Agile Process Improvement: Diagnosis and Planning to Improve Teamwork

Conference Paper *in* Communications in Computer and Information Science · June 2011

DOI: 10.1007/978-3-642-22206-1_15 · Source: DBLP

3 authors, including:

Torgeir Dingsøyr
Norwegian University of Science and Technology
150 PUBLICATIONS **8,782** CITATIONS
SEE PROFILE

Nils Brede Moe
SINTEF
175 PUBLICATIONS **5,245** CITATIONS
SEE PROFILE

Agile Process Improvement: Diagnosis and Planning to Improve Teamwork

Mats Angermo Ringstad[1], Torgeir Dingsøyr[2,3], and Nils Brede Moe[2]

[1] Acando AS, NO-0159 Oslo, Norway
[2] SINTEF, NO-7465 Trondheim, Norway
[3] Dept. of Computer and Information Science,
Norwegian University of Science and Technology,
NO-7491 Trondheim, Norway

Abstract. Agile software development addresses software process improvement within teams. Process improvement, although a central concept in agile development, is still hard to achieve. This paper argues for the use of diagnosis and action planning to improve teamwork in agile software development. Diagnosis and action planning is illustrated in a small and immature team and in a large and more mature team. The action planning focused on improving shared leadership, team orientation and learning. The improvement project provided most new insight for the mature team.

Keywords: software process improvement, teamwork, agile software development.

1 Introduction

Software process improvement (SPI) [1] is an important part of all approaches to software development. In the plan-driven or traditional software development, the process improvement focus has mainly been on explicitly defining processes that can be standardized both within and across organizations [2]. SPI in this approach focuses on optimization. In agile software development [3], the goal of optimization is replaced by goals of high flexibility and responsiveness [4]. Subsequently, the agile perspective also changes the way of doing software process improvement. According to Salo and Abrahamsson [5] this requires new SPI mechanisms. Agile software development addresses software process improvement and management of software development practices within individual teams.

Given the focus on improving teamwork, there is a need for methods and techniques describing and diagnosing such teams. The research method action research [6] involves diagnosis and action planning, and fosters participative improvement. This method has further been suggested as a research method that can give results relevant to industry in addition to preserving scientific rigour. Our research question is: *How to efficiently improve teamwork in agile software development?*

The rest of this paper is organized as follows: First, we give an overview of theory on the topic of teamwork in agile software development teams. Further, we outline

R.V. O'Connor, J. Pries-Heje, and R. Messnarz (Eds.): EuroSPI 2011, CCIS 172, pp. 167–178, 2011.

previous research on process improvement in this setting. Second, we describe the context of research, and how diagnosis and action planning was conducted, and continue to show results from this research during the diagnosis and action planning phases. Third, we discuss this way of organizing process improvement on agile software development and contrast it to previous work. Finally, we describe main conclusions and implications for theory and practice.

1.1 Characteristics of Agile Teams

To understand process improvement in agile software development, it is important to understand the nature of agile teams.

Agile development focuses on collaboration, informal communication and desire an organic organizational form [7]. Such organizations are characterized by being flexible, participative and encouraging cooperative social action.

Agile teams are usually co-located and arrange daily meetings, which means that the team-members can see what the others are working on and the tasks they are doing. Then team-members get immediate evidence of the progress of the work, can adjust their own work accordingly, and know who is responsible for which tasks [8]. This makes the work predictable and easier for the team to create a common understanding. Also the bottom-up approach of planning helps creating a common understanding [8]. Further, the agile team is supposed to be self-managed and empowered, which means that the team members are responsible for managing, monitoring and improving their own processes [9].

The literature on self-organizing and self-managing teams, claims that the decision authority and leadership needs to be shared [10, 11]. This means that 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 [12]. While the project manager should maintain the leadership for project management duties, 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 [13]. The jointly shared decision authority should replace the centralized decision structure where one person makes all the decisions and the decentralized decision structure where all team members make decisions regarding their work individually and independently of other team members [14].

For the team to be able to self-manage, it must have a degree of redundancy [11]. The members need multiple skills so that they are able to perform (parts of) each other's jobs and substitute each other as circumstances demand. In this respect, socio-technical literature is concerned with "multiskilling" [15]. Studies of self-managing teams also show that this kind of organization requires a capacity for learning that allows operating norms and rules to change in relation to transformations in the wider environment [11]. Therefore, to succeed with agile development, both team and organization needs to focus on improving the development processes.

1.2 Process Improvement

Software process improvement has its roots in general improvement philosophies like total quality management, which has been tailored to software engineering in the

Quality Improvement Paradigm (QIP) [16], and in efforts on standardisation like the Software Engineering Institute's Capability Maturity Model Integration (CMMI).

Because the field has been found to be rather dominated by the capability maturity model (CMM) [18] - now CMMI, we refer to this model when we explain what we mean by the "traditional approach" or "classical SPI". CMMI focus on software processes, standardisation and software metrics as a basis for improvement [18]. This focus on software process is based on the premises that:

- The process of producing and evolving software products can be defined, managed, measured, and progressively improved.
- The quality of a software product is largely governed by the quality of the development process [19].

This approach prescribes norms for how individuals, teams or organizations should operate, and for how processes should be standardized and improved [20].

There are several fundamental differences between traditional and agile software development regarding SPI[5]. First, while SPI in the plan driven perspective prescribes norms for how the individual, team and organization should operate, agile software development address the improvement and management of software development practices within individual teams [2]. In agile development, processes are not products, but rather practices that evolve dynamically with the team as it adapts to the particular circumstances [21]. Second, plan-driven methods, such as the waterfall model, usually adopt a top-down approach for improving the software development process [5], while the agile view has a bottom-up approach. Third, SPI in plan-driven development often emphasizes the continuous improvement of the organizational software process for future projects, while the principles of agile software development focus on iterative adaption and improvement in the on-going projects. Short development cycles provide continuous and rapid loops to iterative learning, to enhance the process and to pilot the improvement.

When doing agile development, there are typically two meetings where the team focuses on improving the process. 1) Daily meetings. In the daily meeting the team members are supposed to coordinate their work and focuses on solving problems that stop the team from working effectively. In Scrum, the Scrum-master is supposed to facilitate this meeting and making sure impediments to the process are removed 2) Retrospective [22]. At the end of each iteration, a retrospective is held. In this meeting the team focuses on what was working well and what needs to be improved. Measures are then taken.

While the conclusion of the study of Aaen et al. [23] is that there is no recognized SPI model supporting the agile approach, we found two such frameworks. Qumer and Henderson-Sellers [24], suggest a framework that can be used to create, modify or tailor situation-specific agile software processes. The model includes an agility measurement model and an agile adoption and improvement model. Salo and Abrahamsson [5] defined an iterative improvement process for conducting SPI within agile software development teams.

A more specific approach to improve teamwork is the use of the team radar by Moe et al. [25]. In the next section we will describe usage of this.

2 Research Context; Diagnosis and Action Planning

The research was conducted in two teams in different companies. The teams were selected to illustrate diverse starting points with respect to software process improvement (key information on the teams can be found in table 1 and table 2).

Table 1. Properties of the maintenance and development team

Context	"Maintenance"	"Development"
Type of system	Web-based	Back-end of large system
Technology	Primarily Java	C and C++
Project size	140.000 lines of code, and several, open-source modules	3.000.000 lines of code
Project phase	Maintenance and adding new functionality	New development
Project length	Started in 2008, handed over to customer fall of 2009.	Started in early 1990's, still on-going.
Team size	Five: One senior and four junior developers	Eight senior developers
Team composition	Almost eight months	Almost four months

The *maintenance team* was a small team doing maintenance and adding new functionality to a web-based enterprise system that is used by operators all over Norway. The team consisted of three junior developers, one service desk operator with some system and programming knowledge, and a senior developer. The team had worked together for almost eight months, located in one room.

The *development team* worked in a division of a large international corporation, adding new functionality on a large system that was over 20 years old. The team developed new functionality for administrating the software, server software, and low-level modules used by a graphical client. The company had used Scrum for more than two years. The Scrum master also worked on another development project. The team had eight team members (including the product owner) who were all senior developers with several years of software development experience. Three of the team members were external, hired from consulting companies, all working for more than two years on the system under study. The team members worked in individual offices.

Table 2. Agile practices in the two teams

Agile practice	"Maintenance"	"Development"
Iterative development	Yes	Yes
Continuous integration	Yes	No
Sprint planning	No	Yes
Sprint demo	No	Yes
Sprint retrospective	No	Yes
Daily standup	No	Yes
Self-managing team	Yes	Yes
Refactoring	Yes	Yes
Co-location	Yes	Yes
Pair-programming	2 people	No

The diagnosing means to identify the primary problems and underlying causes of the organizations desire to change [6]. In our case, the scope was limited to improving teamwork, and we used an instrument developed earlier, the team radar [25], with the factors listed in table 3. The team radar is based on a literature review and experience from case studies, which have identified the five dimensions of the instrument as playing a pivotal role in agile teamwork.

Table 3. Factors in the team radar diagnosis instrument

Factor	Description
Shared leadership	Leadership is rotated to the person with key knowledge, there is jointly shared decision authority.
Team orientation	Priority is given to team goals more than individual goals, team members respect other members' behaviour.
Redundancy	Members have multiple skills so that they can perform (parts of) each others tasks.
Learning	The team develops shared mental models, and a capacity for learning to allow operating norms and rules to change.
Autonomy	The ability to regulate the boundary conditions of the team, the influence on management (and other externals) on activity.

The diagnosing phase consisted of collecting a rich data material for analysis, through observation and semi-structured interviews. The interviews lasted on average 30 minutes, and were transcribed for analysis. The first author, observed teamwork practice in daily work, and meetings like daily meetings, iteration planning and retrospective. Field notes were taken from the observations and integrated with the interview material for analysis. In the maintenance team it was collected 4 interviews and 8 observations, and in the development team 6 interviews and 7 observations. In both teams there was a diagnosing period of two weeks each. In addition, the first author had discussions with some of the team members about the projects and work methods to gain a solid understanding of the surrounding environment.

The end-result of diagnosing was a score between zero and ten on selected team radar factors (See Figure 1). The score was given on the basis of the collected answers from all team members as well as the observed practice. In the next chapter, we show characteristic statements that form the basis of the score. Note that the diagnosing should not be seen as a precise instrument to diagnose teamwork, but the instrument enables both knowledge of important aspects and the development of a language for engaging with teamwork change and follow-up.

The action planning seeks to specify organizational actions that should relieve or improve the primary problems identified in the diagnosis [6]. In action research, the plan should be guided by a theoretical framework, in our case the theory of teamwork effectiveness underlying the team radar used in the diagnosis phase. The planning was organized as a presentation of the results of the diagnosing, with an open discussion on whether the team recognized how teamwork was portrayed in the findings. Then, we discussed which areas should be given priority to improve teamwork, and finally discussed concrete actions to form an action plan.

The scope of this article is to give a better understanding of the diagnosing and action planning phases focusing on teamwork in agile development. However, as a result of the two phases described below, a subsequent visit to the maintenance team showed that two of the suggested improvement actions, daily meetings and

retrospectives, were re-implemented by the team. The development team made adjustments to their sprint planning based on the feedback. They focused on doing it more informally, using less time, and made it voluntary to attend.

3 Diagnosing Teamwork

To diagnose teamwork in the two teams, we used the team radar instrument to evaluate five aspects of teamwork. The total score on each factor is given in figure 1.

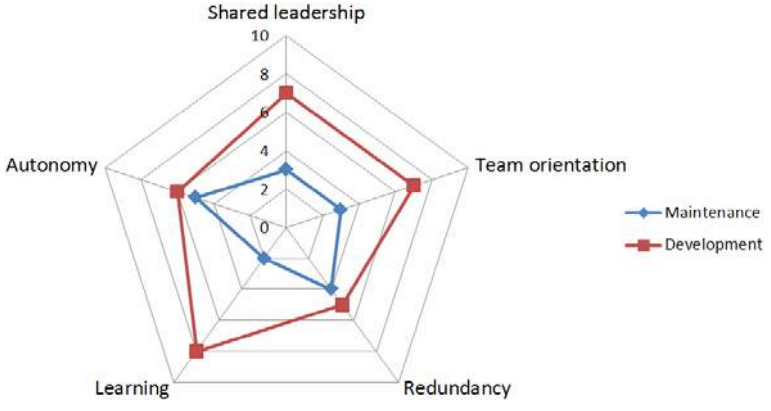

Fig. 1. A plot of teamwork characteristics of the two teams

The action planning phase involves a team-discussion to identify the right level for each factor, and the factors where both the company and researchers see a potential for improvement. Both teams chose shared leadership, team orientation and learning. As we see from figure 1, it is not necessarily the factors with the lowest score that are selected for action planning.

3.1 Shared Leadership

Shared leadership has a low score when the team-leader uses a "command and control" style of management, and when few take part in the decision-making process. A high score is given to teams which seek to engage everyone in leadership. Shared leadership implies that team members with knowledge about a certain area lead the discussions, and there is a shared decision-making process [25].

Maintenance team: The team members expressed that the team was well composed. When they felt they had knowledge about the issues discussed, the team members usually contributed to discussions and decision-making. The most important decisions, however, were made by the senior developer and the team leader in their weekly planning and status meetings. The reason for not involving the rest of the team in this meeting was the heavy workload on the rest of the developers. After these meetings, the senior developer reported back decisions, and what the team should

prioritize. Some decisions, like how the customer wanted the support function organized were received negatively by the team.

Another reason for why some were not participating in the shared decision-making process was lack of knowledge. Missing knowledge resulted in some team members not being able or interested in discussing other's tasks. As one of the developers said, "We have a competency hole in the system, there are some components we don't know… and other components that only one person knows. But we have a future goal of having overlap regarding knowledge about the most important components."

With respect to the project goal, the team felt that the initial goal and release-plan was clear. However, during the first month the product had severe performance problems, and this resulted in the customer contacting the team every day with change orders. So instead of following the plan, the team focused on day-to-day work trying to solve the performance issues.

Development team: The team members were pleased with the team composition, and as one of the team members said, "we have a very strong team".

Decisions regarding work and who was supposed to solve which tasks were usually taken during daily stand-up meetings. Team members were free to pick whatever task they wanted, but sometimes the observation revealed that certain tasks were always solved by the same team members. This typically happened when one of the team-members were seen as an expert on the task.

The team members were active in discussions on topics where they felt they had enough knowledge to take part, this was evident during the sprint planning and the daily stand-ups observed. The team would discuss until they decided by consensus. We observed that the team being located in individual offices was a barrier to a shared decision-making process. One said, "It can be hard to go into another office and ask for opinions or help. Therefore, our best arenas for discussions and alternative proposals are the meetings we have".

3.2 Team Orientation

For team orientation, a low score is given when individual goals are more important than the team goals, and where team members do not respect other team member's decisions. The highest score is where the team goals are the most important, and when team members respect each other's decisions [25].

Maintenance team: Alternative proposals were not common for several reasons; the senior would often make the decisions for the team, specialization within selected components resulted in developers not discussing issues with "their" components with others, and because of a high workload the team never prioritized discussing alternative proposals. Missing a shared decision-making process resulted in individual goals becoming more important than team goals. During observation, we saw little communication between the team members in the team room, except when coordinating who should do what, and reporting status. As one of the junior developers said; "We have not had much communication lately since everyone has been so busy and overworked…. the task-assigning communication which happens quite often, is disturbing. " This situation clearly hindered team-orientation.

The team members did not show an interest in other team members work unless it was affecting their own work, and subsequently it was difficult to strengthen the importance of the team-goals. "The only person here who is interested in what the others are doing, is the senior", said one of the junior developers.

Development team: Team orientation was stronger in this team, and it was clear that alternative proposal from other team-members when planning work was appreciated. "We are very open when it comes to suggest alternative solutions", said one developer. A good example of shared leadership was during an observed sprint demo where one of the team members held the whole presentation, not the Scrum master. "We have a very professional orientation to how we work with the product and the projects", one of the team members said, pointing to the fact that they would usually have thorough discussions in the team before making decisions.

While team commitment was strong, the team members did not have a clear conception of the long-term vision of the project, even though they had clear goals for each sprint. The product owner, who got the full overview of the system, acknowledged this, realizing that he was not good enough at sharing the long-term goals with the team.

Some of the team members explained that they felt ownership to the team-plans, while others said they had ownership to the system being developed but not the project. This decreased the team-orientation.

3.3 Learning

The learning factor has the lowest score in situations where there are no feedback mechanisms. The highest score is given when there is continuous improvement of work methods based on feedback [25].

Maintenance team: Because the team stopped holding retrospectives, there were no formal arenas for learning and improving. The team members did not see the need for a common improvement and feedback meeting, since this meeting had not earlier resulted in an improved process. The team continued work in the same manner every day. The only feedback given to the team was from the weekly meetings where only the senior developer and the team leader participated.

Development team: The team had several arenas for giving feedback on other's work. The most appreciated one was the sprint retrospective. In addition, they had daily stand-up meetings and additional design meetings.

The team discussed process related problems in the sprint retrospective, which made it possible to adjust the Scrum process to make it better fit the organization, and the team. However, several of the interviewees said they were missing good discussions on how to improve the teamwork. Also we found that some process problems were not reported in the retrospectives. One example was problems related to the planning meeting. This meeting spanned over two days, and every user story was discussed in detail. Usually everyone participated in the discussions, however sometimes two or three team members could discuss a user story for a long period of time, while the others were only listening. Then team-members felt excluded from participating actively in the meeting, and subsequently the meeting was seen as less

productive. “I am aware that our sprint planning is often ineffective, but I’m not sure how we can improve that”, the product owner said. This problem was not reported or discussed in the retrospective.

4 Action Planning: Measures to Improve Teamwork

To improve teamwork in the two teams, we presented the results of the diagnosis phase, and discussed priority on teamwork factors together with the teams. As a result concrete measures to be taken to improve the development processes and the teamwork were suggested.

For the maintenance team we observed challenges related to *shared leadership, team orientation, and learning*. As for leadership, the team was dominated by junior developers, there was little involvement of the team in leadership and little process in place. The team was heavily specialized, with team members working on independent modules, which again lowered team orientation. Finally, the team had no arenas for learning except for being in the same room, but observation showed little discussion and feedback on the actual work tasks the team members were involved in.

In a workshop, we presented the scores, problems and consequences to the team. The team decided to reintroduce important agile practices they had stopped doing. In prioritized order:

- Sprint retrospective to improve learning. Team members would be able to give feedback and improve both the development process as well as the product.
- Daily stand-up meetings to improve coordination of tasks, team communicating, and solve problems daily. The meeting was expected to have an effect on shared leadership, team orientation and learning.
- Code review to improve software quality, learning and increase redundancy.

The development team got higher scores on all factors compared to the maintenance team. The team prioritized to improve the problems with the highest potential for the team: inefficient sprint planning, variable ownership to project goals, and not solving process related problems in the retrospective. The following actions were suggested:

- Open space[1] sprint planning, to conduct sprint planning more efficiently. The sprint planning meetings in the team were dominated by specialists and long lasting. Using the open space process, the team members would suggest topics to discuss and then several discussions could happen in parallel in the same room. Team members are encouraged to walk between discussions. This action was expected to improve shared leadership and team orientation.
- Pair programming to improve team orientation. Making people to closely together constantly giving feedback could also improve shared decision-making and improve learning.
- Collocating the team in the same room, would improve communication and oversight, and improving team orientation.

[1] www.openspaceworld.org

5 Discussion

Now we return to our research question, "how to efficiently improve teamwork in agile software development?" We have shown results from using diagnosis with the team radar and action planning in a small and immature team and in a large and more mature team.

Both the teams perceived the diagnosing and the outcome as something they learned from, because it illuminated issues they had seen individually but not discussed within the team. It is not enough to do retrospectives if the team is not able to discuss the cause of the problems they are experiencing.

The cost associated with the improvement method reported in this article was perceived as low, with a short data collection period (interviews and observations), and little disturbance of the team. The feedback meeting where the team got concrete feedback and had the ability to discuss software process improvement measures, was the meeting taking most time. The teams stated that the radar produced a realistic and "spot on" analysis of the situation in the team. The method presented here, helped the companies improve, however, to use the team radar as a diagnosis instrument was not without challenges. Setting a score on the team radar was difficult, because the score is both subjective and imprecise. However, the main motivation for giving a score is to get a basis for discussion with the team. Also the score is discussed and verified by the team before an improvement program is suggested. Working with an instrument like the team radar should be seen as a start of a process, not as an end-mean in itself.

A question is then whether it would make more sense to have a more open approach to software process improvement, for example by basing improvement initiatives on the retrospective. There are two main differences in the approach reported in this article and an approach relying on retrospectives. First when using an external person, he or she gets more insight into the work of the team through interviews and observation. This might discover process related problems not reported in the retrospective, and give the team a better understanding of the problems. This is important to suggest the right measures to be taken. Second, since the team radar is based on the factors necessary for achieving self-management, the instrument gives more precision in identifying problems than what typically is identified in a retrospective. Redundancy for example, is a factor which is often mixed with learning, and a team might see problems but not relate them to root causes such as a lack of team orientation.

In the development team, as a larger and more mature team already experienced with process improvement, the diagnosis using a team radar led to more precise recommendations than they had experienced previously. In the maintenance team, one could argue that the results only confirmed what the team already knew. However it was not until the results from the team radar was discussed, that they were able improve their processes.

6 Conclusion and Further Work

This study indicates that process improvement, although a central concept in agile development is still hard to achieve. This study indicates that diagnosis using a specific instrument, the team radar, has an effect on action planning in teams.

This study has the following implications: The implication for theory is that there are positive indications that the team radar instrument identifies relevant challenges for agile software development teams. This form of diagnosing and action planning can be valuable in action research, and the diagnosis instrument can also be of use in case studies and ethnographic studies of teams.

The main implication for practice is that this study with two teams reveals that process improvement does not happen by itself even in agile methods, there needs to be effort invested to actively experiment with solutions.

Acknowledgements. We are grateful to participants in the two teams from the companies, who willingly shared their experience on teamwork and were willing to try out new practices. This project has been partially supported by the Research Council of Norway in the TeamIT project, through grant 193236/I40.

References

1. Aaen, I., Arent, J., Mathiassen, L., Ngwenyama, O.: A Conceptual MAP of Software Process Improvement. Scandinavian Journal of Information Systems 13, 81–101 (2001)
2. Lycett, M., Macredie, R.D., Patel, C., Paul, R.J.: Migrating agile methods to standardized development practice. Computer 36, 79–85 (2003)
3. Dybå, T., Dingsøyr, T.: Empirical Studies of Agile Software Development: A Systematic Review. Information and Software Technology 50, 833–859 (2008)
4. Nerur, S., Balijepally, V.: Theoretical Reflections on Agile Development Methodologies. Communications of the ACM 50, 79–83 (2007)
5. Salo, O., Abrahamsson, P.: An iterative improvement process for agile software development. Software Process: Improvement and Practice 12, 81–100 (2007)
6. Baskerville, R., Wood-Harper, A.T.: A critical perspective on action research as a method for information systems research. Journal of Information Technology 11, 235–246 (1996)
7. Nerur, S., Mahapatra, R., Mangalaraj, G.: Challenges of migrating to agile methodologies. Communications of the ACM 48, 72–78 (2005)
8. Okhuysen, G.A., Bechky, B.A.: Coordination in Organizations: An Integrative Perspective. Academy of Management Annals 3, 463–502 (2009)
9. Trist, E.: The evolution of socio-technical systems: a conceptual framework and an action research program, Ontario Quality of Working Life Centre, Toronto, Ontario (1981)
10. Kirkman, B.L., Rosen, B.: Beyond self-management: Antecedents and consequences of team empowerment. Academy of Management Journal 42, 58–74 (1999)
11. Morgan, G.: Images of Organizations. SAGE publications, Thousand Oaks (2006)
12. Pearce, C.L.: The future of leadership: Combining vertical and shared leadership to transform knowledge work. Academy of Management Executive 18, 47–57 (2004)
13. Hewitt, B., Walz, D.: Using Shared Leadership to Foster Knowledge Sharing in Information Systems Development Projects. In: Proceedings of the 38th Annual Hawaii International Conference on HICSS 2005 (2005)
14. Hoegl, M., Parboteeah, P.: Autonomy and teamwork in innovative projects. Human Resource Management 45, 67 (2006)
15. Emery, F., Thorsrud, E.: Democracy at work: the report of the Norwegian industrial democracy program. Martinus Nijhoff Social Sciences Division, Leiden (1976)
16. Basili, V.R.: Software development: a paradigm for the future. Presented at Computer Software and Applications Conference, COMPSAC 1989 (1989)

17. SEI, Capability Maturity Model ® Integration (CMMI) (2002)
18. Hansen, B., Rose, J., Tjørnehøj, G.: Prescription, description, reflection: the shape of the software process improvement field. International Journal of Information Management 24, 457–472 (2004)
19. Humphrey, W.S.: Managing the software process. Addison-Wesley, Reading (1989)
20. Hansen, B., Rose, J., Tjornehoj, G.: Prescription, description, reflection: the shape of the software process improvement field. International Journal of Information Management 24, 457–472 (2004)
21. Aaen, I.: Essence: Facilitating Agile Innovation. In: XP 2008, pp. 1–10. Springer, Heidelberg (2008)
22. Dingsøyr, T.: Postmortem reviews: Purpose and Approaches in Software Engineering. Information and Software Technology 47, 293–303 (2005)
23. Aaen, I., Börjesson, A., Mathiassen, L.: Navigating Software Process Improvement Projects. In: Baskerville, R., Mathiassen, L., Pries-Heje, J., DeGross, J. (eds.) Business Agility and Information Technology Diffusion. IFIP International Federation for Information Processing, vol. 180, pp. 53–71. Springer, Boston (2005)
24. Qumer, A., Henderson-Sellers, B.: A framework to support the evaluation, adoption and improvement of agile methods in practice. Journal of Systems and Software 81, 1899–1919 (2008)
25. Moe, N.B., Dingsøyr, T., Røyrvik, E.: Putting Agile Teamwork to the Test – A Preliminary Instrument for Empirically Assessing and Improving Agile Software Development. Presented at XP 2009, Pula, Italy (2009)