Understanding Shared Leadership in Agile Development: A Case Study

Nils Brede Moe[1], Torgeir Dingsøyr[2], Øyvind Kvangardsnes[2]
[1]SINTEF ICT, NO-7465 Trondheim, Norway
[2]Dept. of Computer and Information Science, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway
nilsm@sintef.no, dingsoyr@idi.ntnu.no, oyvinkva@stud.ntnu.no

Abstract

In agile software development methods such as Scrum, software is developed in self-organizing teams. In such teams, leadership should be diffused rather than centralized; also the team-members need to affect managerial decisions for achieving the benefits of a self-managed team. When the team and team leaders share leadership, leadership is rotated to the person with the key knowledge, skills, and abilities for the particular issues facing the team at any given moment. Therefore, we argue that the team leadership in Scrum should be divided among the Product-owner, Scrum-master, and the self-organizing team. If teams are to succeed at implementing shared leadership in Scrum, not only do the vertical (or traditional) leaders need training and development but so too do the team members themselves.

1. Introduction

Agile software development [11] represents a new approach for planning and managing software projects. Agile development differs from the traditional plan-driven approaches as it puts less emphasis on up-front plans and strict plan-based control and more emphasis on mechanisms for change management during the project [25]. Further, agile development relies on people and their creativity rather than on processes [7]. Leadership and collaboration, informal communication and an organic (flexible and participative, encouraging cooperative social action) organizational form are other characteristics of agile software development [24].

Cohn and Ford [9] have referred to a number of possible setbacks or errors that may occur when an organisation is making the transition from plan-driven towards agile processes. The problems are mainly caused by resistance to, or over-enthusiasm for, agile practices within a software development team. When changing from plan-driven to change-driven, this may also impact several aspects of the organization including its structure, culture, and management practice [24]. Neither culture nor mind-sets of people can be easily changed, which makes the move to agile methodologies all the more formidable for many organizations [6]. Software development processes depend significantly on team performance, as does any process that involves human interaction. Two important factors to achieve team performance are feedback and communication. Also leadership, in addition with composition and motivation, are of near-universal importance to teams [15, 28].

Agile development relies on teamwork, as opposed to individual role assignment that characterizes plan-driven development [24]. In a team following a plan-driven model, the usually independently focused self-managing professionals have role specifications, including a specialized leadership role. This view on leadership assumes that there is usually one person who has much more influence that other members. The opposite view is that that leadership is no different from the social influence process occurring among all members of a group, and leadership is as a collective process among the members [37]. Agile methodologies supports this view and requires a shift from command-and control management to leadership-and-collaboration [24].

Augustine et al. [4] describe agile project management at a form of enabling the project managers and employees to adapt to changing circumstances rather than imposing formal control. They refer to Complex Adaptive Systems, which explains for example the complex behaviour of ants as based on a limited number of simple rules. A question is then, what kind of simple rules should be in place to enable leadership in an agile team? And if leadership is supposed to emerge from simple rules, what is the role of the managers? How should good leadership be established?

Motivated by the importance of both teams and team leadership in software development, and the challenge of introducing self-organizing teams in change-driven development, our research question is:

What are the main challenges in establishing leadership in an agile software development project?

The remainder of the paper is organized as follows. In Section 2, we present background information on agile development, and we use the literature to understand team leadership. In section 3, we describe our research method in detail. In Section 4, we present results from a single-case study of a Norwegian software company on team leadership in agile development. We discuss our findings in Section 5. Section 6 concludes and presents key findings from the study.

2. Agility and team leadership

In this section we use the literature to describe agile development and Scrum, self-organizing teams and two major characteristics which have to be in place for self-organized teams: Shared leadership and good mechanisms for learning.

2.1. Agile development and Scrum

Agile software development comprises a number of practices and methods [1, 8, 11, 12]. Among the most known and adopted agile methods are Extreme Programming (XP) [5] and Scrum [29]. In this study the focus is on Scrum since Scrum is an agile approach to management of software development projects [1, 8, 12]. For studies of Scrum, see [10, 14, 20, 22, 26].

In Scrum, the 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" [29].

Scrum and Agile development favour a leadership-and-collaboration style of management where the traditional Project-manager's role is replaced with the Scrum-master's role of a facilitator or coach [1, 8, 12]. The Scrum-master does not organize the team, but the team organizes itself and makes the decisions concerning what to do. The Scrum-master works to remove the impediments of the process, runs and makes decisions in the daily meetings and validates them with the management [29].

In addition to the Scrum master and the Scrum-team, the Product-owner is an important role regarding team leadership. The Product-owner is the official responsible for the project, managing, controlling and making visible the product backlog list. He or she is selected by the Scrum-master, the customer and by the management. He makes the final decisions on the product backlog, participate in estimating the effort for backlog items and turns the issues in the backlog into features to be developed [29].

In a self-organizing team, leadership should be diffused rather than centralized [23]. Therefore we argue that the team leadership in Scrum should be divided among the Product-owner, Scrum-master, and the team. To understand how this should be implemented, it is essential to understand the concept of self-organizing teams and shared leadership.

2.2. Self-organizing teams

Self-organizing, autonomous or self-managed teams have been found to stimulate participation and involvement. An effect of this is increased emotional attachment to the organization, resulting in greater commitment, motivation to perform and desire for responsibility. As a result, employees care more about their work, which may lead to greater creativity and helping behaviour, higher productivity and service quality [13]. Self-management can also directly influence team effectiveness since it brings decision-making authority to the level of operational problems and uncertainties and, thus, increase the speed and accuracy of problem solving [32]. Self-organizing teams are seen as one of the premises for succeeding with innovative projects [16, 31]. Miles et al. [21] suggested that self-management is the "first design principle" for an innovative and collaborative organization.

Takeuchi and Nonaka [31] found that although the members of a self-organized team are largely on their own, they are not uncontrolled. Management should establish enough checkpoints to prevent instability, ambiguity, and tension from turning into chaos. At the same time, managers should avoid the kind of rigid control that impairs creativity and spontaneity. Tata and Prasada [32] found that employees really need to affect managerial decisions for achieving the benefits of a self-managed team. Also there is evidence indicating that the *type* of autonomy may be just as important as the *amount* [17].

In the following, we define autonomy as the degree to which the task provides substantial freedom, independence, and discretion in scheduling the work and in determining the procedures to be used in carrying it out [33]. When discussing autonomy in relation to a self-organizing team, we define autonomy on three levels [22]:

2.2.1. External autonomy. External autonomy is the influence of management and other individuals (outside the team) on the team's activities [16]. Such influence can be deliberate actions from management to limit autonomy, such as requiring the team to make certain decisions regarding work strategies or processes, project goals and resource allocation. While some forms of team-external influence are sometimes beneficial because they provide important feedback, the specific type of team-external influence considered here is detrimental to teamwork in projects e.g. developing software [16].

When decisions are made outside the team, such decisions reflect outside perspectives. If the team members feel that the project reflect largely external demands, then they are less likely to identify with the project [22]. The level of external autonomy can be reduced by the lack of a system for support and people working on several projects because the team will then have problems protecting their resources. [22]. Also the team need to be involved in the initially planning of the project (deadlines, contracts etc) [22].

2.2.2. Internal autonomy. Internal autonomy refers to the degree to which all team members jointly share decision authority, rather than a centralized decision structure where one person (e.g., the team leader) makes all the decisions or a decentralized decision structure where all team members make decisions regarding their work individually and independently of other team members [16]. Not every decision must be made jointly with equal involvement by every team member; rather the team can also delegate authority to individuals or subgroups within the team.

One big threat against internal autonomy is highly specialized skills and corresponding division of work [22]. This can result in a lack of redundancy, which reduce the flexibility and, thus, the internal autonomy of the team. But a paradox arises: how much redundancy should be built into a system?

2.2.3 Individual autonomy. Individual autonomy refers to the amount of freedom and discretion an individual has in carrying out assigned tasks [19, 33]. Langfred [19] describes individuals with high autonomy as individuals who have few rules and procedure constraints, high control over rules and procedures, and high control over the nature and pace of their work. As individual work becomes more independent, and as individuals exert more control over the scheduling and implementation of their own tasks, there will be less interaction between group members. Given reduced contact between group members, it may be a problem to achieve a high level of internal autonomy.

Autonomy on an individual level may conflict with autonomy on a group level, because when individual goals are more important than the team's goals this reduces the internal autonomy [22].

2.3. Shared leadership

For the team to have internal autonomy, they need shared decision authority and shared leadership. When the team and team leaders share leadership, leadership is rotated to the person with the key knowledge, skills, and abilities for the particular issues facing the team at any given moment [19]. While the project manager should maintain the leadership for project management duties (vertical leader), team members should be allowed to lead when they possess the knowledge that needs to be shared or utilized during different phases of the project [11].

It is one thing to say that we need shared leadership in a Scrum team but another thing entirely to develop it effectively. According to Pearce [19] the team leader should be responsible for designing (and re-designing) the team, including clarifying purpose, securing resources, articulating vision, selecting members, and defining team processes. He or she should also manage the boundaries of the team. When applying Scrum, these responsibilities should be the assigned to the Scrum-master, except for the vision which should be assigned to the Product-owner.

Insofar as possible, team leaders should select team members based on their technical, teamwork, and leadership skills. If shared leadership is to be developed, the right people must be on the team, and the teams need to be small [19]. The vertical leader must also articulate how the team will approach its task and function as a team [19]. At the same time, the team leader must articulate trust and confidence in the team

Training is essential when introducing shared leadership. If teams are to succeed at implementing shared leadership, not only do the vertical (or traditional) leaders need training and development but so too do the team members themselves [19].

2.4 Learning and agile leadership

Single-loop and double-loop learning [3] is important in agile leadership. In single-loop learning, one receives feedback in the form of observed effects and then acts on the basis solely of these observations to change and improve the process or causal chain of events that generated them. In double-loop learning, one not only observes the effects of a process or causal chain of events, but also understands the factors that influence the effects. Continuous self-organizing requires a capacity for double-loop learning that allows operating norms and rules to change along with transformation in the wider environment [23].

Therefore, from the literature described earlier in this paper, we argue that there are two major characteristics which have to be in place in order to ensure distributed leadership in agile development, namely good mechanisms for learning, and a sufficient division of external, individual and internal autonomy. The Scrum-master is responsible for establishing this. We discuss these characteristics in section 5.

3. Research design and method

The goal of this research is to understand team leadership in agile software development; hence, it is important to study software development teams in practice. We chose a flexible research design [27], namely a single-case holistic study [36], where we studied team leadership in a project in a small software development department, using multiple sources of data. The company had recently started to use Scrum.

The company introduced Scrum because they wanted to improve their ability to deliver iteratively and on time, increase the quality of software, improve the team-feeling and team communication. The whole development department with 16 developers were introduced to Scrum one year prior to the start of the project described in this study. The project started in February 2007 and was supposed to be delivered by January 2008, but was not finished by February 2008 when our data collection ended. It was the third Scrum project in the company.

The data material for this case study is a subset of an ethnographically informed study.

Table 1: Data sources used in this study.

Source	Comment
Observation	Participant observation and observation of daily stand-up meetings (19), sprint planning meetings (4), sprint reviews (6), and sprint retrospective (1) as well as other meetings. The 72 observations were documented in field notes, which also include pictures.
Interviews	Interviews with three developers and Scrum-master, transcribed.
Documents	Product backlog, sprint backlogs, burn-down charts and recorded data from Team System as well as documented on index cards on wall in the development area.

As for observation, the first and the last authors conducted ethnographic observations (Participant observation [18]) from April 2007 until January 2008. In addition to participant observation, we used interviews and documents as data sources, see Table 1.

Choosing this approach was inspired by Sharp and Robinson [30]. In the beginning of the project the first author worked extensively with the team on GUI-design. After the design was finished the first and third author was involved in workshops on high-level architectural design, estimation and planning meetings. In the two first development sprints, the third author was responsible for running unit tests, and also set up automated testing. From sprint 3 and onwards, the researchers became more outside researchers by only observing and not actively participate in meetings.

The observation sessions lasted from ten minutes to eight hours. We visited the team one to five times a week, in total 72 observations. The semi-structured interviews were performed towards the end of the project.

As for documents, we gathered material from Microsoft Team System, which gave an overview of items in each sprint, estimates and burndown charts for the first eight sprints. However, the data in Team System sometimes deviated from what was reported in the company and also from what could be seen in the wall which will be described in the following.

As for analysis, we organized two data analysis sessions where all three researchers discussed the material from the study. We used time-plots of events and weekly write-ups in addition to reading interview transcripts and plotting the data collected. We organized the material to give a rich description of the case, but focused primarily on the aspects related to team leadership and team self-organizing.

4. Team leadership in an agile project

The goal of the project under study was to redevelop a software package used internally to handle reports from customers. Winter is the low season for the use of this software and the high season begins in March/April with 60-70 simultaneously users. The seasonal constraints gave a relatively narrow time frame for introduction of a new version of the software, and therefore the deadline was initially January 2008. The project started February 2007, and was estimated to use 3000 hours. The old version of the system was developed and maintained by a consultancy company, but the company now wanted to improve the system and to able to maintain it themselves.

Initially the project consisted of two developers (one junior developer recently hired and working 100% on the project, and an experienced developer working 50-100% on the project), a Scrum-master, a Product-owner, and a Project-manager. The Scrum-master was also doing some development and also had the role as department head, which meant he had many other obligations. The Project-manager was cooperating with the Product-owner to define the features, in addition to being responsible for the budget and communicating with the top management. The Product-owner and the Project-manager were situated in another city. Two months after start-up, a third experienced programmer joined the team. He was working 50-100% on the project.

Traditionally this was a plan-driven company relying on highly specialized skills and corresponding division of work. This was now gradually changing after introducing Scrum. One developer said: *We are now working more together, and we share the code, and we know which functionality the others are working on.* Another said: *This is the first time I have experienced that we are editing each others code, and that we all have a good overview. If a person for some reason goes away, this will not stop the project.*

Table 2. Sprints with number of workdays, planned and actually used hours.

Sprint	Work days	Hours planned	Hours used	%
1	22	300	182	61%
2	24	350	245	70%
3	43	327	372	114%
4	22	400	375	94%
5	20	350	260	74%
6	11	254	278	109%
7	9	254	253	100%
8	8	NA	NA	NA

4.1. Pre-game: February to April

This phase started with a kick-off meeting, including five developers in addition to the Scrum-master. Three of these five developers were planned to join later. However, the Scrum-master lost two of these developers to other projects.

The focus of the first phase was analysis, design and planning. The vision for this phase was: Describe the user interfaces which we will need to change. This vision highlighted that the project in essence was a re-engineering project.

During this phase the team extensively used white-board prototyping to create sketches of the GUI. They also produced diagrams for describing workflow, architecture, and the design. This way of working stimulated shared decision-making and then shared leadership. As the Product-owner was not physically present, the team communicated frequently with him using telephone and e-mail. But the team often got stuck because they could not get immediate feedback from him. This resulted in discussions ending without conclusion because the team was lacking vital information.

During some of the workshops in this phase the Scrum-master was frequently interrupted. When he left the workshop the progress usually stopped. One of the developers commented while waiting for the Scrum-master: *A lot of time is wasted because people run in and out of meetings. Team members getting interrupted all the time slows us down.* It seemed like the team was lacking skills when the Scrum-master was absent, and when a person in lead suddenly left the work stopped.

One of the design-workshops was prepared by the newly hired developer. In the beginning of the workshop, it became clear that he was lacking some core competence, the team had to redo his work, and someone else lead the design effort. Since the newly hired developer was the only one working full time on the project, he often had to take responsibility for important work. He later commented: *It is a bit odd that the fresh developer with no experience on vital part of the technology used, is the one ending up doing most of the project. It is strange that experience is not more important when planning and accomplishing a project.*

The problem with resources vanishing from the project was discussed during one lunch: Chris (the developer joining later): *I do not think the management knows how many hours we use on all the changes in other projects, and I'm not sure we get paid either. There has been a crisis in one of my projects lately, and there is continuously fire fighting.* Lindy (working part time on the project): *If you get the responsibility for a project in this company, you are stuck with it forever. It is like a quagmire, and the result is people not wanting to take responsibility. You get really tired when people continuously come and ask you to do something; I get exhausted because it takes my energy. If you do not say "NO" you will use all your time dealing with request from the internal customers. They blame us on delivering bad quality, and we blame them on lacking technical competence.* Mike: *We are too specialised, resulting in only one person being able to solve an issue. So when anything happens you need to leave whatever you are doing at the moment.*

This conversation describes one of the reasons for why people were frequently interrupted, why developers did not want to assume leadership, and as a result the difficulty of applying shared leadership.

The first phase ended with a planning meeting, where the Product-owner presented the first draft of the product backlog in addition to a preliminary release plan. The Scrum-master and the team also presented what they had identified as the project concerns. A project concern [34] is a functional aspect that is considered to be of such importance that they should be treated separately and be specifically visible through the project. The team defined the concerns to be performance and reliability. The concerns should make it easier for the team to prioritize, decide and lead when working on the architecture and implementation. We observed several times that this helped the team identify and dealing with conflicting features.

Another important tool for understanding the project-environment was the work on defining the project risk list and corresponding measures. The risks were identified by the Scrum-master and later discussed with the team to identify measures. This was then presented in this planning meeting.

During the first planning meeting we observed that the two developers were mostly quiet. We also noticed that the discussions were mostly about issues relevant to the Product-owner, Project-manager and Scrum-master. This, as well as a fear of taking leadership ending up in another quagmire, could be reasons for the silence.

4.2. Everyday work: Sprint one to five

From the start of sprint one in April, to the end of sprint five in October, the length of the sprint was around 20 working days (see table 2). Both the Scrum-master and Product-owners seemed reasonably satisfied with the progress during the review-meetings. The third developer started working partly on this project during this phase, but it was clear that he was missing taking part of some of the early discussions. Six months later he said: *most of the decisions were taken and agreed upon a long time before I joined the project.*

On the day of the review- and planning-meeting, the Product-owner and Project-manager usually arrived by airplane early in the morning and usually leaving the same afternoon. In the planning meetings there were a lot of discussions about the intention of each feature and how the system was supposed to be used. It was obvious that this was important for being able to decide on what to do; however these discussions were time-consuming and seemed somehow to exclude the developers. Also these meetings ended in a discussion between the Scrum-master, Product-Owner and Project-manager. These discussions reduced the time available for planning the upcoming sprint, and excluded some team-members from the decision-making. This probably affected the team's chance of really committing to the sprint. Later on we observed another phenomenon affecting the possibility for committing and take leadership. The Scrum-master sometimes suggested who should do what in the planning- and morning-meetings. When we later discussed this topic with the Scrum-master he said: *I probably influence what they do. It difficult to keep your mouth shut when you feel you have some good recommendations to give. It's an old habit, difficult to change. They often agree on my suggestions, but not every time. I think they speak up if they do not agree, and I feel they show initiative regarding their job.*

It was obvious that he felt he got the most competence and should take the leadership on several issues. During the detailed planning the next day, using planning poker, the team often ended up getting stuck in discussions. The team then needed to contact the Product-owner after the meeting to finalize the plan and this complicated the process. Also during this meeting the team found the backlog difficult to use. One developer later commented: *The product-backlog is a mix of very detailed and very general items. Especially all the detailed items are a problem because they also include design, and they often only require 1-2 hours of work. Then you plan to do several of these small items in parallel since they are all related, and then you often get problems. It should be us splitting up the features, not the Product-Owner.*

The team felt they were not able to take the leadership when the product backlog was translated to the sprint backlog. The Product-Owner got good skills and experience in defining traditional requirements, and therefore created a too detailed product backlog.

One recurring phenomenon in this phase was the team never finishing any of the sprint backlogs. There were many unfinished tasks when the sprint ended. In each sprint, the backlog early indicated that it was not possible to complete all tasks. It seemed that everyone knew they could not finish the sprint, but no one suggested any measures to be taken. The Scrum-master commented this at the end of the project: *In the beginning there were so many dependencies to other projects and subcontractors, so we did not know which tasks we could complete during the sprint. It is also about optimization. You should always have the possibility to work on something else if you are stuck. We also started on too many tasks in parallel, and then we lost progress on project-status.* When talking about the effect of this Chris said: *When you during a sprint realized that you would not finish, you do not care if you were 70% or 90% done. It did not matter. These tasks were just moved to the next sprint, but then we also needed to add several other tasks to show progress. We ended up always having a lag from the last sprint but without knowing how big this was. Since we also added too many features we knew we could not finish, and then we did not care if we did not complete all the tasks during a sprint. Some tasks were moved 4 sprints before they were even started.*

During the discussion with Chris it became clear they were not strict about the done criteria, or doing proper testing. This also affected the possibility to finish a sprint. He said: *Another problem was that we classified tasks as finished before they were completed,* *and we knew there were 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.*

Another issue discussed frequently was that resources were taken from the project. Two of the developers were involved in other projects having problems or releases in this period, resulting in the team only getting 50-70% of the planned contribution from the senior developers. The only one working 100% was the fresh developer.

In the daily stand-up the Scrum-master often asked: *Will you work on the project this week?* Lindy: *probably today or tomorrow.* Chris: *probably I will work today.*

In the following stand-up we observed that Chris had not done anything because of several emerging meetings.

During sprint two the team and the product-owner discussed the problem with adding too many features to the sprint backlog and loosing resources. The Scrum-master: *This sprint we did not get as many resources as planned.* Product-Owner: *but this does not answer why you only did half of what we planned to do.* Scrum-Master: *No, we also added too many features. Next time we need to start on the right level*

Our data indicate that this situation did not change much. Leadership was absent regarding this problem.

"The wall" was introduced in the middle of this phase. Every task was then visible, and categorized into "not started", "in progress" and "finished". "The wall" made it visible to everyone that they were working on a lot of tasks in parallel.

One day before a review-meeting we overheard the Scrum-master telling the developers to report more hours than already reported. This report is presented for the Product-owner, and this action seemed to be taken to make the team look as contributing according to the plan. However the Product-owner and Project-Manager seemed to understand that something was not correct regarding the reports. During a planning meeting the Product-owner said: *It is a little bit difficult to know status on the project from how you report.* Project-manager: *and the upper management need a clear status on the project. New projects are coming up. It's my opinion that we need to finish this project first, therefore we need a clear status.*

Also in sprint two, the developers found that a major part of the code needed to be rewritten. One developer said: *These kinds of problems always emerge during a project, but in Scrum they get really visible.* The team used 80 hours on this, 31% of the resources reported in sprint 2.

From our observation it seemed that everyone was aware of the different problems and that this affected the project. However no one took the leadership in this matter, The only official retrospective, conducted during the third sprint:

- testing is not done properly (need a deadline; we are developing to the last minute)
- the resource situation is a problem
- there are several dependencies to other projects

Also in the last planning meeting in this phase, the team discussed to try pair programming on some complex functionality.

The only change done was implementing daily builds to help developers coordinate their development activities. Towards the end of sprint four, it became apparent to everyone that the team was delivering too slowly to be able to meet the final deadline. This was the main issue on the agenda when planning sprint five, but no measures were taken. The team seemed to agree on that if everything went according to their plan in sprint five it would still be possible to finish as planned. However, this was not the case.

In the last planning meeting the Product-owner decided that he wanted to speak with the upper management, to discuss the problem with losing resources. In addition, other measures were discussed:

- isolation 4 days a week
- use overtime
- decrease functionality and extend the deadline.

Also in this meeting there was a big discussion about the importance of the deadline January the 1st. The Product owner argued that it was not possible to move this deadline and that this had been communicated from the beginning of the project. The team did not agree on this. Later we found that our research notes backed the teams' interpretation of this.

After sprint five measures were taken. The Scrum-master later said: *I told the Product-owner and the Project-manager, that we could not possible deliver what was planned before the deadline. This was discussed during the review-meeting. Then, two days later, after speaking with the management, we found a solution to the problem. We agreed on postponing the deadline, keeping the functionality, and then isolate the team.*

4.3. Emergency Scrum: Sprint six to nine

The conclusion from the previous phase was to shorten the sprints to help increasing focus and to make it easier to isolate the team. The Scrum-master decided that the team was not even allowed to receive phone calls, while in isolation. This was broadcasted generally in the company. The Scrum-master now collocated with the team The team agreed on using some overtime.

Isolation of the team was not 100% effective; especially when the Scrum-master was absent. One developer said: *The isolation works how it should do, but we end up in a dilemma. Like today, it's crazy in the other project, where the customers are starting to use the system this week. And I'm the only one who can fix problems. Then I just need to help if they are having problems. I do not like to decide which project will not meet its deadlines, because that is what it is all about. But now the Scrum-master has decided what to prioritize. However I feel I need to do it anyway; during the day or before I go home. There is no time allocated to this kind of support, and there are several people expecting me to solve the problems.*

When we discussed this problem later on the Scrum-master said: *I know that some of them are feeling the pressure from other projects, and that they do other things than we agreed upon. If someone is calling them, this can result in a developer using time on something else, even If they are in full isolation. I do not think it's their fault; it's my responsibility to protect them. Today someone called one of the developers. I told him that he could use 3 minutes. If he had not stopped the conversation, I would have pulled out his phone plug.*

The developer felt there was a contradiction in terms of the official policy of the company and the reality. However the isolation was seen as a big improvement by the team. The isolation was most effective when the Scrum-master was in the area, but the developers also felt it easier to say "no, I'm busy" when asked about something.

The Scrum-master commented: *I can see that the team is working harder and more focused. Earlier, they seemed relaxed the first week of the sprint. Now they are relaxing the first day, but then they start to feel the pressure. But I do not know how long we can do this, how long the effect will last.*

It seemed easier for the developers to focus on the project and they said they felt more dedicated. However during the last sprint one developer commented while writing code: *We need to talk to the Product-owner, but he is on a vacation. I guess they do not really care too much, and this affects us as well.*

The situation regarding making functionality finishes was not changed much. Still they were writing new code on the day of the review meeting, and in this meeting the developers did not know which part of the code would work fine and what could fail. It seemed they were trying to present as much as possible to impress the Product-owner. Also some features were not demonstrated during one of the review meeting because the person who developed the features was sick. It's obvious that even though they are working together and knew each others code, they did not know enough to demonstrate what others were doing.

This was confirmed in sprint 8, when one of the developers had an accident resulting in a sick leave for the rest of the year. No one continued to do his task, partly because they were not seen as critical.

5. Discussion

After this broad description of the project, we return to discuss it in light of our research question:

What are the main challenges in establishing leadership in an agile software development project?

Section 2 concluded with the major characteristics which we argued have to be in place in order to ensure a distributed leadership in agile development and let the team adapt to changing circumstances, namely a sufficient distribution of external, individual and internal autonomy, and good mechanisms for learning. The leadership should be rotated to the person with the key knowledge, skills, and abilities for the particular issues facing the team at any given moment. The team should be manned with the right people. The team leader should be responsible for designing (and re-designing) the team, and also manage the boundaries of the team.

It seemed clear in the project that the team leader did not manage to change the company culture and protect the team as much as he should do. He lost resources and the team was constantly interrupted. However this situation improved in the last phase. He also did not manage to design the team as originally planned, resulting in lacking competence, which again made it difficult to rotate leadership. He often needed to take leadership regarding design and architecture, but when he was absent the work sometimes stopped.

Introducing agile methods like Scrum takes time, because it requires the whole organization to change. It requires transformational leadership [37]. This leadership is about empowering subordinates to participate in the process of transforming the organization. And without focus on transformational leadership it is difficult for the project to change and improve since they are heavily integrated with the rest of the organization. Transformational leadership was mostly missing in this project. Even though applying most of the Scrum practices and experiencing an improved development processes, it was clear that project-external issues were reducing both external and internal autonomy.

As for mechanisms for learning, we see that the major change in the development process, the change from four-week to two-week iterations, was improving the resource-situation. However this change came late (the problems were reported already in the first sprint). Using project retrospectives more might have led to this discussion coming earlier; however the scrum-master did not take enough leadership on this issue. Also, the fact that the team and the Scrum-master wanted to present more features as finished than was the case, and to report more hours than actually used. This is defined as “impression management” by Morgan [23], and is a barrier to double-loop learning.

Another issue observed in the project, was that work progress was sometimes slow when a person had to leave a meeting or when the product owner was not available. This could indicate a lack of shared mental models on what was to be done in the project and lacking core competence, and might be an issue that could have been improved if the sprint planning meetings had been more focused on planning development work. Lacking competence makes shared leadership difficult.

As for external autonomy, we see from the description in the previous section that the frequent interruption from other projects resulted in less resources than allocated was reported in four of the five first sprints. Also, when people suddenly had to leave a workshop to handle a problem somewhere else, this delayed everyday work. When the team was isolated, this changed.

A threat to the internal autonomy of the development team was that the planning meetings were not sufficiently used for short-term planning in the project. This might be another cause why the sprints in the first phase had many items which had to be transferred to the next sprint, which again led to little commitment on the backlog from the team.

The fact that the team never finished any of the sprint-backlogs, made it difficult for the team to really commit to the sprint. They knew they could not finish even before starting. That the Scrum master was assigned work, and that developers often had to do other work than planned might also have lead to lowered commitment from the developers. They were experiencing symbolic self-organising [32].

In hindsight, it is easy to point at the planning meetings, concluding that these meetings should have been better prepared by all parties.

Since both the internal and external autonomy was reduced, in addition to strong individual autonomy this affected the ability to be fully self-organized, which again weakened the distributed leadership.

6. Conclusion and future work

This paper presented data from an eleven month ethnographically informed case study of professional developers in a Scrum team. We have focused the description of the project on challenges when establishing leadership in an agile software development project. From the described project, we identified two main challenges:

First, we see that mechanisms to enable efficient learning were not in place in the project. We observed that the problem with loosing resourced was improved when shortening the sprints. However the decision to isolate the project team was taken late, although there were clear signs of problems at the sprint review and planning meetings. The project lacked a mechanism to ensure double-loop learning, and because the team tried to impress the Product-owner this made double-loop learning even harder.

Second, establishing right levels of autonomy for project, externally, internally and individually was challenging as we observed from the problems related to the “quagmire” and problems with short-term planning. The fear of the “quagmire” also resulted in no one few taking leadership. The missing double-loop learning also reduced the possibility to improve self-organising.

For practitioners, we think this paper illustrates some of the difficulties in establishing shared leadership in agile project teams. This is an issue which is not well-described in the agile literature.

Further work in this direction should focus on investigating other barriers with leadership in agile development, for example related to coordination. In particular it would be interesting to study more mature agile development teams to see if the problems identified here have been overcome.

7. References

[1] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, “Agile Software Development Methods: Review and Analysis,” VTT Technical report 2002,

[2] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, “New Directions on Agile Methods: A Comparative Analysis,” Proceedings of the 25th International Conference on Software Engineering (ICSE'03), IEEE Press, 2003.

[3] C. Argyris and D. A. Schon, *On Organizational Learning Ii: Theory, Method and Practise*. Reading, MA: Addison Wesley, 1996,

[4] S. Augustine, B. Payne, F. Sencindiver, and S. Woodcock, “Agile Project Management: Steering from the Edges,” *Communications of the ACM*, no. 12, vol. 48, pp. 85-89, 2005.

[5] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Chage (2nd Ed)*: Addison-Wesley, 2004, ISBN 978-0321278654.
[6] B. W. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*: Addison-Wesley 2003,
[7] A. Cockburn and J. Highsmith, "Agile Software Development: The People Factor," *Computer*, no. 11, vol. 34, pp. 131-133, 2001.
[8] D. Cohen, M. Lindvall, and P. Costa, "An Introduction to Agile Methods," in *Advances in Computers, Advances in Software Engineering*, vol. 62, M. V. Zelkowitz, Ed. Amsterdam: Elsevier, 2004.
[9] M. Cohn and D. Ford, "Introducing an Agile Process to an Organization [Software Development]," *Computer*, no. 6, vol. 36, pp. 74-78, 2003.
[10] T. Dingsøyr, G. K. Hanssen, T. Dybå, G. Anker, and J. O. Nygaard, "Developing Software with Scrum in a Small Cross-Organizational Project," in *13th European Conference on Software Process Improvement (Eurospi)*, *Lecture Notes in Computer Science*, vol. 4257, I. Richardson, P. Runeson, and R. Messnarz, Eds. Joensuu, Finland: Springer Verlag, 2006, pp. 5-15.
[11] T. Dybå and T. Dingsøyr, "Empirical Studies of Agile Software Development: A Systematic Review," *Information and Software Technology*, vol 50, pp. 833-859, 2008.
[12] J. Erickson, K. Lyytinen, and K. Siau, "Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research," *Journal of Database Management*, no. 4, vol. 16, pp. 88 - 100, 2005.
[13] M. Fenton-O'Creevy, "Employee Involvement and the Middle Manager: Evidence from a Survey of Organizations," *Journal of Organizational Behavior*, no. 1, vol. 19, pp. 67-84, 1998.
[14] B. Fitzgerald, G. Hartnett, and K. Conboy, "Customizing Agile Methods to Software Practices at Intel Shannon," *European Journal of Information Systems*, no. 2, vol. 15, pp. 200-213, 2006.
[15] R. A. Guzzo and M. W. Dickson, "Teams in Organizations: Recent Research on Performance and Effectiveness," *Annual Review of Psychology*, vol. 47, pp. 307-338, 1996.
[16] M. Hoegl and K. P. Parboteeah, "Autonomy and Teamwork in Innovative Projects," *Human Resource Management*, no. 1, vol. 45, pp. 67-79, 2006.
[17] B. D. Janz, J. A. Colquitt, and R. A. Noe, "Knowledge Worker Team Effectiveness: The Role of Autonomy, Interdependence, Team Development, and Contextual Support Variables," *Personnel Psychology*, no. 4, vol. 50, pp. 877-904, 1997.
[18] D. L. Jorgensen, *Participant Observation: A Methodology for Human Studies*. Thousands Oak, California: Sage publications, 1989,
[19] C. W. Langfred, "The Paradox of Self-Management: Individual and Group Autonomy in Work Groups," *Journal of Organizational Behavior*, no. 5, vol. 21, pp. 563-585, 2000.
[20] C. Mann and F. Maurer, "A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction," in *Proceedings of Agile 2005*. Denver: IEEE Press, 2005.
[21] R. E. Miles, C. C. Snow, and G. Miles, "Thefuture.Org," *Long Range Planning*, no. 3, vol. 33, pp. 300-321, 2000.
[22] N. B. Moe, T. Dingsøyr, and T. Dybå, "Understanding Self-Organizing Teams in Agile Software Development," Proceedings of the Australian Software Engineering Conference (ASWEC), Perth, Australia, pp. 76-85, 2008.
[23] G. Morgan, *Images of Organizations*. Thousand Oaks, CA: SAGE publications, 2006.
[24] S. Nerur, R. Mahapatra, and G. Mangalaraj, "Challenges of Migrating to Agile Methodologies," *Communications of the ACM*, no. 5, vol. 48, pp. 72, 2005.
[25] S. Nerur and 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*, no. 3, vol. 50, pp. 79-83, 2007.
[26] L. Rising and N. S. Janoff, "The Scrum Software Development Process for Small Teams," *IEEE Software*, no. 4, vol. 17, pp. 26-+, 2000.
[27] C. Robson, *Real World Research*. Oxford: Blackwell, 2002,
[28] E. Salas, D. E. Sims, and C. S. Burke, "Is There A "Big Five" In Teamwork?," *Small Group Research*, no. 5, vol. 36, pp. 555-599, 2005.
[29] K. Schwaber and Beedle, *Agile Software Development with Scrum*: Upper Saddle River: Prentice Hall, 2001,
[30] H. Sharp and H. Robinson, "An Ethnographic Study of Xp Practice," *Empirical Software Engineering*, no. 4, vol. 9, pp. 353-375, 2004.
[31] H. Takeuchi and I. Nonaka, "The New New Product Development Game," *Harvard Business Review 64: 137-146* 1986.
[32] J. Tata and S. Prasad, "Team Self-Management, Organizational Structure, and Judgments of Team Effectiveness," *Journal of Managerial Issues*, no. Issue 2, vol. Vol. 16 pp. p248 - 265, 2004.
[33] H. van Mierlo, "Individual Autonomy in Work Teams: The Role of Team Autonomy, Self-Efficacy, and Social Support," *European Journal of Work and Organizational Psychology*, no. 3, vol. 15, pp. 281, 2006.
[34] S. Walderhaug, E. Stav, S. L. Tomassen, L. Røstad, and N. B. Moe, "Mafiia – an Architectural Description Framework: Experience from the Health Care Domain," in *Interoperability of Enterprise Software and Applications*, D. Konstantas, J.-P. Bourrières, M. Léonard, and N. Boudjlida, Eds. Geneva, Switzerland: Springer Verlag, 2005, pp. 43-54.
[35] G. Walsham, "Doing Interpretive Research," *European Journal of Information Systems*, no. 3, vol. 15, pp. 320-330, 2006.
[36] R. K. Yin, *Case Study Research: Design and Methods*, 3rd ed: Sage Publications, 2003, ISBN 0-7619-2553-8.
[37] G. Yukl, "Managerial Leadership - a Review of Theory and Research," *Journal of Management*, no. 2, vol. 15, pp. 251-289, 1989.