prototype based on only a vision

In the first sprint, the customer was not available. In the sprint planning he was substituted by another person from the customer's company, who was in charge of the overall user experience of the service. This person presented only the vision of the whole product. This information alone was not enough for the developers to clearly understand what needed to be done. Therefore, the first sprint went into the wrong direction. The outcome was a prototype, from which the developers had learned some relevant things, which however were of no use for the customer.

5.1.3 Two miscommunications

Two important *issues* emerged in the interviews. They had played a major role within the project.

"*The overplanned task"*: One of the planned and (half-) implemented subtasks was not required for the sprint. Two developers (D2 and D5) had interpreted the customer's requirements for a sprint in the way that they saw three desired sub-functions, although the customer actually only had wanted two of them. The third task was not completely superfluous for the product, but had such a low priority that the customer did not want it in one of the current or next sprints. One developer (D1) knew that. However, he and D3 had not been at the sprint planning. Instead they only received their tasks and started working on these. On the last day of the sprint, after D2 and D5 had worked on the wrong subtask for two or three days, they realized that they would not be able to make it and brought this subtask up in a team conversation. Only then, D1 could clarify that this subtask was not in the sprint's focus.

This issue shows three things. For that story, there were communication issues with the customer, communication issues inside the team, and a planning issue, which only allowed the developers to see on the last day of the sprint that they would not be able to implement everything they had intended to do.

"The suggestion that was a requirement": The customer wanted a user requirement (authentication) to be solved with the help of a specific library (passport.js). He mentioned it in the customer meeting. The developers, however, understood it as a suggestion and solved the user requirements with their own – in their eyes simpler – method. Only in the next customer meeting the customer saw that the desired library had not been used. This had to be fixed in the next sprint.

This event shows a communication issue with the customer.

5.2 How are requirements artifacts (stories and tasks) used to facilitate team communication and collaboration? (RQ1)

5.2.1 Value of user stories and tasks

Both stories and tasks had a value for the developers. The user stories were seen as helpful to understand what the actual goals of a sprint were. Further, it gave the developers a good feeling to see what benefit for the user they had created at the end of a sprint. ("*In a way, it was good, because at least we get some direction where to go.*", D1)

Tasks divided a user story into smaller pieces. Often, a story was divided into a client-side and a server-side task, which were then implemented by a developer pair each. The tasks were further divided into subtasks. Sometimes, this happened explicitly by writing the subtasks on the task card, often the splitting happened only implicitly when a developer planned how to solve the task. The splitting was considered as important by the developers. It is what in the first place made the big sprint goal tangible to them and lead their daily work. ("*It's more, like, usable. Because, if you have once sentence – [the customer] wants to see active clients or something. Everybody is like, "ok", but these [tasks] are the actual things to do. And a story is more like the topic of the week.*", D1) ("*Cut stories to small tasks. Like... that's making it easier to implement and also easier to focus, I think. [...] And also you feel like you are doing something. You see the results.*", D2) ("*[...]it divides a big part into smaller ones, so we can focus on smaller ones. Also, it kind of explained what we need to do for this task, step by step.*", D5)

Although the developers saw a value in the stories and cards, they tried to minimize the amount of items within a sprint. One reason for this was that they felt highly pressed for time and did not want to waste much by maintaining too detailed a current project status on the board. Another reason was that they felt that they did not need very thorough planning in a project this small.

5.2.2 Kanban board vs. implicit communication

In this team, there was much communication. The developers were sitting in the same room almost all the time, paired up on most tasks, and had lunch together, where they often communicated about the current status of their respective tasks. Because of the small team and project size and because of the rich communication, the members did not see a great need of the Kanban board for their communication. ("*So, we constantly were talking, so there [was] no information point [in the board] - that "ok, now I'm doing this and this" - to write 5 more [cards] daily. It was more like informing the coaches.*", D1)

The developers saw value in the board for making things visible. Since they themselves did not depend on this additional visibility, they saw the board's greatest use in informing the coaches (as more external persons) about what was going on in the project. One developer, who was responsible for quality assurance, also stated that he found the overall view provided by the board helpful for deciding which quality assurance tasks were most appropriate for the according sprint.

One member found that his actual work was completely detached from the task cards on the board. Later, he remarked that he was working on the client side mostly on his own and did not feel the need to coordinate with others. ("*We would talk to the customer also on a technical level and then we would make the tasks up by ourselves. So, we would be living on this low abstraction level, but then, when doing [sprint planning], we would have to think at a high level, just for the sake of how this works. This was unnecessary and frustrating to me, in a way*", D3)

Interacting with the customer also happened mainly through communicating with him in the weekly customer meeting. They discussed the topic for the next sprint in great detail, because the customer was also experienced with the technical aspects of the desired product. Since the developers wrote the actual story and task cards after the customer was gone, he did not see the actual cards before the next meeting (at the end of the sprint).

5.2.3 Additional value from explicit information

Although all team members found that their internal communication was sufficient, they still had communication issues, namely the two issues explained in Section 5.1.3. These consisted of misunderstandings the respective parties were not aware of. However, in both cases, there existed a representation of one of the parties in the tasks. For *the overplanned task*, a task card stated all three subtasks the developers intended to implement. If the customer or the other two team members had seen this task, they might have noticed the misunderstanding. Equally, for *the suggestion that was a requirement*, there was a task card that explained how the developers were going to satisfy the user story. If the customer would have seen this task, he might have noticed that the developers were going to implement the story without using the desired library.

We asked all participants in the interviews, whether they thought that showing the tasks to the customers might have helped with the misunderstandings. They all believed that showing some form of externalized tasks (definitions of done, task cards, meeting minutes) would have helped. However, this is only speculation and possibly not the only solution to this problem.

5.3 Does the granularity of requirements artifacts affect communication and collaboration? (RQ2)

In the interviews, the participants did not directly recount shortcomings they experienced

The participants were previously not familiar with the concept of granularity, especially in the sense in which we mean it. They stated that in general they could imagine working with smaller stories and tasks, but they did not see the need for it due to the small project size. However, we saw some effects that we bring in connection with the story granularity.

5.3.1 Length of Feedback Loops

The developers asked for customer feedback after a user story has been implemented. Even after they decided to communicate more often in the course of sprints, the developers asked the customer about feedback after they had implemented a user story. Therefore, in this setup, the size of a user story directly influences the length of feedback loops.

5.3.2 Planning Precision

The developers did not estimate their story and task cards in detail. However, by planning a story or task into a sprint, they implicitly estimated that it would get finished within the sprint. In particular, the sum of all tasks should take one week to be implemented by four or five developers.

In general, the developers found it difficult to predict how long a task will take. ("*The estimations of the tasks are really hard, so I don't know whether it takes one day or three days. We had like an overall vision that "that's doable in one week.*"", D1)

In one of the first sprints, two developers had underestimated one of their tasks. They had expected the task to take approximately one week. Only on the last day of the sprint, were the developers able to predict that they would not implement the task on time.

After this sprint, the team planned more defensively. They only took things into the sprint, when they were sure that they would finish it in the sprint. The effect in the following sprints was that the team always finished early with the tasks.

5.3.3 Problematic granularity ranges

Most stories and tasks took one week. Since the developers had little experience with smaller items, we are not able to relate effects to a specific size. The developers did not see the granularity of their stories or tasks as problematic. They stated that in a team like theirs, where there is a lot of communication, it was ok to have relatively vague stories and tasks.

However, our observations showed that with stories and tasks of this size, some negative effects occurred. First, we saw communication issues. When a weeklong story was discussed as a whole, it was not always clear, what exactly was included in the story and what might be excluded (see Section 5.1.3). Further, we saw planning issues. For a task that covers so much functionality at a time, the developers were only able to predict whether it would fit into a certain time window when it was already very late in the sprint.

Further, we identified issues that arise when stories or tasks are made smaller. Since there will be more of the artifacts in turn, the cost to maintain all artifacts rises. Especially, moving cards in-between linked development tasks can disturb implementation. ("*If you are in a hectic working mode, "ok, now I need to stop and write some [cards] [for the next] 10 minutes".*" D1) Also, planning and estimating all small tasks takes a lot longer. ("*To make a good plan, it's quite time-consuming. And the sprint planning was right after the customer meeting, which is like at three o'clock in the afternoon. And most of the team members are ready to go home. So, they just don't have the mind to do it well.*", D5) ("*I think, we only had two sprint plannings that were quite thorough and clear. and actually, for both plannings it took two hours, which is quite long for us.*", D5)

Splitting a user story into smaller pieces also has an additional effect: if a user story gets too small, it can lose in value for the customer. ("*No. because, this [task a)] doesn't bring anything to the end user. Maria [- the persona -] doesn't see that. So, that was the story [...] that the end user has something to experience or a benefit that makes her life easier. So, that [task a)] doesn't actually do anything; you can't see that.*", C)

5.4 How are user stories, tasks, and actual implementation work mapped to each other? (RQ3)

5.4.1 Horizontal Splitting of User Stories

The developers mainly split user stories into tasks, as compared to splitting them into smaller stories. The main purpose was to coordinate work among team members. Most sprints contained one story and the developers divided it into work packages that each pair could focus on. The team members stated that splitting a story into tasks helped them to make it more tangible, what needs to be done. (See Section 5.2.1.)

The tasks split a story in a horizontal way, i.e. a task consisted of multiple user-oriented parts, but covered only one technical tier. The most prominent division was between the client and server side. Most of the tasks took almost the full week.

We asked the developers if they could imagine splitting their stories vertically, i.e. one item (story or task) contains one user-oriented piece and implements it on all technical levels. They thought that this would be possible but did not see additional value in this. They thought so mainly because their project and the team were so small and they had such a good understanding of what needed to be done.

5.4.2 Early Interfaces

To facilitate coordination of their work in the middle of a sprint, the developers used to define a shared interface as early as possible in the sprint. This happened via the definition of the protocol between client and server.

This allowed the developers to integrate and test parts of implemented functionality often in the sprint, although their tasks actually took almost the whole sprint. With this strategy, coordination did not occur in relation to a finished task but at independent coordination points in the course of a task.

5.4.3 Actual Implementation Work as Implicit Subtasks

For dividing work among pairs, the developers wrote actual task cards and used them on the Kanban board. The task cards explicitly reflected this division of responsibility. Division into subtasks, however, was either not reflected by task cards at all or only by writing the subtasks on the respective task card. ("*I think, for some weeks, I had like only one ticket there and that's what I do. But of course, that's only for the Kanban, because the real thing is not that I do only one thing in a week.*", D1)

The developers preferred to have a clean Kanban board with as few cards as possible. One reason for this was the constant hurry the team felt. They felt like using much time to reflect on all small happenings would waste too much development time. ("*It was more like "ok, we have these four [cards] on the Kanban board, if I know what to do, I would take extra time for me to split them into smaller [cards] and then always update it daily.*" D1) ("*but our mindset was "ok, let's not waste time on this because we will lose half a day to do [planning] properly. Because there was like a constant hurry, to actually complete the stuff.*", D1)

Another reason was because the team was small and co-located. The team members felt like they were aware of what was happening in the team, so that they did not need extra task cards to reflect that. ("*So, maybe we were not so focused on the board to check. ... I have done one project earlier, where we had nine people on the team. Then the board is necessary, I think, to know, what's the [status], what others are doing.*", D2)

In general, the developers saw a value in making small tasks visible. For example, in other projects they already had used tasks to represent what a team member is implementing on a particular day. ("*Ok, this was almost like real-life work - but in the real-life work situation, I think, trello would be better, or maybe duplicated somehow. Because lots of people work like - ok, we were always in class, but normal people work remotely and so on, so it would be nice to know in details what the team members are doing.*", D1) ("*Yes. Even for a small team like us, there were some disagreements about the requirements. So, for a larger team, there is a larger probability [of miscommunication] to happen.*", D5)

6. DISCUSSION

We have seen many interesting events in this study, which we would like to discuss.

Requirements artifacts that take one week or more to implement are too vague.

The investigated project mainly dealt with requirements artifacts that took one week to be implemented by one or two pairs. In two of five sprints, the team experienced collaboration issues that we also attribute to the granularity of the requirements artifacts. The size of the artifacts affected misunderstandings as well as planning precision.

In the investigated case, the team was small, the project was comprehensible, and communication was good. However, even under such good circumstances, communication and collaboration problems occurred. In bigger or more detached teams, the probability of such problems is even higher.

In addition, we have previously discovered that such a relatively big size of requirements artifacts is not untypical in user story-based projects. In an earlier survey [14] with about 50 practitioners, about a half of our participants stated that 30% or more of their user stories take one week or more to implement. They also suggest that feedback and tangibility are more problematic for larger user stories.

Therefore, we recommend splitting requirements artifacts whose granularity is in the range of one week or more.

Requirements must be made explicit

Despite good communication in the team, misunderstandings happened in the course of the project. Misunderstandings are hard to spot just with regular communication, because often, the participants are not aware that the other party has a different understanding (“Symmetry of Ignorance”, [8]). They might not even mention the part that contains the misunderstanding.

Therefore, it is important to communicate about requirements on as concrete a level as possible. In the studied project, there existed a textual representation of the conflicting part in both miscommunication cases. However, it was only seen and considered by some of the participants. All participants believed that if the customer would have seen some textual representation of the developers’ interpretations, the miscommunications could have been discovered earlier.

The participants suggested different forms of textual representations. In addition to the tasks themselves, definitions of done or meeting minutes with outlined decisions were suggested. However, it needs to be noted that in long textual representations like meeting minutes, relevant information might be difficult to find.

Requirements artifacts in agile development are exactly the key for this activity. When openly displayed to everybody, they are a means to effectively and efficiently represent what will be implemented and make requirements visible.

This underlines the indispensability of requirements artifacts in agile projects. Further, integrating the artifacts in a meeting where everybody is present (like customer meetings) helps to make sure that everybody takes note of the artifacts. Mechanisms for ensuring that artifacts have been seen and understood could include signing off each artifact separately, and for more critical cases, testing that the artifact has been understood by posing questions that will reveal differences in interpretation.

Collaboration challenges in Kanban

We have seen that it is not trivial to collaborate in Kanban. Effective collaboration requires a lot of discipline from all participants. Although the process and the practices already support communication to a great extent, it can still happen quickly that not all practices are followed and collaboration suffers.

Our results show that the customer has to be involved in the Kanban process. If the team uses Kanban only internally and, for example, does not include requirements artifacts in customer meetings, the collaboration might suffer from misunderstandings. The customer might feel that she does not get what she wants; the developers might not know what the customer wants; customer and developers might not understand each other.

Specifically, a good process that prevents miscommunication about requirements could incorporate the following items. The customer should “own” the backlog together with the team, and the customer should own the “*Next*” column on the Kanban board. When both, team and customer, have signed off on a task, the customer is permitted to put it into the *Next* column. Then, the team members can pull tasks from the *Next* column into subsequent columns. The customer then needs to be involved again in the final acceptance of the task.

These are all ideas on how Kanban (and Scrumban) should work in theory. However, our study shows how difficult it is for the team and customer to understand how this interaction is supposed to work and how difficult it is for them not to bypass the system and start to collaborate in an ad-hoc manner.

The second challenge emerged in the planning process. In Kanban, the backlog, planning, and estimations are optional. Since the focus is on the flow of tasks, it is not important to predict how long a task will take. When using a timeboxed method like Scrumban, however, they become crucial. We have seen that too simplistic sprint planning can result in incorrect estimations of what will be delivered within one iteration. If a task is underestimated and then not delivered, this might result in a lack of trust. The other way around, when the amount of doable work is underestimated, the sprint needs to be filled up with additional tasks. This might slow down development. Especially, if the customer does not have control over what gets added to the sprint, she might feel that having to wait for the end of a sprint is a burden.

6.1 Limitations

We have investigated the concrete events and experiences in one specific project setting. Our results are not generalizable to all other agile projects. The time period of the studied project is relatively short. The team was still in the phase of learning and adapting to the process. Also, the team members were new to this process. Their experience might not be comparable to that of software developers who have experience in Kanban. Therefore, they might have encountered problems that do not apply to experienced teams.

Further, the research approach was mainly qualitative, which reflects subjective opinions of the participants. These cannot be generalized, but must be seen as indicators of possible challenges.

Despite all limitations, we believe that our results have a value for many projects. They are suggestions of possible problems and solutions. They demonstrate that even in small projects, communication and collaboration issues can occur. The results show that just good internal communication is not enough for meeting collaboration challenges.

We also think that many other teams can easily get in a situation where they face the same challenges. For example, many teams might drift to rather simplistic planning practices due to time pressure. Also, teams might be using Kanban only internally while having a different process to the outside, which affects customer collaboration and requirements specification. Further, newly formed teams always need to establish good communication practices first. Equally, teams that have only recently moved to Kanban need a certain time to learn.

7. CONCLUSIONS

The presented study provides insights into the use of requirements artifacts in a Kanban project. It shows that requirements artifacts played an important role in the project. Although the participants felt that their communication was good, the project still suffered from misunderstandings. Especially when the requirements artifacts were not made visible to all participants, misunderstandings occurred. In addition, the study points to collaboration challenges that arise when artifacts are too coarsely grained in scope size.

In the future, we would like to conduct more in-depth studies in the area of requirements artifacts. We have seen many possible challenges and would like to examine these more thoroughly. Further studies should evaluate, whether our suggested solutions – like explicitly signing off story cards, splitting story and task cards and establishing more sophisticated planning – really have an effect on collaboration in software teams.

8. ACKNOWLEDGMENTS

We would like to thank the project participants for their help by providing insights in the interviews.

9. REFERENCES

[1] Nik Nailah Binti Abdullah, Shinichi Honiden, Helen Sharp, Bashar Nuseibeh, and David Notkin. Communication patterns of agile requirements engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering*, page 1. ACM, 2011.

[2] Kent Beck and Cynthia Andres. *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[3] Lan Cao and Balasubramaniam Ramesh. Agile requirements engineering practices: An empirical study. *IEEE Software*, 1:60–67, 2008.

[4] Mike Cohn. *User Stories Applied: For Agile Software Development*. Prentice Hall, 2004.

[5] Mike Cohn. *Agile estimating and planning*. Pearson Education, 2005.

[6] F. Fagerholm, N. Oza, , and J. Münch. A platform for teaching applied distributed software development: The ongoing journey of the helsinki software factory. In *Proceedings of the 3rd International Workshop on Collaborative Teaching of Globally Distributed Software Development*, 2013.

[7] Fægri, T. E. Adoption of team estimation in a specialist organizational environment. In A. Sillitti, A. Martin, X. Wang, & E. Whitworth (Eds.), *Agile Processes in Software Engineering and Extreme Programming (11th International Conference, XP2010),* Vol. LNBIP 48: 28-42. Springer Berlin Heidelberg, 2010.

[8] G. Fischer. Social Creativity, Symmetry of Ignorance and Meta-Design. *Knowledge-Based Systems Journal*, 2000.

[9] Ellen Gottesdiener, Mary Gorman. *Slicing Requirements for Agile Success.* Better Software, 2010. (http://ebgconsulting.com/Pubs/Articles/SlicingRequirementsForAgileSuccess_Gottesdiener-Gorman_August2010.pdf)

[10] N.C. Haugen. An empirical study of using planning poker for user story estimation. In *Agile Conference, 2006*, pages 9–34, 2006.

[11] Marko Ikonen, Elena Pirinen, Fabian Fagerholm, Petri Kettunen, and Pekka Abrahamsson. On the impact of kanban on software project work: An empirical case study investigation. In *Engineering of Complex Computer Systems (ICECCS), 2011 16th IEEE International Conference on*, pages 305–314. IEEE, 2011.

[12] C. Ladas. *Scrumban: Essays on Kanban Systems for Lean software development*. Scrumban: Essays on Kanban Systems for Lean software development, 2009.

[13] Dean Leffingwell. *Agile software requirements: lean requirements practices for teams, programs, and the enterprise*. Addison-Wesley Professional, 2010.

[14] Olga Liskin, Raphael Pham, Stephan Kiesling, and Kurt Schneider. Why we need a granularity concept for user stories. In *Agile Processes in Software Engineering and Extreme Programming (15th International Conference, XP2014),* in press. Springer Berlin Heidelberg, 2014.

[15] Viljan Mahnic and Tomaž Hovelja. On using planning poker for estimating user stories. *Journal of Systems and Software*, 85(9):2086–2095, 2012.

[16] Eduardo Miranda, Pierre Bourque, and Alain Abran. Sizing user stories using paired comparisons. *Information and Software Technology*, 51(9):1327 – 1337, 2009.

[17] Jerzy Nawrocki, Michal Jasiñski, Bartosz Walter, and Adam Wojciechowski. Extreme programming modified: embrace requirements engineering practices. In *Proceedings of 2002 IEEE Joint Conference of Requirements Engineering (RE)*, pages 303–310. IEEE, 2002.

[18] Nilay Oza, Fabian Fagerholm, and Jürgen Münch. How does kanban impact communication and collaboration in software engineering teams? In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, 2013.

[19] Chetankumar Patel and Muthu Ramachandran. Story card based agile software development. *International Journal of Hybrid Information Technology*, 2(2):125–140, 2009.

[20] Kai Petersen and Claes Wohlin. Measuring the flow in lean software development. *Software: Practice and experience*, 41(9):975–996, 2011.

[21] Marian Petre, Helen Sharp, and Sallyann Freudenberg. The mystery of the writing that isn't on the wall: Differences in public representations in traditional and agile software development. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on*, pages 120–122. IEEE, 2012.

[22] Hugh Robinson and Helen Sharp. Collaboration, communication and co-ordination in agile software development practice. In *Collaborative Software Engineering*, pages 93–108. Springer, 2010.

[23] Helen Sharp, Hugh Robinson, and Marian Petre. The role of physical artefacts in agile software development: Two complementary perspectives. *Interacting with Computers*, 21(1-2):108–116, 2009.

[24] Ritesh Tamrakar and Magne Jørgensen. Does the use of fibonacci numbers in planning poker affect effort estimates? In *Proceedings of 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012)*, pages 228–232, IET Conference Proceedings, 2012.

[25] William C. Wake. *INVEST in Good Stories, and SMART Tasks*. XP123, 2003. (http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/)