Understanding Self-organizing Teams in Agile Software Development

Nils Brede Moe, Torgeir Dingsøyr, Tore Dybå
SINTEF ICT, Norway
(nilsm|torgeird|tored)@sintef.no

Abstract

Traditional software teams consist of independently focused self-managing professionals with high individual but low team autonomy. A challenge with introducing agile software development is that it requires a high level of both individual and team autonomy. This paper studies the barriers with introducing self-organizing teams in agile software development and presents data from a seven month ethnographic study of professional developers in a Scrum team. We found the most important barrier to be the highly specialized skills of the developers and the corresponding division of work. In addition we found a lack of system for team support, and reduced external autonomy to be important barriers for introducing self-organizing teams. These findings have implications for software development managers and practitioners.

1. Introduction

Agile software development [9] 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 [24]. Further, agile development relies on people and their creativity rather than on processes [5]. Leadership and collaboration, informal communication and an organic (flexible and participative, encouraging cooperative social action) organizational form are other characteristics of agile software development [25].

Practitioners who have developed agile methods have formulated the agile manifesto[1], a statement that expresses a set of basic principles and rules: (1) individuals and interactions over processes and tools; (2) working software over comprehensive documentation; (3) customer collaboration over contract negotiation; and (4) responding to change over following a plan.

Cohn and Ford [7] 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 [25]. 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 [4]. Both practitioners and academics have criticized agile development methods, e.g., the lack of scientific support for many of the claims made by the agile community.

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 [14].

Agile development relies on teamwork, as opposed to individual role assignment that characterizes plan-driven development [25]. A team following a plan-driven model often consists of independently focused self-managing professionals. A transition from high individual and low group autonomy to a high level of individual *and* group autonomy is probably one of the biggest challenges when introducing change-driven development based on self-organizing teams [24].

Motivated by the importance of both teams in software development and self-organizing teams in agile development, our research question is:

What are the barriers with introducing self-organizing teams in agile software development?

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 describe self-organizing teams and why such teams are important in

[1] www.agilemanifesto.org

1530-0803/08 $25.00 © 2008 IEEE
DOI 10.1109/ASWEC.2008.28

agile development. 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 the barriers with introducing self-organizing teams in agile software development. We present the results according to the findings in the literature. We discuss our findings in Section 5. Section 6 concludes and presents important barrier for achieving self-organization.

2. Background

In this section we present background information on agile development, and we use the literature to describe self-organizing teams and why such teams are important in agile development.

2.1. Agile development and Scrum

Agile software development comprises a number of practices and methods [1, 6, 10]. Among the most known and adopted agile methods are Extreme Programming (XP) [3] and Scrum [29]. XP focuses primarily on the implementation of software, while Scrum focuses on agile project management [2]. In this study the focus is on Scrum as Scrum encourages self-organizing teams.

Scrum focuses on project management in situations where it is difficult to plan ahead, with mechanisms for "empirical process control"; where feedback loops is the core element [29]. Software is developed by a self-organizing team in increments (called "sprints"), starting with planning and ending with a review. Features to be implemented in the system are registered in a backlog, and a product owner decides which backlog items should be developed in the following sprint. Coordination between team members are done in daily stand-up meetings. One team member, the scrum master, is in charge of solving problems that stops the team from working effectively. For studies of Scrum, see [8, 12, 20, 27].

The 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" [29].

The name "Scrum" was inspired by an article characterizing new product development teams in Japan, by Takeuchi and Nonaka [31], where "self-organizing project teams" was one of six key characteristics.

The Scrum master is often described as a coach or facilitator. He or she does not organize the team; 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]. Scrum and Agile development favor 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 coordinator [1, 6, 10].

Although agile development is a relatively new approach, this is not the case with self-management. Research in this area has been around since Trist's examination of self-regulated coal miners in the 1950s [33].

2.2. Self-organizing teams

We use the label "self-organizing" teams as a synonym for "autonomous teams" and for "empowered teams". Guzzo and Dickson [14] describe such teams as teams of employees who typically perform highly related or interdependent jobs, who are identified and identifiable as a social unit in an organization, and who are given significant authority and responsibility for many aspects of their work, such as planning, scheduling, assigning tasks to members, and making decisions with economic consequences.

Autonomous teams stimulate participation and involvement, and 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 behavior, higher productivity and service quality [11]. 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 [15, 31]. Kirkman and Rosen [18], for example, studied 111 teams in four organizations, and found that empowered teams (autonomy central in these teams) were more productive and proactive than less empowered teams. Miles et al. [22] 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.

However, there is substantial variance in research findings regarding the consequences of such teams on such measures as productivity, turnover, and attitudes [14, 32]. Tata and Prasada [32] found that employees really need to affect managerial decisions for achieving the benefits of a self-managed team. It is important that the team do not experience symbolic self-management. This variance may indicate that the effects of such teams are highly situationally dependent. Also there is evidence indicating that the *type* of autonomy may be just as important as the *amount* [16].

2.3. Autonomy on different levels

Different types of autonomy include autonomy over product, people and planning decisions [16]. The team will also have autonomy if it has the authority to set its own goals (goal-defining autonomy), its own social identity and boundaries to other social systems (structural autonomy), resources to fulfill its tasks and survive until the task is completed (resource autonomy), and freedom for self-organizing the behavior of its members (social autonomy) [13]. Langfred [19] argues that a software team probably needs group autonomy and individual autonomy, but that autonomy on an individual level may conflict with autonomy on a group level, producing countervailing influence on the cohesiveness and, indirectly, effectiveness of the team. Uhl-Bieh and Graen [34] found that if a cross-functional team is staffed with independently focused self-managing professionals, this may harm the organization and decrease the effectiveness. Langfred [19] argues that group autonomy does not relate to internal processes of the group. For example, if a group is granted autonomy over group action in the organization, the decision on how to use such autonomy might be arrived at in a number of different ways (e.g. consensus, vote, and coalition).

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 [35]. When discussing autonomy in relation to a self-organizing team, we will discuss how autonomous the team is with respect to: (1) the rest of the organization – external autonomy; (2) internal organization of the work in the group – internal autonomy; and finally (3) how free individuals are to organize their work – individual autonomy.

2.3.1 External autonomy. External autonomy is defined by Hoegl and Parboteeah [15] as the influence of management and other individuals (outside the team) on the team's activities. 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 to help project completion or encourage creativity within the team by discouraging groupthink, Hoegl and Parboteeah [15] argue that the specific type of team-external influence considered here is detrimental to teamwork in projects e.g. developing software.

2.3.2. 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 [15]. 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.

A group may have considerable discretion in deciding what group tasks to perform and how to carry them out, but individual members within the group may have very little control over their jobs.

2.3.3 Individual autonomy. Individual autonomy refers to the amount of freedom and discretion an individual has in carrying out assigned tasks [19, 35]. Langfred 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. Individual autonomy influences group cohesiveness by the reduction in interpersonal interaction that associates with individual autonomy. 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 [19].

3. Research design and method

The goal of this research is to understand the barriers with introducing self-organizing teams in agile

software development; hence, it is important to study software development teams in practice.

We report on a single-case holistic study [36], in which we studied one phenomenon in one project.

3.1. Study context

This study was done in the context of a larger action research program, where several companies have introduced elements from agile development in response to identified problems. For the company where the projects in this study are taken from, Scrum was introduced because they wanted to improve their ability to deliver iteratively and on time, increase the quality, improve the team-feeling and team communication. The whole development department with 16 developers were introduced to Scrum at the same time. This was the first Scrum project in the company.

The goal of the project studied was to develop a plan and coordination system for owners of cables (e.g., electricity, fibre) and pipes (water, sewer). The project produced a combination of textual user-interfaces and map-functionality. The company will also be responsible for maintenance and support after final installation.

4000 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, but was situated in another city. He acted as a representative for the customer who was the local government of a Norwegian city. The project kick-off was in May 2006, first installation was in October and final installation was planned to be January 2007. Prior to the kick-off some initial architectural work was done, and some coding activities had started. The project used .Net technology.

3.2. Data Sources and analysis

The two first authors conducted ethnographic observations (Participant observation [17]) from May 2006 until January 2007. During the observation periods, we visited the team once or twice a week the first weeks, in total 45 observations. In each observation session, we participated in daily scrum meetings, planning meetings, retrospective, review meetings, and observations of developers working. These sessions lasted from 10 minutes to 8 hours. We sat behind the developers, taking notes on their dialogues, interactions and activities. The dialogues were transcribed and integrated with our notes to produce a detailed record of each session.

All data from this study was imported into a tool for analyzing qualitative data, Nvivo (a tool for analyzing qualitative data, www.qsrinternational.com). The first author coded the material, using open coding or "postform" coding [30], looking for material related to individual, internal and external autonomy.

4. Autonomy in an agile project

Scrum was introduced through a two day workshop in June 2006. At the beginning of the project, the team was partly situated in an open office. Those situated in traditionally offices moved out into the open office after a few months.

The daily meeting in Scrum is supposed to last maximum 15 minutes, and in this project the daily scrum meeting lasted from 10-20 minutes. The team members responded to the following: 1) What did you do since the last Scrum Meeting? 2) Do you foresee any obstacles? 3) What will you do before next meeting? The length of each of the five sprints was one month – except for a 2 month summer-sprint. Each sprint started with a planning meeting and ended with a review meeting, where the developed functionality was presented. Shortly after the review meeting, the next planning meetings were conducted. Several retrospective meetings were also organized, but since the product owner was situated in another city, he did not attend all of these.

The system was integrated with a map-module delivered from a subcontractor one month before the first installation. This integration did not work well because of trouble with the response-time. During the first integration period you could find empty pizza-boxes every morning in the office. Because of the problems with the response time the final installation was postponed until October 2007 and the project used a considerable amount of extra hours. It was a problem to allocate new resources for this extension since people were already allocated on new projects. Three developers continued on the project in 2007.

4.1. Individual autonomy

The project was divided into modules, and each person worked on the module where he or she was seen as a specialist (e.g. user interface, map-interfaces or databases). This way of dividing work was typical for this company, and there was a common understanding among managers and developers that this was the most efficient way of working.

[Talking to the Scrum-master two months after Scrum was introduced]

The scrum-master: "Let the person that knows most about the task solve it! It will take too many resources if several persons are working on the same module, and there is no time for overlapping work in this project. The tasks are delegated and solved the best possible way today. Maybe we can do more overlapping in the next project".

Because of the work being divided into modules, a developer typically created his own "plan" for this module, often without discussing it with the team. One developer even implemented features for future projects. This was discussed on a daily stand-up in the second sprint.

Developer: We got our own vision on what to do, the sprint plan is only a guide, a check list. 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.

Development problems related to their module were often seen as personal, and therefore not reported to the group. This was discussed in a retrospective four weeks before the first delivery.

Developer: 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 time.

The level of individual autonomy was high in this project, since everyone worked on their own module, created their own plan for this module and perceived problems related to their module as personal. Each developer had a great amount of freedom and discretion when carrying out assigned tasks. This was also the case for the two developers cooperating on the user-interface. They found it difficult to discuss or work on the other colleague's code when others for some reason were prevented from working on the project.

4.2. Internal autonomy

The team-members described themselves as a highly motivated team working together to achieve a common goal, and the Scrum-master perceived the team as enthusiastic.

Scrum-master: *"you can feel the enthusiasm in the team"*.

However, most team-members came late one or several times to meetings and some did not inform the rest of the team when they did not show up. It seemed that some did not feel 100% committed to the project meetings. One reason was that some of the developers perceived these meetings as a waste of time. The high individual autonomy resulted in developers focusing on their own module; and not knowing what others were doing. In the third retrospective this was discussed.

Developer: "we distribute the tasks so everyone works on his own module. I do not know what Chris and Mike are doing."

This again resulted in some of the developer's not paying attention to what others were saying during the daily Scrum-meeting. As individuals are independent, and have more control over their schedule and implementation of their tasks, there will be less interaction between the group members [19]. In this project this resulted in people not listening or talking to each other during the meeting. This problem was discussed in the retrospective four weeks before the first delivery.

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

The developers focusing on their own "plan" resulted in a lack of trust from the Scrum-master. The developer implementing features for the next project was not fully trusted anymore, resulting in the Scrum-master started to give him instructions on what to work with. This reduced both the individual autonomy of the developer and the internal autonomy of the group.

Another reason for developers not paying full attention to the daily meetings was because of the topics being discussed. The topics were often not related to the features being implemented. The focus of the meetings was often more general project problems. One of the early daily meetings was organized without the Scrum-master, and in this meeting the developers felt they could focus differently.

One said: Today was probably our best daily meeting. Yesterday reflects how we typically organize the meeting. It was mostly about the contract and the customer. It was probably useful for the Scrum-master but not for us. He is also more interested in what we have done than in what we are going to do.
Another said: It's not possible to talk about what you want to talk about. You only report on what you have done, and then you are done.

The Scrum-master probably felt it was important to inform about the customer and the contract since a lot had happened lately, and he felt this was important for

understanding the goal of the project. But since several of the team members perceived this as of little relevance for their present work, they felt the meeting was a waste of time. The focus on reporting on what was done since the last time made it difficult for the developers to talk about what they were going to do. A discussion about future work and future solutions would probably increase the internal autonomy. The project participants did not give feedback on the problem regarding the structure of the meeting before the last retrospective.

Since problems were seen as personal and therefore seldom discussed in the meetings, it was difficult to have an equal influence over the solution of important problems. In the third retrospective this was discussed.

Developer: "if we had more information about the problems, then we would discuss them earlier. The problems are reported too late"

The Scrum-master became aware of problems not being reported. Therefore, when he discovered a problem, he tried to ensure that it got a lot of focus, was thoroughly discussed and quickly solved. But then the developers felt the scrum-master was overreacting and creating a lot of noise. Therefore they reported even fewer problems. This again decreased the internal autonomy.

The focus on individual modules was probably one of the reasons that several developers felt they were missing the total overview of the project. Without a clear understanding of the system being developed, planning was difficult. The monthly planning meetings turned out to be discussions between the Scrum-master, product owner and the developer that was going to do the work. Sometimes the Scrum master even proposed estimates. The developers probably felt they did not need to discuss the total plan in detail since they later would create their own plan for their module. Anyway, a lack of thorough discussions was probably one of the reasons for important tasks being identified during and in the end of each sprint instead of at the beginning of each sprint. This again reduced the validity of the common backlog, making the developers focus more on their own plan. Since the planning had weaknesses and none of the developers felt they had the total overview, this probably was one of the reasons for design-problems discovered later.

Little overlapping work resulted in developers not wanting to edit others code and difficulties with doing others work when people were not available to the project (e.g. sick, vacation, travel). This caused delays, but the developers did not seem to bother. They probably felt there was nothing they could do about it.

The product owner was the person responsible for communicating the features this system was going to provide. But he lived in another city, was always very busy, and became sick in a critical phase. There was a lot of communication between the developers and the Product Owner but according to the developers it should have been more, to help them better understand how the system was going to be used. This again reduced the internal autonomy, and with a reduced internal autonomy the team did probably not communicate their needs well enough. Lacking a good communication process with the Product Owner was probably one of the main reasons for the backlog missing important tasks, which reduced the quality of the team's common plan and reduced the internal autonomy.

When the team understood that they would have problems with the first deliverable because of problems with integration, they canceled several of the daily meetings, reducing communication and internal autonomy even more.

The team members did not have an equal influence over project decisions since they were focusing on solving their own tasks in their own module, and did not pay to much attention to issues not directly relevant to their module. Their decision structure was therefore partly decentralized.

4.3. External autonomy

In earlier projects the developers felt there was too much noise around the projects. Supporting several existing systems and participating on several project teams was problematic. The developers felt they used too many hours on tasks not being relevant for their primary project. After introducing Scrum, the team-members felt they where more protected against external noise and other projects than before, which increased their external autonomy.

The problem with integration was partly a result of the management believing too much in new technology, and not understanding the complexity of the system being developed. When signing the contracts with the customer and sub-contractor, the management accepted the sub-contractor to deliver only one month before the first installation. Since the contracts were signed before the project started, the team could only accept these constrains. However the team should probably have identified the potential problems regarding the first installation, but the team was lacking core competence on the technology used (related to maps) in the module that was to be integrated. The experts on this technology worked on other projects and they were not involved in this

project. The management also chose a strategy where the sub-contractor was not involved in the development process, and little communication with the sub-contractor made it more difficult for the team to detect the potential problems.

The company was missing a formal system for support, which reduced the external autonomy of the project. If a developer was responsible for a module or system being developed, this developer would always be responsible for supporting the same solution. And the developer would never know when the customer would call, and interrupt the work. If you got involved in a project you were stuck with it forever. The developers described this as the "quagmire".

[The developer is picking up a coffee at the coffee machine, and walking back to his computer when an internal customer passes]
Internal customer*: Hi, you need to fix the issue I sent you an e-mail about*
Developer *[starting to walk back]: I do not have the time now*
Internal customer *[walking after him]: This is really important and it must be solved now*
Developer *[walking faster and obviously stressed]: I'm working on something very important now, and I do not have the time now*
Internal customer *[getting irritated, but stops]: it need to be solved now*
[The developer gets back to his computer and talks about what just happened]
Developer*: This is the problem with working here. If you ever get involved in developing a system you get stuck in the quagmire. And for every new project you are on, it get worse. We should build a wall of plexiglass around the developers. When someone want us to do anything they could come over and ring a bell, and leave a note. Later we could look on the notes and decide what to do [laughing].*

The problem with external people "stealing" resources directly from the project decreased the external autonomy and affected the individual autonomy. It also resulted in developers not wanting to take responsibility. If you became responsible for a solution you got lost in the quagmire. The problem with developers not taking responsibility of the solution reduced the internal autonomy even more since individual goals were more important team goal's.

People were allocated to several projects, so when there were problems in other projects this resulted in less progress in this project, and since no one else could work on their modules the progress is delayed.

Compared to previous projects in the company, they had a higher external autonomy, but this level could have been improved.

5. The Scrum team and self-organization

If we go back to the description of Scrum as the development method that inspired the team in the case described, we see that a Scrum team should be given "full authority to do whatever it decides is necessary to achieve the goal" [29]. The Scrum master should work as a coach and facilitator in order to make the team organize itself, and in order to remove impediments of the process.

From the description of the case, we see that the project was not using Scrum as it is intended to be used. A major difference is on what the product was like after a sprint. Often, testing tasks were given lower priority, so that the product was not in a potential shippable state, and features had to be worked on further in the next sprint.

As for self-organization, we see that the project in our case had a high level of individual autonomy, while they were less autonomous as a team (internal autonomy). Compared to previous projects in the company, they had a relatively high external autonomy but the level could be higher. The main difference between a team as described in the scrum process and the team in our case is the lack of internal autonomy. We now proceed to discuss our case in light of our research question:

What are the barriers with introducing self-organizing teams in agile software development?

The company's focus on division of work according to specialization was clearly seen by the Scrum master's statements that "it will take too many resources if several persons are working on the same module," "there is no time for overlapping work in this project" and "the tasks are delegated and solved the best possible way today." From a traditional bureaucratic viewpoint, eliminating redundancy is clearly desirable and efficient.

While this may be efficient for relatively simple and clearly specified tasks, it is not necessarily so for more complex tasks that are to be solved by a team. Indeed, in the literature on self-organizing teams, we find that redundancy has been identified as an important prerequisite for self-organization [23, 26]. In particular, the "redundancy of functions", in which team members acquire multiple skills so that they are able to perform each other's jobs and substitute as the need arises, is essential for creating flexibility. The lack of such redundancy, and consequently, flexibility, is probably one of the most important reasons for several of the problems experienced by the case company.

In the literature of teamwork the concept of redundancy is often described as backup behavior. Backup behavior is an important mechanism that affects the team's capability to adapt to changing situations and environments [28]. If backup is to occur effectively, teammates need to be informed of each others' work in order to identify what type of assistance is required at a particular time [21]. Marks [21] identify three means of providing such backup behavior: (1) providing a teammate verbal feedback or coaching, (2) assisting a teammate behaviorally in carrying out a task, or (3) to complete a task for the team member when an overload is detected. All these means were mostly absent in this project, which reduced the level of internal autonomy.

Thus the highly specialized skills and corresponding division of work resulted in a lack of redundancy. This lack of redundancy reduced the flexibility and, thus, the internal autonomy of the team, and was therefore the most important barrier for self-organizing. The highly specialized skills and division of work also resulted in high individual autonomy, which often is described as important for individuals to become more motivated and satisfied with their job [19].

Team orientation [28], often described as a team's goals over individual goals, is important for every team; particularly for an autonomous team. In this project the individual goals were more important than the team's goals, which was a result of both high specialization and the problem with the "quagmire" (missing a system for support). The quagmire resulted in people not taking responsibility because their personal goals became more important than the team's goals. Therefore the quagmire can also be seen as a barrier for internal autonomy.

If the team has a high degree of autonomy over project decisions (external autonomy), team members are believed to rely upon themselves for task decisions, which will likely increase the sharing of information as well as the coordination of task activities horizontally within the team [15]. It is also likely that 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. Such lower identification is likely to decrease team members' willingness to fully contribute their knowledge to problem-solving processes. In our case, the level of external autonomy was reduced by the lack of a system for support, people working on several projects, and how the management had planned the project (deadlines, contracts etc). In turn, this may also have reduced the level of internal autonomy. Hence the reduced level of external autonomy was an important barrier of self-organizing. Even though company culture and practice was the reason of reduced external autonomy, the Scrum-master probably should have tried to protect his team better from the external noise.

6. Conclusion and future work

This paper presented data from a seven month ethnographic study of professional developers in a Scrum team. We found that highly specialized skills and corresponding division of work was the most important barrier for achieving self-organization. However, this also had a positive effect since it gave high individual autonomy, but in this case it made it difficult for the team to self-organize. In addition we found a lack of system for support, and reduced external autonomy to be other important barriers for introducing self-organizing teams.

Before starting with Scrum and in the second retrospective (one month before the first deliverable) the problem with "highly specialized skills and corresponding division of work" was thoroughly discussed, and the team decided to focus on this challenge. However they did not manage to radically change their working practice during the project. Changing the way of working is difficult, and when it involves a transition from specialized skills to redundancy of functions, it requires a reorientation not only by the developers but also by the management. This change takes time and resources, and it must be solved to be able to succeed with Scrum.

This study highlights several barriers with introducing self-organizing teams in agile development. To succeed with introducing agile development it's important to be aware of these barriers. Accordingly, further work should focus on investigating other barriers with introducing agile development.

7. Acknowledgment

We appreciate the input received from the project participants of the investigated company. This research is supported by the Research Council of Norway under Grant 174390/I40

8. References

[1] Abrahamsson, P., Salo, O., Ronkainen, J., and Warsta, J., *Agile software development methods: Review and analysis*. 2002, VTT Technical report. p. 107.

[2] Abrahamsson, P., Warsta, J., Siponen, M.T., and Ronkainen, J. *New directions on agile methods: a comparative analysis*. 2003.

[3] Beck, K. and Andres, C., *Extreme Programming Explained: Embrace Chage (2nd ed)*. 2004: Addison-Wesley. 224.

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

[5] Cockburn, A. and Highsmith, J., *Agile software development: The people factor.* Computer, 2001. **34**(11): p. 131-133.

[6] Cohen, D., Lindvall, M., and Costa, P., *An Introduction to Agile Methods*, in *Advances in Computers, Advances in Software Engineering*, M.V. Zelkowitz, Editor. 2004, Elsevier: Amsterdam.

[7] Cohn, M. and Ford, D., *Introducing an agile process to an organization [software development]*. Computer, 2003. **36**(6): p. 74-78.

[8] Dingsøyr, T., Hanssen, G.K., Dybå, T., Anker, G., and Nygaard, J.O., *Developing Software with Scrum in a Small Cross-Organizational Project*, in *13th European Conference on Software Process Improvement (EuroSPI)*, I. Richardson, P. Runeson, and R. Messnarz, Editors. 2006, Springer Verlag: Joensuu, Finland. p. 5-15.

[9] Dybå, T. and Dingsøyr, T., *Empirical Studies of Agile Software Development: A Systematic Review.* Submitted to Information and Software Technology..

[10] Erickson, J., Lyytinen, K., and Siau, K., *Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research.* Journal of Database Management, 2005. **16**(4): p. 88 - 100.

[11] Fenton-O'Creevy, M., *Employee involvement and the middle manager: evidence from a survey of organizations.* Journal of Organizational Behavior, 1998. **19**(1): p. 67-84.

[12] Fitzgerald, B., Hartnett, G., and Conboy, K., *Customizing agile methods to software practices at Intel Shannon.* European Journal of Information Systems, 2006. **15**(2): p. 200-213.

[13] Gemunden, H.G., Salomo, S., and Krieger, A., *The influence of project autonomy on project success.* International Journal of Project Management, 2005. **23**(5 SPEC ISS): p. 366-373.

[14] Guzzo, R.A. and Dickson, M.W., *Teams in organizations: Recent research on performance and effectiveness.* Annual Review of Psychology, 1996. **47**: p. 307-338.

[15] Hoegl, M. and Parboteeah, K.P., *Autonomy and teamwork in innovative projects.* Human Resource Management, 2006. **45**(1): p. 67-79.

[16] Janz, B.D., Colquitt, J.A., and Noe, R.A., *Knowledge worker team effectiveness: The role of autonomy, interdependence, team development, and contextual support variables.* Personnel Psychology, 1997. **50**(4): p. 877-904.

[17] Jorgensen, D.L., *Participant Observation: A Methodology for Human Studies*. 1989, Thousands Oak, California: Sage publications.

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

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

[20] Mann, C. and Maurer, F., *A case study on the Impact of Scrum on Overtime and Customer Satisfaction*, in *Proceedings of Agile 2005*. 2005, IEEE Press: Denver.

[21] Marks, M.A., *A temporally based framework and taxonomy of team processes.* The Academy of Management review, 2001. **26**(3): p. 356.

[22] Miles, R.E., Snow, C.C., and Miles, G., *TheFuture.org.* Long Range Planning, 2000. **33**(3): p. 300-321.

[23] Morgan, G., *Images of organization*. Updated edition ed. 2006: Sage Publications.

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

[25] Nerur, S., Mahapatra, R., and Mangalaraj, G., *Challenges of migrating to agile methodologies.* Communications of the ACM, 2005. **48**(5): p. 72.

[26] Nonaka, I. and Takeuchi, H., *The knowledge-creating company*. 1995: Oxford University Press.

[27] Rising, L. and Janoff, N.S., *The Scrum software development process for small teams.* Ieee Software, 2000. **17**(4): p. 26-+.

[28] Salas, E., Sims, D.E., and Burke, C.S., *Is there a "big five" in teamwork?* Small Group Research, 2005. **36**(5): p. 555-599.

[29] Schwaber, K. and Beedle, *Agile Software Development with Scrum*. 2001: Upper Saddle River: Prentice Hall.

[30] Strauss, A. and Corbin, J., *Basics of Qualitative Research: Second edition*. 1998: Sage Publications.

[31] Takeuchi, H. and Nonaka, I., *The New New Product Development Game.* Harvard Business Review 64: 137-146 1986.

[32] Tata, J. and Prasad, S., *Team Self-management, Organizational Structure, and Judgments of Team Effectiveness*. Journal of Managerial Issues, 2004. **Vol. 16** (Issue 2): p. p248 - 265.

[33] Trist, E., *The evolution of socio-technical systems: a conceptual framework and an action research program*. 1981, Ontario Quality of Working Life Centre: Ontario.

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

[35] van Mierlo, H., *Individual autonomy in work teams: The role of team autonomy, self-efficacy, and social support.* European Journal of Work and Organizational Psychology, 2006. **15**(3): p. 281.

[36] Yin, R.K., *Case Study Research: design and methods*. 3rd ed. Applied Social Research Methods. 2003: Sage Publications. 179.