Information and Software Technology 52 (2010) 480–491

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A teamwork model for understanding an agile team: A case study of a Scrum project

Nils Brede Moe *, Torgeir Dingsøyr, Tore Dybå

SINTEF, NO-7465 Trondheim, Norway

ARTICLE INFO

Article history:
Received 30 July 2009
Received in revised form 26 October 2009
Accepted 12 November 2009
Available online 20 November 2009

Keywords:
Agile software development
Scrum
Software engineering
Teamwork
Empirical software engineering
Case study

ABSTRACT

Context: Software development depends significantly on team performance, as does any process that involves human interaction.
Objective: Most current development methods argue that teams should self-manage. Our objective is thus to provide a better understanding of the nature of self-managing agile teams, and the teamwork challenges that arise when introducing such teams.
Method: We conducted extensive fieldwork for 9 months in a software development company that introduced Scrum. We focused on the human sensemaking, on how mechanisms of teamwork were understood by the people involved.
Results: We describe a project through Dickinson and McIntyre's teamwork model, focusing on the interrelations between essential teamwork components. Problems with team orientation, team leadership and coordination in addition to highly specialized skills and corresponding division of work were important barriers for achieving team effectiveness.
Conclusion: Transitioning from individual work to self-managing teams requires a reorientation not only by developers but also by management. This transition takes time and resources, but should not be neglected. In addition to Dickinson and McIntyre's teamwork components, we found trust and shared mental models to be of fundamental importance.

1. Introduction

Software development depends significantly on team performance, as does any process that involves human interaction. A common definition of a team is "a small number of people with complementary skills who are committed to a common purpose, set of performance goals, and approach for which they hold themselves mutually accountable" [22].

The traditional perspective on software development is rooted in the rationalistic paradigm, which promotes a plan-driven product-line approach to software development using a standardized, controllable, and predictable software engineering process [15]. Today, this traditional mechanistic worldview is challenged by the agile perspective that accords primacy to uniqueness, ambiguity, complexity, and change, as opposed to prediction, verifiability, and control. The goal of optimization is being replaced by those of flexibility and responsiveness [33].

Setting up a work team is usually motivated by benefits such as increased productivity, innovation, and employee satisfaction. Research on software development teams has found that team performance is linked with the effectiveness of teamwork coordination [19,25]. In the traditional plan-driven approach, work is coordinated in a hierarchy that involves a command-and-control style of management in which there is a clear separation of roles [33,34]. In the agile approach, work is coordinated by the self-managing team, in which the team itself decides how work is coordinated [8].

A team that follows a plan-driven model often consists of independently focused self-managing professionals, and a transition to self-managing teams is one of the biggest challenges when introducing agile (change-driven) development [33]. Neither culture nor mind-sets of people can be changed easily, which makes the move to agile methodologies all the more formidable for many organizations [8]. In addition, it is not sufficient to put individuals together in a group, tag them "self-managing", and expect that they will automatically know how to coordinate and work effectively as an agile team.

Our objective is to provide a better understanding of the nature of self-managing agile teams, which can in turn benefit the effective application of agile methods in software development. To this end, we conducted a longitudinal study that draws on the general literature of teamwork and self-managing teams. Such a study can provide valuable insights for understanding the challenge of introducing the self-managing agile team. We sought to answer the following research question:

> How can we explain the teamwork challenges that arise when introducing a self-managing agile team?

* Corresponding author. Tel.: +47 93028687; fax: +47 73592977.
E-mail addresses: nilsm@sintef.no (N.B. Moe), torgeird@sintef.no (T. Dingsøyr), tored@sintef.no (T. Dybå).

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.11.004

The remainder of this paper is organized as follows: Section 2 gives an overview of the literature on teamwork and agile software development. Section 3 describes our research question and method in detail. Section 4 presents results from a nine-month fieldwork of teamwork in a Scrum team. Section 5 contains a discussion of the findings. Section 6 concludes and provides suggestions for further work.

2. Background: teamwork and agile software development

In this section, we give a short introduction to the field of teamwork, teamwork in agile development, and the teamwork model that is used as the basis for our work.

2.1. Teamwork

The topic of teamwork has attracted research from several disciplines [10,17,41]. The concept of teamwork carries with it a set of values that encourage listening and responding constructively to views expressed by others, giving others the benefit of the doubt, providing support, and recognizing the interests and achievements of others [22]. Such values are important because they promote individual performance, which boosts team performance, and they help teams to perform well as a group, and good team performance boosts the performance of the organization.

Research on teamwork includes the development of tests to identify personality characteristics, because it has often been argued that good teams need a certain blend of personalities. Examples are the Belbin test [7] and the Myers–Briggs Type indicator. There is also a great deal of research on climate at work group and team level. The most studied model of team climate is that of [48] who suggests that four climate factors (vision, participative safety, task orientation, and support for innovation) are essential for team innovation to occur.

Furthermore, there are studies of teams over time, which indicate that teams go through set phases. The most well-known of these studies are those of Tuckman [46], who identified the phases as forming, storming, norming, and performing. Other studies have focused on the relationships between team members and argue that group cohesiveness is important for team success (cited in [41]). However, the use of teams does not always result in success for the organization [17]. Team performance is complex, and the actual performance of a team depends not only on the competence of the team itself in managing and executing its work, but also on the organizational context provided by management.

Much research has been devoted to what is described as self-managing, autonomous, or empowered teams [17,23,26,45,47]. One of the reasons that the use of self-managing teams has become popular is that some research suggests that their use promotes more satisfied employees, lower turnover, and lower absenteeism [10]. Others also claim that self-managing teams are a prerequisite for the success of innovative projects [20,44].

Although the majority of studies report that using self-managing teams has positive effects, some studies offer a more mixed assessment; such teams can be difficult to implement, and they risk failure when used in inappropriate situations or without sufficient leadership and support [18]. In addition, research on team performance indicates that the effects of autonomous work groups are highly situational dependent and that the effects of autonomous work-group practices depend on such factors as the nature of the workforce and the nature of the organization [10,17]. Further, autonomy on the individual level may conflict with autonomy on the group level. When a team as a whole is given a great deal of autonomy, it does not follow that the individual team members are given high levels of individual autonomy. Barker [5], for example, pointed out that self-managing groups may end up controlling group members more rigidly than they do under traditional management styles, while Markham and Markham [29] suggested that it may be difficult to incorporate both individual autonomy and group autonomy in the same work group. For Individuals to be motivated and satisfied with their jobs they need to have control over their own work and over the scheduling and implementation of their own tasks [1,26].

2.2. Teamwork in agile development: the Scrum team

In a software team, the members are jointly responsible for the end product and must develop shared mental models by negotiating shared understandings about both the teamwork and the task [28]. Project goals, system requirements, project plans, project risks, individual responsibilities, and project status must be visible and understood by all parties involved [21].

Most current development methods have it as a premise that software teams should self-organize or self-manage [36,42]. Scrum, which is a project-management-oriented agile development method, was inspired by a range of fields, such as complexity theory, system dynamics, and Nonaka and Takeuchi's theory of knowledge creation [35], and has adapted aspects of these fields to a setting of software development. Self-management is a defining characteristic in Scrum. Compared with traditional command-and-control oriented management, Scrum represents a radically new approach for planning and managing software projects, because it brings decision-making authority to the level of operational problems and uncertainties.

Rising and Janoff [36] describe Scrum as a development process for small teams, which includes a series of short development phases or iterations ("sprints"). A Scrum team is given significant authority and responsibility for many aspects of their work, such as planning, scheduling, assigning tasks to members, and making decisions: "The team is accorded full authority to do whatever it decides is necessary to achieve the goal" [43].

However, despite the popularity of the method, a systematic review of empirical studies of agile development [16] found only one case study of Scrum in the research literature prior to 2006.

2.3. Dickinson and McIntyre's teamwork model

The issue of what processes and components comprise teamwork and how teamwork contributes to team effectiveness and team performance has been much studied [9,19,26,30,39], but there is no consensus concerning its conceptual structure [38]. Salas et al. [40] identify 136 different models in their literature review and present a representative sample of 11 models and frameworks.

Using recent research and previous reviews, Dickinson and McIntyre [13] identified and defined seven core components of teamwork. Using these components and their relationships as a basis, they proposed the teamwork model that is used in this work. The model consists of a learning loop of the following basic teamwork components: communication, team orientation, team leadership, monitoring, feedback, backup, and coordination (Fig. 1).

We selected the Dickinson and McIntyre teamwork model for the following reasons:

1. It includes the most common elements that are considered in most research on teamwork processes [38,39]. In addition, it considers important elements that are required in self-managed teams: team orientation, functional redundancy and backup behavior [32,35], communication, feedback and learning [33], and shared leadership [22]. Further, the model covers important

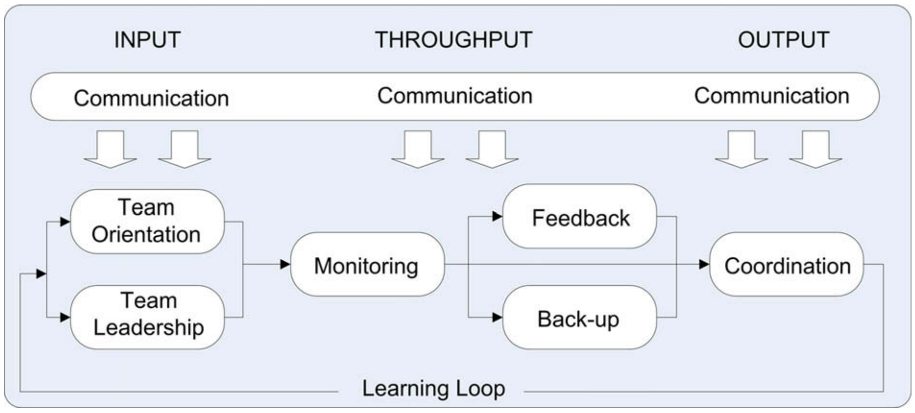

Fig. 1. The Dickinson and McIntyre teamwork model [13].

elements that are found in software teams, such as coordination of work [25].

2. It specifies what teamwork skills should be observed, in that the model is presented with a conceptual framework for developing measures of teamwork performance that can ensure effective individual and team performance [13], pp. 22.
3. It considers the teamwork process as a learning loop in which teams are characterized as adaptable and dynamically changing over time. Continuous self-management requires a capacity for double-loop learning that allows operating norms and rules to change along with transformation in the wider environment [32].

Each component of the model is explained in Table 1. According to Dickinson and McIntyre, team leadership and team orientation are 'input' components of teamwork because at least one of these attitudes is required for an individual to participate in a team task. Team leadership can be shown by several team members that is also a prerequisite for a team's being self-managing. In such teams, team members should share the authority to make decisions, rather than having: (a) a centralized decision structure in which one person (e.g. the team leader) makes all the decisions or (b) a decentralized decision structure in which all team members make decisions regarding their work individually and independently of other team members [20]. So, while the traditional perspective of a single leader suggests that the leadership function is a specialized role that cannot be shared without jeopardizing group effectiveness, when leadership is shared, group effectiveness is achieved by empowering the members of the team to share the tasks and responsibilities of leadership [22].

In the Dickinson and McIntyre model, the components of monitoring, feedback, and backup are the intermediate processes for ensuring effective teamwork. Finally, the 'output' component is coordination because it defines the performance of the team. Communication is a transversal component of particular importance, because it links the other components. To build software effectively, there is a need for tight coordination among the various efforts involved so that the work is completed and fits together [25].

Table 1
The Dickinson and McIntyre teamwork model: definitions of teamwork components.

Team orientation: Refers to the team tasks and the attitudes that team members have towards one another. It reflects an acceptance of team norms, the level of group cohesiveness, and the importance of team membership, e.g.
• assigning high priority to team goals
• participating willingly in all relevant aspects of the team
Team leadership: Involves providing direction, structure, and support for other team members. It does not necessarily refer to a single individual with formal authority over others. Team leadership can be shown by several team members, e.g.
• explaining to other team members exactly what is needed from them during an assignment
• listening to the concerns of other team members
Monitoring: Refers to observing the activities and performance of other team members and recognizing when a team member performs correctly. It implies that team members are individually competent and that they may subsequently provide feedback and backup, e.g.
• being aware of other team members' performance
• recognizing when a team member performs correctly
Feedback: Involves the giving, seeking, and receiving of information among team members. Giving feedback refers to providing information regarding other members' performance. Seeking feedback refers to requesting input or guidance regarding performance and to accepting positive and negative information regarding performance, e.g.
• responding to other members' requests for information about their performance
• accepting time-saving suggestions offered by other team members
Backup: Involves being available to assist other team members. This implies that members have an understanding of other members' tasks. It also implies that team members are willing and able to provide and seek assistance when needed, e.g.
• filling in for another member who is unable to perform the task
• helping another member correct a mistake
Coordination: Refers to team members executing their activities in a timely and integrated manner. It implies that the performance of some team members influences the performance of others. This may involve an exchange of information that subsequently influences another member's performance. Coordination represents the output of the model and reflects the execution of team activities such that members respond as a function of the behavior of others, e.g.
• passing performance-relevant data to other members in an efficient manner
• facilitating the performance of other members' jobs
Communication: Involves the exchange of information between two or more team members in the prescribed manner and using appropriate terminology. Often, the purpose of communication is to clarify or acknowledge the receipt of information, e.g.
• verifying information prior to making a report
• acknowledging and repeating messages to ensure understanding

Table 2
The use of Klein and Myers' principles in this field research.

The principles for interpretive field research [24]	How we used each principle
1. The fundamental principle of the hermeneutic circle	We improved our understanding of the project by moving back and forth between phases and events. The project had three main phases, which had different teamwork characteristics. For each of the phases, we described concrete events. The data analysis involved multiple researchers having ongoing discussions about the findings
2. The principle of contextualization	To clarify for our readers how situations emerged, we describe the work and organization of the company, as well as the context of the project we used to study teamwork
3. The principle of interaction between researchers and subjects	The researchers' understanding of the project developed through observations, interviews and discussions with the team participants in the coffee breaks and during lunch. We discussed project status, progress, and how issues were perceived by team participants
4. The principle of abstraction and generalization	We describe our findings and relate them to the model of Dickinson and McIntyre [13]
5. The principle of dialogical reasoning	We use Dickinson and McIntyre's model to identify areas of investigation in the case. Our assumptions are also based on the general literature of teamwork and self-management Our social background is European
6. The principle of multiple interpretations	To collect multiple, and possibly contradictory interpretations of events we collected data from all participants in the project and from multiple data sources. The case study narrative and findings have been presented to Alpha and led to feedback
7. The principle of suspicion	By means of the analysis, we were sensitive to how roles and personalities affected attitudes to teamwork to discover false preconceptions In addition to observations, we also performed interviews with different roles at different levels, and multiple interviews with all team members. This increased the chance of unveiling possibly incorrect or incomplete meanings

In the rest of the paper, we will explain the challenges that arise when introducing agile methods by appeal to the mechanisms that influence teamwork that are suggested by Dickinson and McIntyre [13].

3. Research method

We designed a single-case holistic study [49] of a project that used Scrum, focusing on mechanisms that influence teamwork. When designing the study, we focused on human sensemaking and on how the mechanisms of teamwork were understood by the people involved. Given that our study was an interpretative field study, we used the seven principles for conducting such studies that were proposed by Klein and Myers [24] in order to determine the main choices that were related to research method. Table 2 gives an overview of these principles and a description of how we used them.

3.1. Study context

The field study was conducted in a company that introduced Scrum in order to improve their ability to deliver iteratively and on time, increase quality, and improve teamwork. The company has three regional divisions with one separate ICT division. The ICT division consists of a consulting department, an IT management department, and a development department. The ICT division develops and maintains a series of off-the-shelf software products that are developed in-house, in addition to software development projects for outside customers. During the study, the development department had 16 employees, divided into a Java and a .Net group.

The goal of the project studied was to develop a plan and coordination system for owners of cables (e.g. electricity, fiber) and pipes (water, sewer). We refer to the project as "Alpha", because this was the first project for which the company used agile methods, in this case Scrum. Alpha produced a combination of textual user interfaces and map functionality. Alpha was to use a commercial package for the map functionality, which was to be customized by a well-known subcontractor located in another city. The subcontractor could only deliver their part of the system 4 weeks before the first deliverable to the customer. This was recognized as a risk, but it was decided that it would be even riskier to develop this component internally. The company would also be responsible for maintenance and support after final installation. Four thousand hours, six developers, one Scrum master, and a product owner were allocated to the project. The product owner was employed by the same company as the developers and acted as a representative for the client, which was the local government of a Norwegian city. Internally, there were plans for reusing deliveries from Alpha and to re-sell the product to other public departments when it was finished. An extra 800 h were allocated to achieve this aim. Before Alpha was begun in May 2006, some initial architectural work was done and some coding activities had started. Alpha used .Net technology and was supposed to last for 10 months.

The developers had usually worked alone on projects divided into modules or on smaller projects, so Alpha was the first experience of working on a larger project for most.

3.2. Data sources and analysis

The two first authors conducted direct observation and collected documents throughout the whole project. In addition, we interviewed the Scrum master, product owner, and developers (Table 3). The interview guide covered the components in the Dickinson and McIntyre model in addition to questions related to Scrum (Appendix A).

Table 3
Data sources.

Source	Comment
Observations and informal dialogues	Participant observation and observation of the daily stand-up, sprint and planning meetings, sprint reviews, and sprint retrospective as well as other meetings. The 60 observations and informal dialogues were documented in field notes, which also include pictures
Interviews	We interviewed the Scrum master and three developers in June 2006, five of the developers in September 2006, and all developers, the Scrum master, and the product owner after the project was completed (March 2007). The 17 interviews were all transcribed
Documents	Product backlog, sprint backlogs, burn-down charts, minutes from review, retrospective and planning meetings

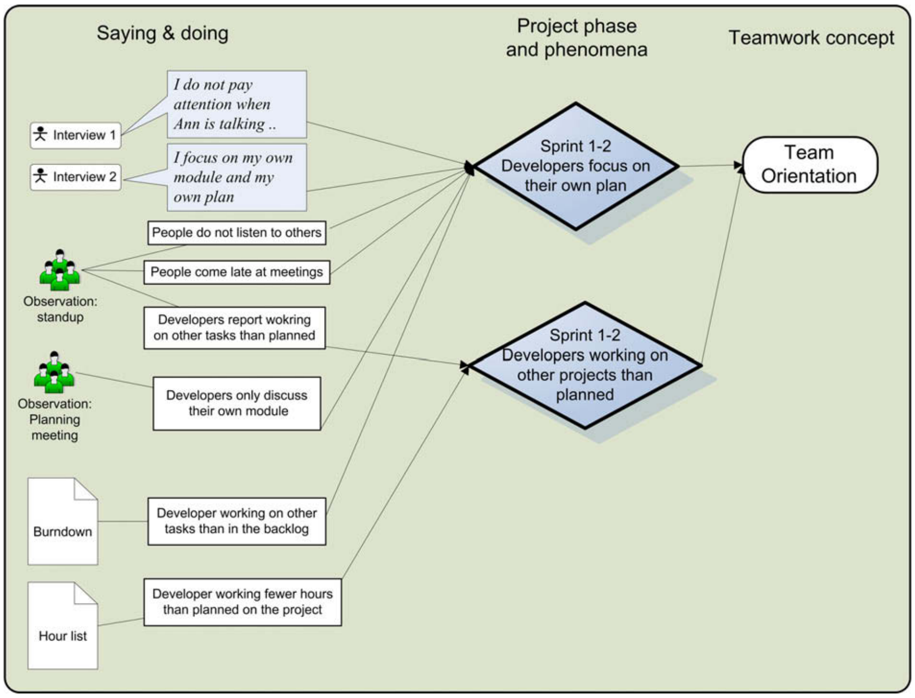
Saying & doing
Project phase and phenomena
Teamwork concept
Interview 1
I do not pay attention when Ann is talking ..
Interview 2
I focus on my own module and my own plan
People do not listen to others
People come late at meetings
Observation: standup
Developers report wokring on other tasks than planned
Observation: Planning meeting
Developers only discuss their own module
Burndown
Developer working on other tasks than in the backlog
Hour list
Developer working fewer hours than planned on the project
Sprint 1-2 Developers focus on their own plan
Sprint 1-2 Developers working on other projects than planned
Team Orientation

Fig. 2. Overview of the coding process. Example material from the concept "team orientation".

We visited the team once or twice a week, conducting a total of 60 observations, each of which lasted from 10 min to 8 h. We observed project meetings and developers working. We often discussed Alpha's status and progress, and how team participants perceived issues during their coffee breaks and lunch. Notes were taken on dialogues, interactions, and activities. The dialogues were transcribed and integrated with notes to produce a detailed record of each session. We also collected Scrum artifacts, such as product backlogs, sprint backlogs, and burn-down charts. All data from the interviews, observations, and documents were imported into a tool for analyzing qualitative data, Nvivo (www.qsrinternational.com). We categorized interesting expressions, observations, and text from documents, using the teamwork concepts proposed by Dickinson and McIntyre as the main categories.

We used a variety of strategies to analyze the material [27]. First, we described the project and context in a narrative to achieve an understanding of what was going on in the project. Then, we described aspects of teamwork using Dickinson and McIntyre's model by pointing to events in three main phases of the project, which had different teamwork characteristics.

In the analysis, we emphasized how events were interpreted by different participants in the project. Material to describe an event was taken across all sources and synthesized, as shown in the example in Fig. 2.

4. Results: teamwork in an agile project

The team that worked on Alpha organized the project according to generally recommended Scrum practices. Plans were made at the beginning of each sprint, after the team had reviewed what was produced in the previous sprint. Features were recorded in the sprint backlog. The team that worked on Alpha held three project retrospectives to identify and discuss problems and opportunities that arose during the development process. Daily meetings were organized throughout the project, though these were less frequent in the last two sprints. These meetings were usually about updating the others on progress, development issues, and the project in general. The daily meetings we observed lasted from 10 to 35 min, but were usually shorter than 15 min. The product owner, who was situated in another city, often participated in these meetings by telephone. He participated because both he and the Scrum master thought that it was important to share information constantly and participate in the decision-making process.

Alpha began in May 2006, with the first installation planned for October and the final installation for November 2006. However, the first installation was not approved until December 2006 and from January 2007, two developers continued working with change requests until the final installation was approved in October 2007. Five of the sprints lasted 1 month, the sprint during summer for two. Fig. 3 shows major events in the project together with a project-participant satisfaction graph. This figure was created by the team in the final project retrospective and was based on a timeline exercise [12]. To create the project-participant satisfaction graph, each team member first drew his own graph for the emotional ups and downs during the project, after which the graphs were merged.

In the initial planning phase, before coding began, several meetings were used to discuss the overall architecture, and decide on the technology and development platform. As can be seen from Fig. 3, the team was frustrated in this period, because of what the team described as "endless discussion without getting anywhere". After Scrum was introduced and code writing began, the team was more satisfied with Alpha. In the first retrospective, the team itself concluded that the team members were taking responsibility, that they were dedicated to the project, and that the team was protected against external issues. Meetings and work were perceived as well-coordinated. During the first retrospective, a developer said:

> Earlier we worked more alone, and when you got a project doomed to failure, you would get a lot of negative response. That was unpleasant. Now we share both the risk and opportunities.

The team was satisfied with their performance in sprints 1–4. However, in sprint 5, problems with integrating a deliverable from the subcontractor emerged, which resulted in the two last sprints being chaotic and the project being delayed. During this period, we saw many empty pizza boxes in the office space, which

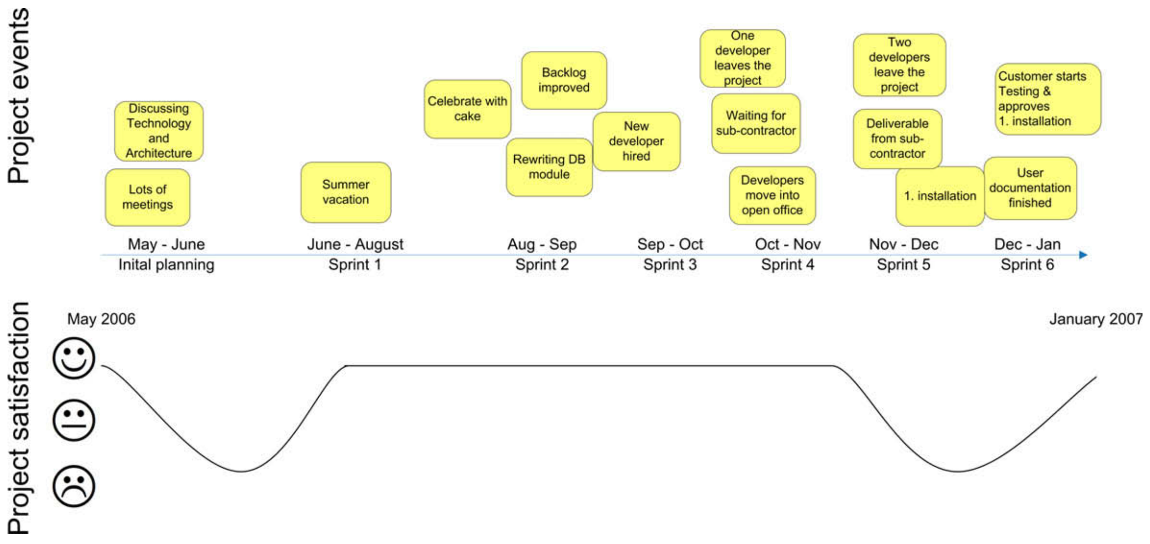

Fig. 3. Main events in the project and project satisfaction.

indicated that the developers were working late. Developers told us they also worked at weekends. The team became less satisfied, as shown in Fig. 3.

However, after the client had approved the first installation, the team became more satisfied. In the last retrospective, the team described the project as a good one, except for the problems related to the deliverable from the subcontractor. Then again, the team saw this as something beyond their control.

Despite the teams' overall satisfaction with the teamwork, throughout the project we observed problems with completing the backlog and following the sprint plan, unproductive meetings, developers often being silent in the planning meetings, and developers often reporting working on issues other than those that it had been initially planned to work on. In addition, the developers received little feedback when talking about what they were doing. In what follows, we will use the data we collected to try to explain some of our observations.

4.1. Introducing Scrum: sprints 1–2

The project leader participated in a Scrum master certification course. The first sprint was initiated with a two-day Scrum course. The first day was spent on introducing Scrum to the whole development department, the second on planning the first sprint for Alpha.

The first sprint completed most of the backlog, and the team was satisfied with the progress. However, in the first retrospective (see Fig. 4), the team reported problems with both defining a stable sprint backlog and finishing it. We observed these problems as well. The team also ended up working on tasks that were not discussed or identified during the sprint planning meeting.

In this company, each team member is usually assigned to work on a specific software module from the beginning to the end. This way of working is known as an isomorphic team structure [21]. The advantages with this structure are that it is organizationally

Fig. 4. From the review and retrospective meeting in sprint 2.

simple, it allows many tasks to be completed in parallel, and task responsibilities can be clearly defined and understood. The Scrum master subscribed to this view [interview]:

> Let the person who knows most about the task perform it! We cannot afford several people doing the same thing in this project. We need to continue working as we have done before.

The team mostly kept this structure after introducing Scrum. A developer said [interview]:

> Because we have to deliver every month, there is never time to swap tasks.

Because of the division of work, the developers typically created their own plan for their own module, often without discussing it with the team. A developer commented [interview]:

> Some are more motivated by the perfect technical solution, than thinking of when things need to be done.

In this phase, one developer even implemented features for future projects, without informing the others (this kind of behavior is often referred to as decision hijacking [3]). This was discussed in a daily stand-up of the second sprint:

> Developer: The customer databases will be used by several applications, so I have implemented support for dealing with various technologies, including Oracle. It took a lot of time.
> Scrum master: Did we not agree on postponing this?
> Developer: We need this later and now it is done.

This illustrates how developers prioritized individual goals over team goals, and subsequently a lack of *team orientation*. As a result of this incident, the Scrum master lost trust in this developer and started to supervise him. Consequently, the developer was not part of the *team leadership* any more, even when discussing modules where he was seen as the expert. We observed that he was sometimes absent from the daily meetings. This is consistent with findings from Bandow such that if team members do not feel that their input is valued, they may be less willing to share information [4].

The Scrum master also observed that the team was not reporting problems. In interviews, we found that the developers thought that the Scrum master was overreacting to problems stated at the daily meetings, which resulted in the team not reporting problems when the Scrum master was present. After the Scrum master confronted the team with this issue, the situation improved. However, for the rest of the project, the Scrum master still felt that problems were reported too late. This was confirmed by our observations of daily stand-ups.

In the second retrospective of Alpha, we found two more reasons for problems not being reported: problems were discovered late and they were seen as personal. One developer said:

> People working alone results in the team not discovering problems, because you do not get feedback on your work.

Because of the isomorphic team structure, the developers perceived new emerging tasks and new problems as personal; as a result, they did not seek assistance when needed. They focused on their own modules, which resulted in problems with *monitoring* each other and subsequently with giving *feedback* and implementing *backup behavior*. In the second retrospective meeting, one developer said:

> When we discover new problems, we feel we own them ourselves, and that we will manage to solve them before the next meeting tomorrow. But this is not the case, it always takes longer.

When individuals are independent and have more control over their schedule and the implementation of their tasks, there is less interaction between the group members [26]. One developer said [Retrospective sprint 2]:

> When it comes to the daily scrum, I do not pay attention when Ann is talking. For me, what she talks about is a bit far off the topic and I cannot stay focused. She talks about the things she is working on. I guess this situation is not good for the project.

In Alpha, this resulted in problems with *communication*, giving *feedback*, and the possibility of *monitoring* teammates. In addition, *team orientation* was hindered, because information sharing and *feedback* was delayed by people not listening. It seemed that the isomorphic team structure resulted in individual goals being seen as more important than team goals.

In this phase, the team spent more than 100 h rewriting a module. The developer responsible for the module said [interview]:

> I was supposed to create a database that every project could use. After I had created it, I explained how it was done during a stand-up, and then I went on vacation. Later, when they started using it, they did not understand how it was supposed to be used, and they decided to rewrite the whole module. The team had probably not understood what I was talking about when I explained the database in the daily stand-up. If I had not gone on vacation they would not have needed to do the rewriting... another problem is the daily meeting. It's only a short debrief, there is never time to discuss what you are working on.

The developer did not verify that the team had understood how he had implemented the module (*communication*), and no one gave *feedback* to the effect that they did not understand how the module was implemented during the stand-up. In addition to missing *monitoring*, the lack of communication and feedback was the reason for the rewriting, and the consequence was reduced progress and team efficiency.

In the second retrospective, the team concluded [retrospective report]:

> The team must work more on the same tasks, and then no one will sit alone. Working alone results in knowledge not being disseminated, and there is no backup. Also, problems are being discovered late and developers not getting feedback on their work.

4.2. Everyday work: sprints 3–4

When the team was getting used to planning and conducting sprints, work was perceived as motivating and developers expressed satisfaction at having something completed early in the project. However, in this phase, many expressed having problems in transferring what was written in the sprint backlog to actual work tasks. Initially, the team viewed this difficulty as being caused by the introduction of "features" rather than the specification of technical requirements. However, the difficulty seemed to run deeper. One developer expressed:

> I have the impression that the sprint backlog has been somewhat distanced from what we have really planned to do.

Another stated:

> It is really difficult to get answers to questions, because no-one really knows where we are going.

This indicates that the team lacked a clear idea of how to achieve the final result. Applying two different visions for Alpha, one covering the Alpha project and the other future projects, did not help this situation and weakened the *team orientation*. *Team leadership* was already weak and was seldom shown by team members other than the Scrum master and the product owner. The product owner described the challenge with giving clear direction, support, and structure to the developers [interview]:

> You need to give an answer according to what you think, and sometimes I'm not certain I'm giving the right answer.... and you need to give a quick answer; otherwise the developers will start doing something else.

The team discussions, *communication*, and *feedback* improved in this period. One developer said [interview]:

> Using Scrum forces us to work closer with each other, and the result is more communication.

Another said:

> The good thing about Scrum is that Scrum reminds us to talk to each other about the project.

However, often, discussions ended without conclusion. One developer said [interview]:

> When we discuss technical issues, it often ends in a kind of "religious" discussion, and then I give up. And then you let people continue to do what they are doing.

This shows a challenge with respect to *team leadership*. The person leading the discussion does not listen to the concerns of other team members.

In this period, we observed the structure of the stand-up proposed by Scrum being followed. However there was little *communication, coordination*, and *feedback* between the developers in these meetings. One developer said after a stand-up:

> The daily meetings are mostly about reporting to the Scrum master. When he is not there, the meetings are better because then we communicate with each other.

Another developer said:

> When he is in the meeting we often end up only giving a brief report about status and not the issues we need to talk about.

Without a clear understanding of the system being developed, planning was difficult. In addition, the monthly planning meetings somehow excluded the developers and turned out to constitute *communication* only between the Scrum master and the product owner. During the retrospective, the team identified a need to spend more time planning, but two of the developers whom we observed being silent in the planning meeting thought that they spent too much time on planning. Nevertheless, a lack of thorough discussion was probably one reason for important tasks sometimes not being identified before the end of each sprint. This reduced the validity of the common backlog, did not strengthen the *communication, coordination* of tasks or the possibility of giving *feedback*, and again resulted in the developers focusing more on their own plan, thereby weakening the *team orientation*.

Another reason for the developers performing tasks other than those identified in the planning meeting was the need to adapt to the constantly changing environment. The high complexity of the project and open issues regarding technology, client, and subcontractor resulted in a high level of uncertainty when creating the sprint backlog. We observed the team being sensitive to changes in both the internal and external environment. However, the team did not manage to update the plan to adapt to the changing conditions. Subsequently, it was unclear how much progress had been made, which made it difficult to *monitor* team members' performance.

Despite the lack of *monitoring*, the developers did sometimes look at each other's code. One developer described [interview] the difficulty of giving *feedback* and raised the issue of trust regarding this matter:

> You look at someone's code, and then you think, that was a strange way of doing it. There is no problem getting criticism from people you feel safe with, but when you get feedback from people you do not like, it is different. It is also difficult to give feedback when you are not 100% sure you know that your way of doing it is better.

4.3. Emergency Scrum: sprints 5–6

The major event in this phase was that the deliverable from the subcontractor was delayed, and when it was delivered it was found not to work as intended. The code was unstable and the response time was too long. This came as a surprise to the team. One developer said [interview]:

> This was a shock to us. The end users could not start testing and we had to spend a lot of time trying to fix this. It took almost a month to locate the problems.

Given that the developers were specialized, only two developers worked on this problem, even at weekends. One developer explained [interview]:

> It's chaotic now. We work long hours, but I do not do too much. I have done what I was supposed to, and I cannot help them. I do not know anything about what they are doing, so it does not help if I try assist.

The isomorphic team structure and missing *monitoring* resulted in a lack of *backup behavior*.

The integration problem resulted in a backlog not being finished, but it also became evident that not being strict about the criteria for marking work as having been completed affected this. One developer said [interview]:

> We classified tasks as finished before they were completed and we knew there was still work to be done. It seems that the Scrum master wants to show progress and make us look a little better than we really are. These tasks are then not on the list of the next sprint since they officially are done, but we know there is still some more work needed. Each sprint starts with doing things that we have said were finished and then you know you will not finish the sprint.

The Scrum master and some team members gave the impression that the team was better than they actually were and this is related to a particular challenge to *team leadership*, which is known as "impression management" [32]. Impression management is a barrier for learning and improving work practices.

Before this last period began, developers had given priority to the project for 6 months and had worked more than initially planned. Now, new projects began to start, and because the completion of Alpha was planned to release resources at this time, several developers were supposed to start working full-time on the new projects. Alpha started losing resources, and this became a problem because of poor *backup*. One developer said [interview]:

> When correcting errors in this phase, each person was responsible for correcting the errors he had introduced. This is not how it should have been done.

The Scrum master said [interview]:

> We are having problems in one of the modules, but other developers do not want to fix the problem. They want to wait for the developer who wrote the code.

In this phase, it also became clear that insufficient attention had been paid to long-term planning. One developer said [interview]:

> It turned out that certain parts of the system were simply forgotten. There has been a failure somewhere... the product owner and the client asked for things that no one had thought of and that were not in the backlog.

Two reasons for this omission of certain parts of the system were that the dissemination of information was not *coordinated* among the team-members and that no one had the responsibility for the overall technical solution. Dissemination of information among the team-members became an even bigger challenge at the end. Because the team was losing resources, key personnel were absent from the daily stand-up, which resulted in the rest of the team having problems in *monitoring* the progress of the project and in coming to a common understanding of the changing situation.

After the last planned sprint, only two developers continued to work on the project, in addition to the Scrum master and the product owner. The rest of the team members joined other projects, as had been planned earlier. The two remaining programmers spent 1400 h on finalizing the project (correcting errors and doing more testing). The final testing was done 7 months after the last sprint.

The client was satisfied with the delivered functionality but not with the system performance.

5. Discussion

We have described the introduction of the agile process Scrum in a software development project, using the teamwork model proposed by Dickinson and McIntyre [13], Fig. 1. We now discuss the case in light of our research question: "How can we explain the teamwork challenges that arise when introducing a self-managing agile team?" We found the following:

Dickinson and McIntyre proposed that team leadership and team orientation promote team members' capability to monitor their teammates' performance. This does not seem to be borne out by our case study, in a number of ways. Due to the isomorphic team structure, the developers focused on their own modules and often created their own plan and made their own decisions. In addition, problems were seen as personal. This low team orientation on the part of the developers resulted in them not knowing what the others were doing, and as a result it was difficult to monitor others' performance. The team members seemed to be used to having a very high degree of individual autonomy. This created problems when the team members tried to change their normal way of working to become part of a self-managed team. Our findings confirm previous research by Langfred [26], such that there can be a negative effect on team performance when teams are trying to function as a self-managed team when the team members have high individual autonomy. Team leadership was also not distributed as it should be in a self-managing team [32]. Only a few team-members participated in the decision-making, and the Scrum master focused more on command-and-control than providing direction and support for other team members. The Scrum master even ended up supervising one developer because this developer implemented features for future projects, without informing the others. Because the team-members felt the Scrum master overreacted when they reported problems, they started reporting fewer problems, which again limited the possibility of monitoring each other.

The Dickinson and McIntyre model suggests that performance monitoring drives both the content of feedback and timely backup behavior. Due to the fact that the team members did not monitor each other much, there was little feedback and almost no backup, which become evident when the team started to lose resources at the end of the project. The team members did provide some positive feedback, but several found it difficult to both give and accept negative feedback. Our findings also confirm the results of previous research by Levesque et al. [28], such that when the roles that group members play become increasingly specialized and as a result reduce team redundancy and backup, there is a corresponding decline in the amount of time that team members spend working with or communicating with each other. This is also consonant with Marks et al. [30], such that if effective backup is to be provided, teammates need to be informed of each others' work in order to identify what type of assistance is required at a particular time. Marks et al. [30] identify three ways of providing such backup: (1) providing a teammate with verbal feedback or coaching, (2) physically assisting a teammate in carrying out a task, or (3) completing a task for a teammate when it is observed that the workload is too much for him. These means seems to be missing at Alpha.

Dickinson and McIntyre argue that, when all the aforementioned teamwork competences occur in unison, they serve synergistically as a platform for team coordination. The Alpha team had problems with all the teamwork competences and as a consequence they had problems coordinating the teamwork. Important tasks were even forgotten. Marks et al. [30] also argue that when teams have communication problems they are likely to experience problems with coordinating their work.

According to the Dickinson and McIntyre model, the feedback resulting from team coordination should serve as input back into the team processes. The team identified early on [second retrospective] the need for developers to start working on the same tasks, the lack of backup, problems not being reported, and lack of feedback. The researchers observed, and the team members thought, that the teamwork improved during the course of the project. However, it seemed to be difficult to change the teamwork, because changing meant changing not only the developers' way of working but also organizational structures. The team worked better together in the later phases, but did not improve the team orientation and team leadership in such a way that monitoring was improved. One reason is the observation of what Morgan [32] defines as "impression management", when the team gave the impression to be better than they actually were. Impression management is a barrier to learning [32]. Continuous self-management requires a capacity for learning that allows operating norms and rules to change in response to changes in the wider environment [32].

In the Dickinson and McIntyre model, communication acts as the glue that links together all other teamwork processes. In Scrum, the daily stand-up is the most important mechanism for achieving such communication. Everyone should communicate with everyone else. However, because of problems with team leadership and a lack of monitoring, these meetings were mostly used by the Scrum master for getting an overview of what was going on in the project. Developers were reporting to the Scrum master and not talking to each other. Communication improved when the Scrum master was absent. As a result of the highly specialized skills and corresponding division of work, there was less interaction and communication between the group members. However, this improved in the last phase of the project because by that time, the developers had become accustomed to talking to each other at the daily stand-up.

5.1. Implications for theory

The Dickinson and McIntyre model explains most of our observations. However, it does not model any of the critical antecedents and outcomes of the team process. In the case of Alpha, it was obvious that the team had problems becoming a well-functioning team from the beginning, and that this was one reason for the team having problems in self-managing.

In addition, Dickinson and McIntyre do not describe certain important components, such as trust and shared mental models. We observed that the team had not developed trust at the group level. A lack of trust among the Scrum master and the members of the team was an important reason for why problems were not reported and why a team member was given instructions on what to do. Our findings are consonant with those of Salas et al. [39]: without sufficient trust, team members will expend time and energy protecting, checking, and inspecting each other as opposed to collaborating to provide value-added ideas. It is evident that trust is a prerequisite for shared leadership, feedback, and communication. Our finding regarding the lack of trust also confirms previous research on trust [4], such that team members may not be willing to share information if they fear being perceived as incompetent.

The team lacked a shared mental model on what the outcome of the project should be. Working cooperatively requires the team to have shared mental models [39]. Our results are also consonant with Salas et al.'s [39] findings that without a shared understanding, the individual members may be headed toward different goals, which in turn will lead to ineffective/lack of feedback or assistance. Shared mental models are also a prerequisite for communication, monitoring, and team orientation. In addition, our finding confirms the results of previous research on shared mental models in software development teams, such that not all teams develop increasingly shared mental models over time [28].

Having problems with trust and developing shared mental models could also be a reason why the team did not manage to change the team process more than we observed. In addition, the previous ways of working in the company hindered effective teamwork, and in this setting the team did not succeed in improving their teamwork skills significantly during the project.

For theory, this study shows that:

- There is a vast literature on teamwork that is very relevant for agile development and that deserves more attention.
- Dickinson and McIntyre's model [13] should be extended to include trust and shared mental models.

5.2. Implication for practice

Agile software development emphasizes that teams should be self-managed. However, Scrum and agile methods offer no advice on how shared leadership should be implemented. A practical implication of Langfred's [26] findings is that, if an organization believes in letting teams be more self-managing, great care must be taken in the implementation. This is especially important when the team members have high individual autonomy.

The Alpha project was the first big project for most developers. Even though they had worked together for years, they should probably have spent more time together focusing on improving teamwork in the initial phase of the project. The successful teams that Katzenbach et al. [22] observed all gave themselves the time to learn to be a team. If developers who work together have problems becoming a team, they will also have problems becoming a self-managing team.

What people should do to provide backup is not specified clearly in Scrum. In the literature on self-managed teams, backup behavior has been identified as an important prerequisite for self-management [32,35]. In our study, highly specialized skills and a corresponding division of work was the most important barrier to achieving backup and then self-management.

Scrum is not very specific on how to establish monitoring in development teams, although this is implicitly a prerequisite for feedback, coordination, and backup. Combining Scrum with, for example, the practice of pair programming in XP [6] would improve monitoring, feedback, and backup.

We believe that our study has the following main implications for practice:

- An isomorphic project structure will hinder teamwork because the division of work will make it more difficult for developers to develop shared mental models, trust each other, communicate, coordinate work, and provide backup. One way of handling this is to organize cross-training and appreciate generalist to build redundancy in the organization [31].
- Self-management should be enabled when starting to use agile methods such as Scrum, and be aware that high individual autonomy may results in problems creating a self-managing team.
- The way in which agile practices are taken up is dependent on the companies' former development process. Changes take time and resources, and for the company in this study, previous practices were sustained throughout their first agile project.
- The development process should be adjusted for enabling efficient work, by making room for reflection and learning. However, achieving learning in software processes is not trivial.

5.3. Limitations

The main limitations of our study are the single-case design and the possibility of bias in data collection and analysis. The fact that we used a single-case holistic design makes us more vulnerable to bias and eliminates the possibility of direct replication or the analysis of contrasting situations. Therefore, the general criticisms about single-case studies, such as uniqueness and special access to key informants, may also apply to our study. However, our rationale for choosing Alpha as our case was that it represents a critical case for explaining the challenges for teamwork that arise when introducing self-managing agile teams. We used Alpha to determine whether we could confirm, challenge, or extend Dickinson and McIntyre's [13] teamwork model. Our goal was not to provide statistical generalizations about a population on the basis of data collected from a sample of that population. On the contrary, our mode of generalization is analytical, i.e., we used a previously developed theory as a template with which we compared the empirical results of the case study, which is similar to Yin's [49] concept of Level Two inference.

Another possible limitation is that we based much of our data collection and analysis on semi-structured interviews [14]. The use of multiple data sources made it possible to find evidence for episodes and phenomena from more than one data source; we also observed, talked to, and interviewed the team members over a period of 9 months, which made it possible to study the phenomena from different viewpoints as they emerged and changed.

Could it be that we as researchers influenced the teamwork characteristics by our presence in the project? Our presence and questions might have made the team members more aware of teamwork characteristics, but we do not think their behavior was influenced by our presence. The everyday demands of the projects were high, and we did not observe changes in behavior that seemed to relate to our interview or observation phases.

5.4. Future work

The results of this study point out a number of directions for future research. Firstly, our study highlights several challenges that must be met when self-managing teams are introduced into agile development. Accordingly, further work should focus on identifying and addressing other problems that may arise when introducing agile development.

Secondly, the extended teamwork model should be used for studying mature agile development teams, in order to get a better understanding of the main challenges in such teams. Also, teams using shorter sprints (e.g. 2–3 weeks) should be studied, since this will give the team more frequent feedback, which affect team learning and the other elements in the Dickinson and McIntyre teamwork model.

Thirdly, our study tries to answer "How can we explain the teamwork challenges that arise when introducing a self-managing agile team?" through Dickinson and McIntyre's teamwork model. However, there are other relevant streams of research to address the adoption of methods and technology, e.g. the diffusion of innovation literature [37]. Other models that attempt to explain the relationship between user perceptions, attitudes and use intentions include the technology acceptance model (TAM) [11], and the theory of planned behavior [2].

6. Conclusion

We have conducted a nine-month field study of professional developers in a Scrum team. We found that the model of Dickinson and McIntyre [13], together with trust and shared mental models, explain our findings. In addition to these teamwork components, highly specialized skills and a corresponding division of work was the most important barrier for achieving effective teamwork. We have also seen that Scrum has several mechanisms in place for supporting the recommendations of the framework, but that many of these mechanisms are not easy to implement in practice.

Transitioning from individual work to self-managing teams requires a reorientation not only by developers but also by management. Making such changes takes time and resources, but it is a prerequisite for the success of any kind of agile method based on self-management.

Acknowledgement

This work was supported by the EVISOFT project, which is partially funded by the Research Council of Norway under Grant 174390/I40.

Appendix A

A.1. Interview guide

The respondent was informed of the nature of the study and how long the interview will take. The respondent was told why it is important to tape the interview and that only the researchers would have access to the transcript. The respondent was finally asked if he/she would agree to the interview being taped.

Questions for warm-up:

- What are you working on now?
- What is the status of the project?

Main body of the interview:

- How is work coordinated in the project?
  - How was it done in earlier projects?
- How are problems that emerge in the project solved?
  - How was it done in earlier projects?
- Do you have an overview of what others are doing?
  - How was it done in earlier projects?
- How easy is it to carry on work that was begun by others?
  - How was it done in earlier projects?
- How do you discover changes in the project?
  - How was it done in earlier projects?
- How do you deal with changes in the project?
  - How was it done in earlier projects?
- Does the team have a common project goal?
  - Did earlier projects have a common project goal?
- Does everyone know the expected outcome of the project?
  - How was it done in earlier projects?
- Do team members give each other feedback in the project?
  - How was it done in earlier projects?
- Do team members share relevant project information with each other?
  - How was it done in earlier projects?
- How is the team communication?
  - How was it done in earlier projects?
- How is the team performance?
  - How was it done in earlier projects?
- How do you think Scrum is working in the project?
  - What is working?
  - What is not working?
- Is there anything else you would like to add that you think is interesting in this context, but not covered by the questions asked?

References

[1] S.T. Acuna, M. Gomez, N. Juristo, How do personality team processes and task characteristics relate to job satisfaction and software quality?, Information and Software Technology 51 (3) (2009) 627–639

[2] I. Ajzen, The theory of planned behavior, Organizational Behavior and Human Decision Processes 50 (2) (1991) 179–211.

[3] A. Aurum, C. Wohlin, A. Porter, Aligning software project decisions: a case study, International Journal of Software Engineering and Knowledge Engineering 16 (6) (2006) 795–818.

[4] D. Bandow, Time to create sound teamwork, The Journal for Quality and Participation 24 (2) (2001) 41–47.

[5] J.R. Barker, Tightening the iron cage – concertive control in self-managing teams, Administrative Science Quarterly 38 (3) (1993) 408–437.

[6] K. Beck, C. Anders, Extreme Programming Explained: Embrace Change, second ed., Addison-Wesley, 2004.

[7] R.M. Belbin, Team Roles at Work, Butterworth-Heinemann, Boston, MA, 1993.

[8] B.W. Boehm, R. Turner, Balancing Agility and Discipline: a Guide for the Perplexed, Addison-Wesley, 2003.

[9] C.S. Burke, K.C. Stagl, C. Klein, G.F. Goodwin, E. Salas, S.A. Halpin, What type of leadership behaviors are functional in teams? A meta-analysis, Leadership Quarterly 17 (3) (2006) 288–307.

[10] S.G. Cohen, D.E. Bailey, What makes teams work: group effectiveness research from the shop floor to the executive suite, Journal of Management 23 (3) (1997) 239–290.

[11] F.D. Davis, Perceived usefulness, perceived ease of use, and user acceptance of information technology, MIS Quarterly 13 (3) (1989) 319–340.

[12] E. Derby, D. Larsen, Agile retrospectives: making good teams great, Pragmatic Bookshelf, 2006.

[13] T.L. Dickinson, R.M. McIntyre, A conceptual framework of teamwork measurement, in: M.T. Brannick, E. Salas, C. Prince (Eds.), Team Performance Assessment and Measurement: Theory, Methods, and Applications, Psychology Press, NJ, 1997, pp. 19–43.

[14] T. Diefenbach, Are case studies more than sophisticated storytelling? Methodological problems of qualitative empirical research mainly based on semi-structured interviews, Quality and Quantity 43 (6) (2009) 875–894.

[15] T. Dybå, Improvisation in small software organizations, IEEE Software 17 (5) (2000) 82–87.

[16] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: a systematic review, Information and Software Technology 50 (9–10) (2008) 833–859.

[17] R.A. Guzzo, M.W. Dickson, Teams in organizations: recent research on performance and effectiveness, Annual Review of Psychology 47 (1996) 307–338.

[18] J.R. Hackman, The design of work teams, in: J. Lorsch (Ed.), Handbook of Organizational Behavior, Prentice-Hall, Englewood Cliffs, NJ, 1987.

[19] M. Hoegl, H.G. Gemuenden, Teamwork quality and the success of innovative projects: a theoretical concept and empirical evidence, Organization Science 12 (4) (2001) 435–449.

[20] M. Hoegl, K.P. Parboteeah, Autonomy and teamwork in innovative projects, Human Resource Management 45 (1) (2006) 67–79.

[21] J. Jurison, Software project management: the manager's view, Communications of AIS 2 (1999).

[22] J.R. Katzenbach, D.K. Smith, The discipline of teams, Harvard Business Review 71 (2) (1993) 111–120.

[23] B.L. Kirkman, B. Rosen, Beyond self-management: antecedents and consequences of team empowerment, Academy of Management Journal 42 (1) (1999) 58–74.

[24] H.K. Klein, M.D. Myers, A set of principles for conducting and evaluating interpretive field studies in information systems, MIS quarterly 23 (1) (1999) 67–93.

[25] R.E. Kraut, L.A. Streeter, Coordination in software development, Communications of the ACM 38 (3) (1995) 69–81.

[26] C.W. Langfred, The paradox of self-management: individual and group autonomy in work groups, Journal of Organizational Behavior 21 (5) (2000) 563–585.

[27] A. Langley, Strategies for theorizing from process data, Academy of Management (1999) 691–710.

[28] L.L. Levesque, J.M. Wilson, D.R. Wholey, Cognitive divergence and shared mental models in software development project teams, Journal of Organizational Behavior 22 (2001) 135–144.

[29] S.E. Markham, I.S. Markham, Self-management and self-leadership reexamined: a levels-of-analysis perspective, The Leadership Quarterly 6 (3) (1995) 343–359.

[30] M.A. Marks, J.E. Mathieu, S.J. Zaccaro, A temporally based framework and taxonomy of team processes, Academy of Management Review 26 (3) (2001) 356–376.

[31] N.B. Moe, T. Dingsøyr, T. Dybå, Overcoming barriers to self-management in software teams, Software, IEEE 26 (6) (2009) 20–26.

[32] G. Morgan, Images of Organizations, SAGE publications, Thousand Oaks, CA, 2006.

[33] S. Nerur, V. Balijepally, Theoretical reflections on agile development methodologies – the traditional goal of optimization and control is making way for learning and innovation, Communications of the ACM 50 (3) (2007) 79–83.

[34] S. Nerur, R. Mahapatra, G. Mangalaraj, Challenges of migrating to agile methodologies, Communications of the ACM 48 (5) (2005) 72–78.

[35] I. Nonaka, H. Takeuchi, The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation, vol. 12, Oxford University Press, New York, 1995.

[36] L. Rising, N.S. Janoff, The Scrum software development process for small teams, IEEE Software 17 (4) (2000) 26–32.

[37] E.M. Rogers, Diffusion of Innovations, fourth ed., The Free Press, New York, 1995.

[38] V. Rousseau, C. Aube, A. Savoie, Teamwork behaviors – a review and an integration of frameworks, Small Group Research 37 (5) (2006) 540–570.

[39] E. Salas, D.E. Sims, C.S. Burke, Is there a "big five" in teamwork?, Small Group Research 36 (5) (2005) 555–599

[40] E. Salas, K.C. Stagl, C.S. Burke, G.F. Goodwin, Fostering Team Effectiveness in Organizations: Toward an Integrative Theoretical Framework. in: 52nd Nebraska Symposium on Motivation, Lincoln, NE, 2007.

[41] J. Sapsed, J. Bessant, D. Partington, D. Tranfield, M. Young, Teamworking and knowledge management: a review of converging themes, International Journal of Management Reviews 4 (1) (2002) 71–85.

[42] B. Schatz, I. Abdelshafi, Primavera gets agile: a successful transition to agile development, IEEE Software 22 (3) (2005) 36–42.

[43] K. Schwaber, Beedle, Agile Software Development with Scrum, Prentice Hall, Upper Saddle River, 2001.

[44] H. Takeuchi, I. Nonaka, The new product development game, Harvard Business Review (64) (1986) 137–146.

[45] J. Tata, S. Prasad, Team self-management, organizational structure, and judgments of team effectiveness, Journal of Managerial Issues 16 (2) (2004) 248–265.

[46] B.B.W. Tuckman, Developmental sequence in small groups, Psychological Bulletin 63 (1965) 384–399.

[47] M. Uhl-Bien, G.B. Graen, Individual self-management: analysis of professionals' self-managing activities in functional and cross-functional work teams, Academy of Management Journal 41 (3) (1998) 340–350.

[48] M.A. West, The social psychology of innovation in groups, in: M.A. West, J.L. Farr (Eds.), Innovation and Creativity at Work: Psychological and Organizational Strategies, Wiley, Chichester, 1990, pp. 309–333.

[49] R.K. Yin, Case Study Research: Design and Methods, vol. xiv, Sage, Thousand Oaks, CA, 2009. p. 219 s.