Building Blocks for Self-Organizing Software Development Teams: A Framework Model and Empirical Pilot Study

Henri Karhatsu, Marko Ikonen, Petri Kettunen, Fabian Fagerholm and Pekka Abrahamsson
Department of Computer Science
P. O. Box 68, FIN-00014 University of Helsinki, Finland
{firstname.lastname}@cs.helsinki.fi

***Abstract*—Self-organization is one of the foundations of agile software development. Many positive outcomes have been associated with having teams operating at high levels of self-organization. This paper reports the results of a pilot study which reviews the existing body of empirical literature and presents a novel model for building self-organizing teams. The model is empirically validated in two case studies performed in Software Factory, an academic but close-to-industry experimental R&D laboratory. It is shown that autonomy together with communication and collaboration are the major components for building self-organizing software development teams.**

***Index Terms*—self-organizing teams, self-organization, agile software development**

I. Introduction

There are many claimed advantages that self-organizing teams may bring to organizations [1]–[6]. Positive outcomes are often related to performance effectiveness, member attitudes, and behavior [2]. Self-organizing teams can react to problems quickly since the decision-making is close to the problem [7]. Instead of waiting for a manager's approval, the team has the authority to take necessary actions by itself [8].

A significant amount of self-organization literature exists. The history of self-organizing teams in the literature goes back to the 1950s when Trist et al. examined self-regulated coal miners [7]. Since then the management literature has addressed this area a lot using a number of different concepts and terms such as empowered teams, autonomous work groups, semi-autonomous work groups, self-managing teams, self-determining teams, self-designing teams, crews, cross-functional teams, quality circles, project teams, task forces, emergency response teams, and committees [3].

Self-organization has emerged as an important area of study also within software engineering management, in particular with agile methods such as Scrum. However, despite the often claimed benefits, there is a shortage of conclusive empirical studies in the area. Moe et al. [8], [9] have proposed a framework for measuring self-organizing teams based on an action research study in industrial settings. Hoda [10] interviewed practitioners and identified roles that are taken by self-organizing in action.

This paper aims at contributing to the growing body of empirical understanding in the area on how to build a self-organizing software development team. A novel model for building self-organizing teams is constructed and empirically tested. We present results from two case studies performed in a close-to-industry experimental R&D laboratory called Software Factory [11]. Software Factory facilities are designed to support such empirical studies. The results show that autonomy together with communication and collaboration are the major components for building a self-organizing software development team.

The rest of the paper is composed as follows: In Section II, the research model is created based on related work in the reference disciplines. Section III presents the empirical research design which is followed by the presentation of the case study results in Section IV. The conclusions are identified and discussed in Section V. The paper is summarized with Section VI addressing the limitations and identifying future research needs.

II. Research Model

In this section, a research model is created based on existing literature. Subsection II-A examines the characteristics of self-organizing teams. Subsection II-B describes how agile software development methods support self-organization. The resulting research model is presented in Subsection II-C.

A. Self-organizing Teams

Several definitions for work groups and teams exist [3]. When we refer to a self-organizing team in this paper, we use the definition of an autonomous work group by Guzzo and Dickson [3]: "Autonomous work groups are 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 (usually up to a specific limited value)."

In general, the related literature suggests that self-organizing teams bring many advantages for the organizations. One of the key benefits is performance effectiveness which means, for example, better productivity, lower response time, better quality and customer satisfaction, and more innovations [2]. Customer satisfaction is reported in many studies [1], [4],

[5], [12]. The main reason for the effectiveness is that self-organizing teams can react to problems quickly since the decision making is close to the problem [7], [8]. Instead of waiting for a manager's acceptance, the team has authority to take necessary actions.

Other advantages are positive changes in team member attitudes, including increased job satisfaction, stronger commitment to the organization, and trust toward management [2]. These are suggested in many studies [3]–[5], [8]. One reason for the attitude changes is that self-organizing teams stimulate participation and commitment, which make the employees care more for their work [13], [14]. Moreover, self-organization causes positive behavioral outcomes containing the level of absenteeism, turnover, and safety [2]. This is reported in many studies as well [3].

However, some research results are contradictory [3], [7]. E.g., some results indicate no connection between empowerment and success or that the project performance is not increased [15]. This means that self-organization is not a panacea, and just calling a team self-organizing does not automatically translate into better performance. The organizational context like the reward system, leadership, training, available resources, and the structure of the organization influence how teams can self-organize and perform [2], [7].

The contradictory research results indicate a need to better understand what makes a self-organizing team successful [13]. Moe et al. suggest five characteristics that are important for self-organizing teams: *autonomy, team orientation, shared leadership, redundancy, and learning* [9].

Autonomy refers to the authority and responsibility that a team has in their work [3]. It is a significant factor for team effectiveness [16]. A team must have a real possibility to influence relevant matters; otherwise self-organization is more symbolic than real [7]. On the other hand, a team should not be left completely alone [17]. Instead, while management should give a team substantial freedom, it should maintain subtle control and have regular checkpoints [17].

Three levels of autonomy are external, internal, and individual [14], [18]. The external refers to the degree that the people outside of a team influence the team's decisions [18]. Moreover, it sets the decision-making boundaries for the team. Meanwhile, internal autonomy defines how the work is organized inside the team [18]. The team may have substantial power to make decisions while some individuals have none [19]. Great care should be taken to make sure that there really is internal autonomy instead of, for example, team leader autonomy [18]. Finally, individual autonomy, on its part, tells how much an individual has freedom to decide about his or her own work processes [16].

Team orientation tells how well the goals of a team and the individuals meet [9]. Many researchers suggest that individuals should emphasize the team goals over their own [4], [20], [21]. On one hand, losing individual autonomy is harmful for the individual's motivation [22]. On the other hand, too much individual autonomy is a threat for team work [16].

Shared leadership means that the leadership role should be given to those who have the best skills and knowledge to decide about the particular issue [23]. This is in contrast to centralized decision making where one person makes all the decisions [18]. Shared leadership thus requires that the team has good internal autonomy. In addition, all the team members should be involved in decision making [9].

Redundancy is required in self-organizing teams so that the team members are able to do each other's work [9], [24]. In teamwork literature, this is often called backup behavior [14]. In practice, redundancy requires that the team members know what the others are doing and that they also have complementary skills [17], [24]. The idea of redundancy is opposite to individualism where each person specializes in some area of work. Traditionally, this has been seen as an effective way to organize work but it can be dangerous for an organization since the flexibility of teams deteriorates and the teams become more vulnerable [8].

Learning is needed in a self-organizing team for many reasons. Redundancy requires that the team members learn from each other [4]. Shared leadership requires a mechanism for learning since otherwise team members are not able to make decisions together [25]. Moreover, the team cannot make correct decisions in a changing environment without learning [26]. In addition, team orientation relates to learning. If an individual's success is determined by the team's success, there needs to be cooperative learning [4].

In addition to the five core elements discussed above (autonomy, team orientation, shared leadership, redundancy, and learning), **communication and collaboration** (C&C) play an important role. Communication means sending and receiving information, collaboration means actively working together to deliver a work product or make a decision [27]. C&C is important for many reasons. Shared leadership requires that all the team members actively participate in decision making [9]. One way to enhance team orientation is to increase information sharing [28]. Moreover, redundancy and learning require C&C as well. Autonomy cannot work properly without communication. As autonomy is based on trust from management [17], the team has to somehow communicate their progress to the management so that the trust does not gradually disappear.

B. Agile Software Practices Supporting Self-organization

Subsection II-A examined six general elements that are considered necessary for building self-organization in a team. However, it did not specify how these elements can be realized in practice in software development teams.

Agile software development teams are characterized by self-organization [27]. Following that idea, this subsection explores how agile software methods support self-organization. The methods under discussion are Extreme Programming (XP), Scrum, Crystal family of methodologies, Feature Driven Development (FDD), Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD), Agile Modeling (AM), and lean software development (lean).

Communication and collaboration are at the heart of agile software development. As the Agile Manifesto states,

"individuals and interactions over processes and tools" and "customer collaboration over contract negotiation" [29]. One aspect in C&C is customer cooperation. It is emphasized in XP due to the on-site customer practice [30]. Crystal suggests having user viewings, ASD customer focus group reviews and JAD sessions together with the customer [26], [31].

Another aspect is C&C inside the team. This is supported by the concepts of pair programming and open workspace in XP, and that of co-location in Crystal Clear [30], [31]. Scrum has its sprint planning meetings, daily scrum, and retrospectives [32], [33]. The AM practices model with others, display models publicly, and model to communicate should enhance C&C [34]. Moreover, Scrum board and Kanban board (in lean) are examples of information radiators that enhance communication by increasing the awareness and certainty regarding project activity [35].

One viewpoint to C&C is its continuity. For instance, daily Scrum meetings are held every day, and each iteration should have some kind of customer and team review. Co-location enables the team to talk face-to-face anytime and pair programming makes the communication between two developers inevitable. Also use of information radiators reveals project status to the team all the time.

Team **autonomy** requires that the team is authorized. This is supported in many agile methods. Scrum teams are considered self-organizing teams which have substantial decision authority and responsibility like planning, scheduling, work allocation, and operational decision making [19]. Likewise, empowering the team is a DSDM practice and a lean principle [31], [36]. Preserving autonomy is, however, not self-evident [20]. For this reason, Scrum Master in Scrum and project manager in FDD should protect the autonomy [20], [31].

Team orientation can be supported with increasing information sharing and participatory goal setting [28]. C&C practices listed above support the first one, and XP (planning game), Scrum (sprint planning meeting), and Crystal (staging) encourage to the latter [30], [31], [33]. Since team orientation is closely related to goal setting, priorities are important. In Scrum, the product backlog is re-prioritized at least before every iteration by the product owner [33].

On the general level it is acknowledged that agile software organizations need leadership-and-collaboration instead of command-and-control management [27]. This relates to **shared leadership**. However, agile methods do not provide many practices that actually would support this but rather emphasize the idea of shared leadership on an abstract level, like the leadership practice in lean software development does [36]. Nevertheless, since shared leadership requires that a team must have the right people with mission-critical knowledge, skills, and abilities [23], the concept of cross-functional teams in Scrum and Crystal [31], [33] supports shared leadership.

Redundancy can be supported with practices that increase the team members' ability to do other team members' tasks. Collective ownership in XP and AM [31], [34] is a way to share responsibility of work and that way increase the ability. In addition, if the team agrees on uniformity for example in respect to code, they can more easily continue the work done by the others. Practices supporting this are the coding standards in XP, applying modeling standards in AM, and notation standards, design conventions and formatting standards in Crystal Orange [30], [31], [34]. Moreover, all the practices supporting C&C support redundancy as well since that way team members get better understanding of others' work.

Also **learning**, for example from customers, leans on C&C. In addition, short iterations give the team a possibility to learn about the customer domain and the system to be developed [36]. They are important basically in all agile methods as incremental development with rapid cycles is one of the characteristics of the agile methods [31]. Learning during iterations can be supported with end-of-iteration review sessions like sprint reviews and retrospectives in Scrum, user viewings in Crystal, and JAD sessions and customer focus group reviews in ASD [26], [31]–[33].

Learning requires feedback [25], which is emphasized in ASD, XP, and lean software development [26], [30], [36]. One way to provide feedback for team members are the design, code, test, or plan reviews [26]. On the technical level automated regression testing, continuous integration, and regular builds in XP, Crystal, and FDD provide feedback for developers [30], [31]. Finally, tracking progress gives the team a possibility to learn. XP has a special role called tracker for this [30]. Similarly Crystal encourages progress tracking and monitoring [31].

C. A Model for Building Self-organizing Software Development Team

Based on the previous subsections, we propose a framework model for building a self-organizing software development team (Figure 1). The model contains the six general elements of self-organization (Subsection II-A) and agile practices that support each element (Subsection II-B).

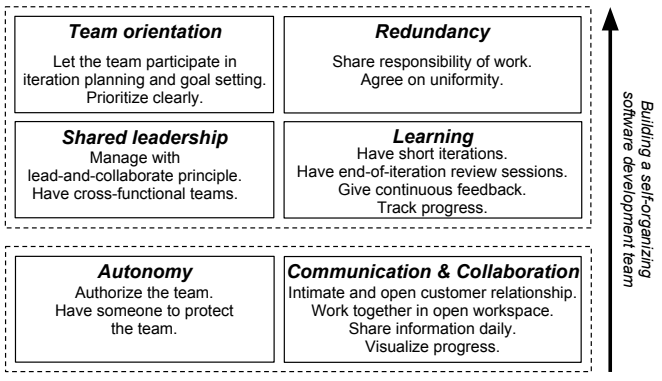

Fig. 1. A framework for building a self-organizing software development team. The arrow indicates the building direction; foundational elements must be in place first.

The model contains two foundational elements: autonomy and C&C. Without autonomy self-organization is symbolic only [7] meaning that if a team has no autonomy, it cannot

really act like a self-organizing work unit. On the other hand, C&C is important for the other elements, as described in Sections II-A and II-B.

The additional four elements of the model are shared leadership, learning, team orientation, and redundancy. A team should aim at these as well in order to become self-organizing.

The different agile practices are aggregated. The "end-of-iteration review sessions", for example, includes reflection workshops, retrospectives, and postmortems.

The proposed model does not prescribe fixed relationships between the elements. However, their inter-dependencies are supposed to be as explained in Sections II-A and II-B. The general idea is that the foundational elements are the most important in order to build a self-organizing software development team.

III. Empirical Study Design

The model presented in Section II-C and elaborated in the whole of Section II is based on selected parts of the literature, combined with observations from two project cases. The following explains the case study environment and approach.

Software Factory is a new software engineering research and education setting at the University of Helsinki [11]. It is basically an advanced R&D laboratory environment for conducting software business projects. The concept comprises the physical laboratory environment coupled with a unique operational model. The laboratory room is equipped with sophisticated computer and monitoring equipment (e.g., smart boards). Such high-end facilities make it possible not only to conduct the actual software engineering work in a modern fashion, but also to collect research data automatically (e.g., logs). In particular, the facility allows rich insight into the human-related aspects of software development [37]. The Department of Computer Science hosts an initial reference implementation. This facility has been operating since the beginning of 2010.

Software Factory continuously runs projects in seven-week cycles. In this study, the first author conducted participatory research in two web application development projects acting as a team leader and coach, and interviewed project team members and customer representatives. He had access to all project information and collaborated directly with customer representatives and fellow team members. The co-authors also observed the project and participated but to a much lesser degree.

In total ten team members and two customer representatives were interviewed. The interviews were thematic [38]. The interviewer concentrated on the same themes but the questions and details could vary between subjects. The themes were the self-organization dimensions described in Section II-A. The interviews were recorded and transcribed. Transcribed passages relevant to self-organization dimensions were analyzed. The transcriptions were also used to assess the project performance outcome as interpreted by the customer representative.

The findings are grounded primarily in the observations and perceptions of the first author. The qualitative interpretation of project goals, decision-making, group dynamics, team structure, and team member performance in different areas are dependent on the internal constructs of the observer. The degree of achievement in different dimensions is compared to the existing literature on self-organization.

The study design allows extraction of actual practices on project team self-organization. However, while the interview technique highlights the desired dimensions of study, it might also mask aspects of self-organization that has not been presented in the literature. This limitation is acknowledged [38].

IV. Results

This section evaluates the research model (Fig. 1) with the empirical data. All the elements and practices are covered. The section contains quotes from the interviews. The capital 'C' refers to a customer and 'TM' to a team member. The trailing number indicates the project (e.g., 'C1'). The letters from A to F designate the team members, respectively.

The research model (Fig. 1) suggests that in order to support **autonomy**, the teams should be authorized and there should be someone protecting the autonomy. In the case studies both teams were very much on their own. Management intervening lacked in daily work and for that reason the teams did not need anyone to protect autonomy.

There was, however, a difference in how the teams utilized their autonomy. Both teams had customer demos once a week where achievements from the previous week were presented and the next week tasks discussed. The first team wanted to keep discussion on a high level, which was appreciated by the customer:

> "I liked that when I started to talk things in too technical level, someone in the team said that 'sounds technical, we will get back to this in a week'." [C1]

On the other hand, the discussion in the second project demos went often to quite a detailed level:

> "They [the questions] were mostly technical but not big questions. - - Well, it was not this question but 'do we need this green button or red button?' type of question." [C2]

The other basic building block of the research model is **communication and collaboration**. The four practices suggested by the framework are intimate and open customer relationship, work together in open workspace, share information daily, and visualize progress.

Both teams had certain problems with customer communication. The first team complained that they would have needed more mutual time, which was also recognized by the customer:

> "I think at some stage when neither [of the customers] was present - - there was this period when we had to a bit longer do like 'I guess it goes like this'. - - The customer doesn't have to be there all the time but somewhere comes a limit that when he's not enough there, you have to start guessing." [TM1F]

> "When in the last week I was all the time present and visited there many times a day, many things that would have waited still a few days for the traditional customer demo, were solved there." [C1]

In the second project, the customer made himself available, but during the first half of the project the team did not use him much. The problem was identified in a retrospective meeting when a team member complained about unclear requirements. After that, communication with the customer increased. However, there was also another problem. When the team had problems in understanding what the customer is saying, they could not admit it:

> "Perhaps we didn't manage to say that it is not clear. We just said together with the other team members that 'yeah, yeah'. It probably slowed down." [TM2B]

The empirical evidence emphasizes the importance of an intimate and open customer relationship. As there were problems with that, both teams had difficulties in learning and becoming more self-organized.

On the other hand, the other practices proposed by the model were followed. Both teams worked in an open workspace, which had clear positive effects. For example, the team members easily got help from the others. The teams also had useful daily stand-up meetings:

> "And I think there were many times [in the daily meetings] that someone said: 'I am going to do this'. And then someone else said: 'Would you rather do this since it is more immediate or important?"' [TM1C]

The comment of the team member shows how daily information sharing supports team orientation. The daily meetings also supported redundancy since the team members awareness of others' tasks increased in the daily meetings.

Both teams used Kanban boards to visualize the project status. This was useful for learning since the boards revealed important information. In the first project, the team leader realized that it is important to get the board rather clean by the demo so that it is faster to react to the changing needs of the customer. In the second project, the board helped the team to see that the tasks get stuck in the code review stage and by changing the work-in-process limit on the board the situation got better.

According to the research model **shared leadership** requires that a team is managed with lead-and-collaborate principle and the team should be cross-functional. The empirical evidence shows that the first supports shared leadership and the lack of the second deteriorates it.

Neither of the teams had a preappointed leader. Instead, the most experienced persons took that role by themselves. The first author took the leader role in the first team.

The roles were not traditional manager roles. Leadership was rather shared among certain people. On the other hand, some team members were very inexperienced and could not take part in decision making:

> "I think the biggest single thing [to be decided] was the definition-of-done. - - It was so that the most experienced of us were discussing about it and the others quietly accepted it since they didn't have any better ideas." [TM1E]
> "I feel that me, [TM2D], and [a third team member] were somehow the driving force there. The others perhaps did not have technical knowledge or courage to make so much decisions." [TM2B]

According to the research model, **learning** can be supported with short iterations, end-of-iteration review sessions, continuous feedback, and progress tracking.

The two first practices were followed since both teams had one week iterations with customer demos in between. In addition, the teams had retrospectives. All these were seen as very useful. In the first project, one week iterations helped both the team and the customers to learn:

> "We had this practice, these customer demos, that became critical communication points. - - What is the direction and have they done what critically was supposed to do? - - In this kind of initiative the goal is crystal clear but still extremely movable." [C1]
> "I think they [the demos] were useful. You got to know the new direction." [TM1F]

Also the retrospectives helped the teams to learn and improve their processes, as was already noted above. In addition:

> "In the retros we noticed that we are not doing some things well enough. So let's try to get better testing routine for everyone." [TM1D]
> "People could tell their own viewpoints of what had happened. It opened my own eyes. When you thought these things that you are uncertain about, you could see them from the others' perspective." [TM2B]
> "Initially no-one was happy with the Kanban [board], and that we got to know in the first retro." [TM2C]

However, it is noteworthy to state that the second team was initially unwilling to hold retrospectives. The coach of the team explained to them why it is good to have one, and afterwards the team appreciated the meeting very much.

Both teams had a code review stage in their processes. The first team considered it a very good source of feedback but the second team had problems with code reviews. The second comment shows how important good feedback is for learning:

> "I think the code review stage was extremely good because you got immediate feedback." [TM1C]
> "In some cases it was that people had done it like it was not supposed to be done. If it was somewhat understandable, I let it pass. Maybe I should have been more rigorous especially in the beginning of the project. It would have perhaps reduced at later stage that there will be more of that [bad code]. - - I was too kind perhaps." [TM2D]

Neither of the teams tracked their progress as the research model would suggest. It is difficult to say based on the empirical data how this affected the teams' performance.

The research model suggests that **team orientation** can be supported with two practices: let the team participate in iteration planning and goal setting, and prioritize clearly. The usefulness of both practices can be seen in the case studies.

Iterations were planned and tasks prioritized in the customer demos. Most of the team members in both teams participated in the demos, which gave them good insight of the next iteration tasks. On the other hand, when one team member could not participate, he commented:

> "I had such a feeling that it would have been useful to be in the demos. I would have known better what is happening in the project. - - Maybe the biggest disadvantage was when I did code review or QA for others. I didn't have a clear vision if they were doing the right thing." [TM1E]

The demos were important also for prioritization. The first team used to always ask from the customer what are the most important tasks for the next week. The customer liked this approach:

> "The team leader asked that on what they should concentrate during the next week. - - I felt it was actually a good thing to do. It makes you to summarize the thematics into a couple of sentences." [C1]

After a demo, it was very clear what is important to do. This gave the team members a possibility to choose tasks on their own:

> "Everyone individually, they are taking tasks from the ticketing system. And they were just checking what are the priorities of the tasks and they are assigning these tasks to themselves." [TM1A]

The team members liked the freedom of choosing tasks but at the same time the clear prioritization helped them to maintain team orientation.

The last element of the research model is **redundancy** with shared responsibility of work and agreement on uniformity practices. In the first team, the first practice was followed. Different team members contributed to the same parts of the system and code review was done by many people. In the second team, the situation was however different:

> "There were many people, majority of people I should say, that they were able to work on the task what I had been working." [TM1B]
>
> "It [transferring a task to someone else] probably wouldn't be easy because I did pretty much, and everyone else as well, individually. Even though we sometimes asked advice and discussed, the implementation was done alone." [TM2B]
>
> "My code review was done by [TM2B] mostly." [TM2A]
>
> "It was a bit that I was ready to review others' code but I felt that [TM2D] was the only one who wanted to check my code." [TM2B]

Since the responsibility of work was not well shared in the second project, certain people were burdened more than the others. On the other hand, the vulnerability of the team did not become visible since the project was so short (seven weeks). It is, however, probable that if the project would have lasted longer, the lack of redundancy would have affected self-organization even more.

Both teams agreed on certain uniformity practices like coding conventions. However, there is not enough empirical data available to determine what kind of effect this had on redundancy.

As a conclusion, the pilot case study supports the research model in general. Both projects ended with favorable customer assessments with respect to the project outcomes. Regardless, the role of certain practices (e.g., tracking project progress) requires more evidence.

V. Discussion

The proposed research model (Fig. 1) leans on the five characteristics of a self-organizing team suggested by Moe et al. [9] and adds two new components: C&C and agile practices. Based on the empirical evidence (Section IV), the research model seems to provides a good framework to understand how self-organizing software development teams can be built.

The two case projects differed in the degree that they could reap benefits from their autonomy. The crucial factor appears to have been the difference of leadership. However, the question is not about vertical leadership which was not needed e.g. for task specification and identifying team members roles as Pearce [23] suggests. Instead, leadership in general is important and it can be shared.

According to the literature, team members should emphasize the team goals over their own goals [4], [20], [35], which may be in contradiction with individual autonomy [14], [19]. The empirical data show how these two can be combined with clear prioritization and individuals' freedom of choosing tasks.

Constantly sharing information and switching responsibilities in tasks in order to avoid specialism and individualism was recognized in the case studies. All of these conform to the literature [8], [14], [20]. Similarly, the importance of honest and continuous feedback supports the existing literature [36].

The case studies emphasized the importance of good customer communication in software development projects. If the customer is not available or if the team and the customer are not able to discuss openly, the team cannot get the feedback needed. From the self-organization point of view, this deteriorates learning in the team. The importance of customer communication is recognized e.g. in XP that suggests co-located customer [30]. If this is not possible, the team should find other ways to communicate effectively with the customer.

The research model is based on agile practices and how they support the elements of a self-organizing team. The empirical data (Section IV) provided initial evidence that the research model can explain how self-organization can be built. This has

two implications. First, the team should have an understanding of agile practices. One way is to use a coach like in the second project. An example of how the coach intervened was the argumentation of the importance of retrospectives (see above). Before the first retrospective, the team was not reluctant to have it but afterwards the team considered it very useful. Second, although e.g. Scrum requires self-organizing teams but does not clearly explain how they can be built [39], the creation of self-organization seems to be possible by using a combination of certain agile practices.

The literature claims that transformation of a work group into a self-organizing work team takes years [5]. The case studies were very short projects (seven weeks) but rather high level of self-organization was reached at least in the first project. Moreover, the customer was satisfied with the project results. On the other hand, the empirical data showed that self-organization does not just emerge without conscious effort. Interviewees expressed in numerous comments how they had to adjust to others and take bearings on each other as they solved project challenges and tried to find their way forward. This is something that has to be done constantly and further research is needed in order to test the research model in longer projects. Nevertheless, the pilot study indicates that building a self-organizing software development team is possible, at least to some extent, in a very short period of time.

VI. Conclusions

This paper proposed a framework model for building self-organizing software teams. The model presented in Section II-C is based on selected parts of the related organizational management and agile software development literature, supported with empirical observations from the two project cases.

The main conclusions of the model construct and empirical evidence are that team autonomy together with efficient communication and collaboration are the foundational building elements for effective self-organization. These are also the key tenets underlying many agile software development methods.

That said, this investigation opens up new avenues for further research: How does the project context factors influence the self-organization model space? For instance, is the project team authorized to make all the decisions related to software requirements? Can the proposed model be used to actively "build" self-organizing teams (possibly even in a very short period of time) and if so, what are the factors of such a building activity? How does the starting point of the team, such as member selection and experience, moderate the self-organization behavior and resulting performance? How does cultural background influence self-organization, and how do multicultural self-organizing teams get built? These are some candidates for directions that could be further explored.

In all, the project performance effects of self-organization under different circumstances need more evidence and further theory-building in order to be able to draw firm practical recommendations for different software organizations.

Acknowledgements

This work was supported by TEKES as a part of the Cloud Software program of Tivit (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT). The authors would also like to thank the case project team members for their cooperation.

References

[1] L. Behnke, R. Hamlin, and B. Smoak, "The evolution of employee empowerment," *Semiconductor Manufacturing, IEEE Transactions on*, vol. 6, no. 2, pp. 143 –155, may 1993.

[2] D. Cohen, Susan; Bailey, "What Makes Teams Work: Group Effectiveness Research from the Shop Floor to the Executive Suite," *Jounal of Management*, vol. 23, no. 3, pp. 239–290, 1997.

[3] R. Guzzo and M. Dickson, "Teams in organizations: Recent research on performance and effectiveness," *Annual review of psychology*, vol. 47, no. 1, p. 307–338, 1996.

[4] B. D. Janz, "The best and worst of teams: self-directed work teams as an information systems development workforce strategy," in *SIGCPR '98: Proceedings of the 1998 ACM SIGCPR conference on Computer personnel research*. New York, NY, USA: ACM, 1998, pp. 59–67.

[5] C. Jian, "Research on Strategies and Empowerment Process to Achieve Self-management Team," *Area*, pp. 25–29, 2008.

[6] J. Underwood, M. Fitzgerald, and M. Cassidy, "Self-directed teams in power electronics manufacturing," in *Applied Power Electronics Conference and Exposition, 1996. APEC '96. Conference Proceedings 1996., Eleventh Annual*, vol. 1, 3-7 1996, pp. 64 –68 vol.1.

[7] J. Tata and S. Prasad, "Team Self-Management, Organizational Structure, and Judgments of Team Effectiveness," *Jounal of Managerial Issues*, vol. XVI, no. 2, pp. 248–265, 2004.

[8] N. Moe, T. Dingsøyr, and T. Dybå, "Overcoming barriers to self-management in software teams," *IEEE Software*, vol. 26, no. 6, p. 20–26, 2009.

[9] N. B. Moe, T. Dingsøyr, and E. A. Røyrvik, "Putting agile teamwork to the test — an preliminary instrument for empirically assessing and improving agile software development," in *Agile Processes in Software Engineering and Extreme Programming*, Pula, Sardinia, Italy, 2009.

[10] R. Hoda, J. Noble, and S. Marshall, "Organizing self-organizing teams," in *ICSE '10: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*. New York, NY, USA: ACM, 2010, pp. 285–294.

[11] P. Abrahamsson, "Unique infrastructure investment: Introducing the Software Factory Concept," *Software Factory Magazine*, vol. 1, pp. 2–3, 2010.

[12] R. Winter, "Self-directed work teams," *Proceedings of 1994 IEEE/SEMI Advanced Semiconductor Manufacturing Conference and Workshop (ASMC)*, vol. 00, pp. 123–125, 1994.

[13] M. Fenton-O'Creevy, "Employee involvement and the middle manager: evidence from a survey of organizations," *Journal of Organizational Behavior*, vol. 19, no. 1, pp. 67–84, January 1998.

[14] N. Moe, T. Dingsøyr, and T. Dybå, "Understanding self-organizing teams in agile software development," in *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, 26-28 2008, pp. 76 –85.

[15] R. Reilly and G. Lynn, "Power and empowerment: the role of top management support and team empowerment in new product development," *PICMET '03: Portland International Conference on Management of Engineering and Technology Technology Management for Reshaping the World, 2003.*, pp. 282–289, 2003.

[16] C. W. Langfred, "The paradox of self-management: individual and group autonomy in work groups," *Journal of Organizational Behavior*, vol. 21, no. 5, pp. 563–585, August 2000.

[17] H. Takeuchi and I. Nonaka, "The new new product development game," *Harvard Business Review*, vol. 64, no. 1, p. 137–146, 1986.

[18] M. Hoegl and P. Parboteeah, "Autonomy and teamwork in innovative projects," *Human Resource Management*, vol. 45, no. 1, p. 67–79, 2006.

[19] H. T. e. a. Barney, "Balancing individual and collaborative work in agile teams," in *Agile Processes in Software Engineering and Extreme Programming*, Pula, Sardinia, Italy, 2009.

[20] N. B. Moe and A. Aurum, "Understanding Decision-Making in Agile Software Development: A Case-study," *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, pp. 216–223, 2008.

[21] R. Whitehead, *Leading a Software Development Team – A Developer's Guide to Successfully Leading People & Projects*. Addison-Wesley, 2001.

[22] M. Gagne and E. L. Deci, "Self-determination theory and work motivation," *Journal of Organizational Behavior*, vol. 26, no. 4, pp. 331–362, 2005.

[23] C. Pearce, "The future of leadership: Combining vertical and shared leadership to transform knowledge work," *Academy of Management Executive*, vol. 18, no. 1, p. 47–57, 2004.

[24] S. Nerur and V. Balijepally, "Theoretical reflections on agile development methodologies," *Communications of the ACM*, vol. 50, no. 3, p. 83, 2007.

[25] N. Moe, T. Dingsøyr, and O. Kvangardsnes, "Understanding shared leadership in agile development: A case study," in *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, 5-8 2009, pp. 1 –10.

[26] J. Highsmith, "Retiring Lifecycle Dinosaurs – Using Adaptive Software Development to meet the challenges of a highspeed, high-change environment," *Software Testing and Quality Engineering*, vol. May/June, p. 22–28, 2000.

[27] A. Cockburn and J. Highsmith, "Agile Software Development, the People Factor," *Computer*, vol. 34, no. 11, p. 131–133, 2001.

[28] E. Salas, D. E. Sims, and S. Burke, "Is there a "Big Five" in Teamwork?" *Small Group Research*, vol. 36, no. 5, pp. 555–599, October 2005.

[29] "Agile manifesto," 2001. [Online]. Available: http://www.agilemanifesto.org/

[30] K. Beck, *Extreme Programming Explained: Embrace Change*, 1999.

[31] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods: Review and Analysis*. Espoo, Finland: Technical Research Centre of Finland, VTT, 2002.

[32] H. Kniberg, *Scrum and XP from the Trenches*. C4Media, 2007.

[33] D. Leffingwell, *Scaling Software Agility: Best Practices for Large Enterprises*. Addison-Wesley Professional, 2007.

[34] S. Ambler, "Lessons in agility from Internet-based development," *IEEE Software*, vol. 19, no. 2, pp. 66–73, 2002.

[35] E. Whitworth, "Experience Report: The Social Nature of Agile Teams," in *Agile 2008 Conference*. IEEE, 2008, p. 429–435.

[36] M. Poppendieck and T. Poppendieck, *Lean Software Development - An Agile Toolkit*. Addison-Wesley, 2003.

[37] F. Fagerholm, "Psychometric measurements in software development," *Software Factory Magazine*, vol. 1, no. 1, pp. 12–13, March 2010.

[38] W. T. Rupp, "Qualitative evaluation and research methods: Michael Quinn Patton, Sage Publications, Newbury Park, CA. 1990," *Journal of Business Research*, vol. 30, no. 2, pp. 197–199, 1994.

[39] N. B. Moe and T. Dingsøyr, "Scrum and team effectiveness: Theory and practice," in *Agile Processes in Software Engineering and Extreme Programming*, Limerick, Ireland, 2008, pp. 11–20.