Chapter 11
Agile Project Management

Tore Dybå, Torgeir Dingsøyr, and Nils Brede Moe

Abstract Agile software development represents a new approach for planning and managing software projects. It puts less emphasis on up-front plans and strict control and relies more on informal collaboration, coordination, and learning. This chapter provides a characterization and definition of agile project management based on extensive studies of industrial projects. It explains the circumstances behind the change from traditional management with its focus on direct supervision and standardization of work processes, to the newer, agile focus on self-managing teams, including its opportunities and benefits, but also its complexity and challenges. The main contribution of the chapter is the four principles of agile project management: minimum critical specification, autonomous teams, redundancy, and feedback and learning.

11.1 Introduction

A software project can be seen as a collection of activities that create an identifiable outcome of value. In its simplest form, project management consists of planning, executing, and monitoring these activities (see Chap. 1). However, the high costs and failure rates of software projects continue to engage researchers and practitioners, and despite several advances, the effective management of software projects is still a critical challenge.

This challenge has led to extensive interest in agile software development in the past decade (Dingsøyr et al. 2012). A number of methods have emerged that describe practices for development phases at the team, project, and organizational

T. Dybå (✉) • T. Dingsøyr • N.B. Moe
SINTEF, Trondheim, Norway
e-mail: tore.dyba@sintef.no; torgeird@sintef.no; nilsm@sintef.no

G. Ruhe and C. Wohlin (eds.), *Software Project Management in a Changing World*,
DOI 10.1007/978-3-642-55035-5_11,

levels (Abramson et al. 2010). Scrum is the method that most clearly addresses software project management (Schwaber and Beedle 2001).

In agile software development, developers work in teams with customers that represent the system's users. The features to be implemented in each development cycle are jointly decided by the customer and the rest of the development team. Augustine et al. (2005) describes the role of the software project manager as one of facilitating and working with a team in making project-related decisions.

Our objective in this chapter is to provide software project managers with a set of principles for handling the complexity and uncertainty inherent in agile software projects. The rest of this chapter is organized as follows: Sect. 11.2 describes challenges and recent developments in software project management. Section 11.3 explains the role of self-managing software teams, while Sect. 11.4 discusses the leadership of such teams. Section 11.5 describes the importance of feedback and learning. Finally, Sect. 11.6 presents a set of principles for agile project management, while Sect. 11.7 concludes the chapter.

11.2 Software Project Management

Managing the unique and complex processes that constitute a project involves the implementation of specific management activities. In software development, as in most other businesses, there has been a tendency toward standardizing these activities by means of formalized, generic project management methodologies like, PRINCE2,[1] which was developed and championed by the UK government. Although there is a global conception of the project management phenomenon, there is no unified theory of project management (Garel 2013) or well-defined measures of project success (see Chaps. 2 and 5).

11.2.1 Traditional Project Management

Traditional project management largely derives from the linear structure and discrete, mechanical views of the systems engineering and quality disciplines of the 1950s and 1960s. Basically, traditional project management views development as a linear sequence of well-defined activities such as requirements, design, coding, and testing. It assumes that you have almost perfect information about the project's goal and expected solution. As a consequence, it does not easily accommodate for deviations in scope, schedule, or resources.

Hardware development seemed to fit well into the traditional approach. However, due to its intangible nature, software was not equally well understood and, as a

[1] www.prince-officialsite.com

consequence, software development did not fit well into the same approach. To counter this, the term "software engineering" was coined at the historic NATO conference in Garmisch-Partenkirchen in 1968 as a solution to these problems in software development, implying the need for software development to be based on the principles and practices seen in engineering.

Thus, the point of departure for most of the subsequent efforts in addressing the problems in software development has been to treat the entire task of software development as a process that can be managed through engineering methods. Hoare (1984), for example, considered the "rise of engineering" and the use of "mathematical proof" in software development as a promise to "transform the arcane and error-prone craft of computer programming to meet the highest standards of a modern engineering profession." Likewise, Lehman (1989) focused on reducing uncertainty in the development process through the "engineering of software."

Humphrey (1989) recognized that the key problems in software development are not technological, but managerial in nature (see also Chap. 1). Consequently, he developed a framework for managing and improving the software process, which was later known as "The Capability Maturity Model for Software" (CMM). Consistent with the views of software engineering, the CMM, and its successor CMMI, is rooted in the engineering tradition, emphasizing predictability and improvement through the use of statistical process control. Humphrey (1989) formulated his fundamental view in this way: "If the process is not under statistical control, sustained progress is not possible until it is."

As these examples from some of the most influential academic leaders within the software community show, software development and software project management are strongly rooted in the rationalistic traditions of engineering. Indeed, most of the writings to date can be seen to have antecedents in the industrial models devised by Frederic Winslow Taylor and Henry Ford, and also in the model of bureaucracy described by Max Weber.

Contemporary project management methodologies, like PRINCE2, are standardized, process-driven project management methodologies, which build on this engineering tradition and that contrast with reactive and adaptive methods such as Scrum. What many seem to forget, is that the acronym PRINCE stands for "PRoject IN Controlled Environment." It should not come as a surprise then that it does not fit equally well within the environment in which many, if not most, software projects operate.

11.2.2 Challenges of Software Project Management

Although there are several challenges with traditional project management principles, two are especially important for the management of software projects: complexity and uncertainty. Project complexity means that the many different actions and states of the software project and its environmental parameters interact, so the effects of actions are difficult to assess (Pich et al. 2002). In complex software

projects, an adequate representation of all the technological, organizational, and environmental states that might have a significant influence on the project's outcome of value, or of the causal relationships, is simply beyond the capabilities of the project team.

Most of the classic problems of developing software derive from this essential complexity and its exponential increase with size; for example, it is estimated that for every 25 % increase in problem complexity, there is a 100 % increase in complexity of the software solution (Woodfield 1979). A further challenge is that the information needed to understand most software problems depends upon one's idea for solving them. The kind of problems that software projects deal with tend to be unique and difficult to formulate and solutions tend to evolve continually as developers gain a greater appreciation of what must be solved (Nerur and Balijepally 2007).

Adding to the complexity of the problem and its solution is the fast-changing and highly uncertain environment, for example, market turbulence and changes in customer requirements and project goals. It is necessary therefore to accept that our assumptions and predictions about future events will, by nature, be uncertain. When managing software projects, we need to be extremely cautious of extrapolating past trends or relying too heavily on past experience. Trends peter out, and the future is full of unexpected developments as well as unpredictable human behavior. The greater the uncertainty inherent in a project, the more the team have to move from traditional approaches that are based on a fixed sequence of activities to approaches that allow to redefine the activities—or even the structure of the project plan—in midcourse (De Meyer et al. 2002). Therefore, as the project complexity and uncertainty increase, managers need to go beyond traditional risk management; adopting roles and techniques oriented less toward planning and more toward flexibility and learning.

11.2.3 From Traditional to Agile Project Management

The position taken in this chapter regarding software project management is strongly influenced by socio-technical theory (Trist 1981). Its central conception is that organizations are both social and technical systems, and that the core of the software organization is represented through the interface between the technical and human (social) system. From an engineering perspective, however, the world is composed of problems whose existence is distinct from the methods, tools, and practices of software development. The technical rationality behind this worldview emphasizes "objective truths" and global "best practices" at the expense of local context and expertise. An important aspect of socio-technical theory, however, is the belief that there may be many optimal solutions—or best ways—to a specific problem, since the "joint optimization" of a particular technical and human system can be implemented in several ways that can be equally efficient. We therefore

Fig. 11.1 The work environment of an agile development team

reject the assumption that complexity, uncertainty, and change can be controlled through a high degree of formalization

At its core, *agile project management* is about managing the impact of complexity and uncertainty on a project, recognizing

- The need for a dramatically shorter time frame between planning and execution
- That planning an action does not provide all the details of its implementation
- That creativity and learning are necessary to make sense of the environment

Agile project management is based on the same principles found in the Agile Manifesto.[2]

Therefore, unlike the linear sequence of well-defined activities of traditional project management, agile project management is characterized by short cycles of iterative and incremental delivery of product features and continuous integration of code changes. Agile project management introduces changes in management roles as well as in practices. Scrum, for example, defines three roles in software projects: development team members, a facilitator, and a product owner. A typical work environment for an agile team is shown in Fig. 11.1.

The task of the facilitator is to organize meetings of the development team and to make sure that the team addresses any obstacles they encounter. The task of the product owner is to prioritize what is to be developed. Apart from that, the team should be self-managed. In practice, however, many companies also appoint a project manager to assist a product owner in working on requirements and to handle other matters than those directly related to software development, such as internal and external reporting.

[2] http://agilemanifesto.org/

However, the introduction of agile development does not change the fundamental knowledge required to develop software, but it does change the nature of collaboration, coordination, and communication in software projects. Moving from traditional to agile project management implies a shift in focus from extensive *up-front planning* to the crucial decisions that are made during the execution of the project. Most importantly, moving from traditional to agile development implies dealing with complexity and unpredictability by relying on people and their creativity rather than on standard processes (Dybå 2000; Conboy et al. 2011), and thus moving from command and control to shared decision-making and self-management in software teams.

11.3 Self-Managing Software Teams

Teams are the fundamental organizational unit through which software projects are executed. A team structure brings business and process knowledge together with design and programming skills. However, three challenges characterize the effectiveness of software teams (Faraj and Sambamurthy 2006). First, the expertise required for the completion of software tasks is distributed across team members who must find effective ways of collaboration, knowledge sharing, and problem solving. Second, software projects need a combination of formal and informal modes of control with appropriate expertise in knowing how to exercise the appropriate combinations of control strategies during the execution of the project. Finally, software projects are characterized by varying levels of task uncertainty and coordination challenges. As a result, the software teams must be capable of dealing with the resulting ambiguity of their project tasks.

In accordance with contemporary perspectives, we conceptualize the software team as embedded in a multilevel system of individual-, team-, and organizational-level aspects (Kozlowski and Ilgen 2006; Moe et al. 2009). This conceptualization is important in our effort to make actionable recommendations on agile software management.

With the introduction of agile software development, self-managing software teams have become widely recommended. While organizing software development in self-managing teams has many advantages such as increased productivity, innovation, and employee satisfaction, it is not enough to put individuals together and expect that they will automatically know how to work effectively in such teams. Succeeding with managing agile teams requires a full understanding of how to create and maintain self-managing teams.

One can argue that leading self-managing teams is more challenging than leading traditional teams, because the project manager needs to enable shared leadership (the opposite of centralized leadership), shared decision-making, shared mental models, and a constant learning and improvement process. This takes time. What makes the leadership of such teams even more difficult is the fact that software development teams are typically formed anew for each project, depending

on project requirements and who is available (Constantine 1993). It is extremely rare for an entire team to move from one project to another.

Self-managing teams are also known as autonomous or empowered teams. While self-managing teams represent a radically new approach to planning and managing software projects, the notion of self-management is not new; research in this area has been conducted since Eric Trist and Ken Bamforth's study of self-regulated coal miners in the 1950s (Trist and Bamforth 1951).

Self-management can be understood as a strategy for learning and improving a software team itself since it can directly influence team effectiveness, improvement work, and innovation. Self-management has also been found to result in more satisfied employees, lower turnover, and lower absenteeism (Cohen and Bailey 1997). Others also claim that self-managing teams are a prerequisite to the success of innovative projects (Takeuchi and Nonaka 1986), especially the innovative software projects (Hoegl and Parboteeah 2006). Furthermore, having team members cross-trained to do various jobs increases functional redundancy, and thus the flexibility of the team in dealing with personnel shortages. Even though there are several studies on the benefits of self-managing teams, there is substantial variance in research findings regarding the consequences of such teams on such measures as productivity, turnover, and attitudes (Guzzo and Dickson 1996).

Self-managing teams offer potential advantages over traditionally managed teams because they bring decision-making authority to the level of operational problems and uncertainties and thus increase the speed and accuracy of problem solving. Companies have implemented such teams to reduce costs and to improve productivity and quality. However, effective self-managing units cannot be created simply by exhorting democratic ideals, by tearing down organizational hierarchies, or by instituting one-person-one-vote decision-making processes (Hackman 1986). Hackman identified five general conditions that appear to foster and support self-management:

- Clear, engaging direction
- An enabling performing unit structure
- A supportive organizational context
- Available, expert coaching
- Adequate resources

To succeed with creating and maintain a self-managing agile team, the project manager must enable all these conditions.

Understanding the different levels of autonomy is also important for being able to succeed as an agile project manager. An agile team needs autonomy on both the team and the individual levels (Moe et al. 2008). The project manager must ensure that

- The team has authority to define work strategies and processes, project goals, and resource allocation
- All team members jointly share decision authority (what group tasks to perform and how to carry them out)
- A team member must have some freedom in carrying out the assigned task

The conflict between individual and team autonomy is one main reason why it is challenging to establish well-functioning agile teams. If individuals are independent and mostly focus on their own schedule and implementation of their own tasks, there will be less interaction between the group members, which will threaten the teamwork effectiveness. However the self-managing team may end up controlling group members more rigidly than they do under traditional management styles, which will reduce the motivation for the individual team member. The question is then: How can an agile project manager balance team level autonomy and individual level autonomy in agile software teams? This is especially challenging when development is done in market-driven agile projects with fixed scope and deadlines.

11.4 Team Leadership

Most models of team effectiveness recognize the critical role of team leadership. However, examining the extensive literature on leadership theories is beyond the scope of this chapter. Still, a relatively neglected issue in the current literature is what leaders should actually be doing to enhance team effectiveness (Kozlowski and Bell 2003). Many leadership theories focus on leading individuals rather than leading in a team context. In this section, we examine the functional role of team leaders and discuss leadership and decision-making in the context of self-managing teams. A recent focus group study on agile practitioners shows that planning, shielding the team from interruptions, agreeing on a work process, ensuring adequate resources, and setting up a working technical infrastructure are seen as important aspects of team leadership (Dingsøyr and Lindsjørn 2013).

In a self-managing team, members have responsibility not only for executing the task but also for monitoring, managing, and improving their own performance (Hackman 1986). Furthermore, leadership in such teams should be diffused rather than centralized (Morgan 2006). Shared leadership can be seen as a manifestation of fully developed empowerment of a team (Kirkman and Rosen 1999). When the team and the team leaders share the leadership, it is transferred to the person with the key knowledge, skills, and abilities related to the specific issues facing the team at any given moment (Pearce 2004). While the project manager maintains the leadership for project management duties, the team members lead when they possess the knowledge that needs to be shared during different phases of the project (Hewitt and Walz 2005).

11.4.1 Shared Decision Making

Product- and project-level decisions in a software company can be considered at the strategic, tactical, and operational levels (Aurum et al. 2006). In traditional development decision-making is governed by the hierarchical command and control

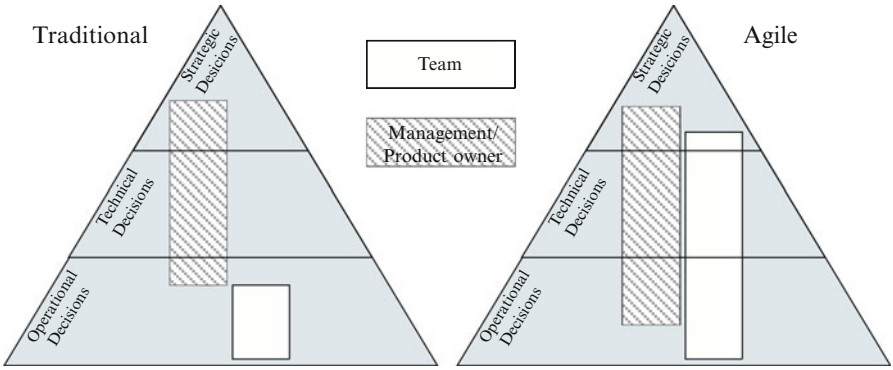

Fig. 11.2 Traditional and agile models of decision-making (Moe et al. 2012)

structure, while in agile development team empowerment and shared decision-making is encouraged at all levels (Fig. 11.2). In most organizations, you will find a mixture of these two ways of making decisions. In an agile product company, strategic decisions are primarily related to product and release plans, which may require creativity and opportunistic inputs, and should be based on an accurate understanding of the current business process and a detailed knowledge of the software product. Tactical decisions in such companies involve the project management view, where the aim is to determine the best way to implement strategic decisions, that is, to allocate the resources. On the other hand, operational decisions in an agile company are about implementation of product features and the process of assuring that specific tasks are carried out effectively and efficiently (Moe et al. 2012).

Adaptability is essential in agile teams since strategic decisions are made incrementally while important tactical and operational decisions are delayed as much as possible, in order to allow for a more flexible response to last minute feedback from the market place. Because the self-managing team is responsible for solving operational problems and uncertainties, this increases the speed and accuracy of problem solving, which is essential when developing software.

While there are several benefits with shared decision-making in agile teams, there also exist some challenges. First, the shared decision-making approach, which involves stakeholders with diverse backgrounds and goals, is more problematic than the traditional approach where the project manager is responsible for most of the decisions (Nerur et al. 2005). Also, despite the benefits of shared decision-making, cohesion has been indicated as a source of ineffective or dysfunctional decision-making; perhaps the most noted problem associated with team cohesion is groupthink. Finally, it is important to understand that 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. The challenge is to understand which team member is supposed to be involved in which decisions.

In agile software development, one important forum for shared decision-making is the standup meeting because it is about coordinating and planning the daily work.

The meeting is supposed to be short, and the purpose of the meeting is to improve communication, highlight and promote quick decision-making, and identify and remove impediments.

The daily meeting is a place where the agile software team members use their experience to make decisions in a complex, dynamic, and real-time environment. To understand the effect of decision making in such a complex environment with time pressure, the theory of naturalistic decision making (NDM) (Meso et al. 2002) is useful. NDM postulates that experts can make good decisions under difficult conditions, such as time pressure, uncertainty, and vague goals, without having to perform extensive analyses and compare options. The experts are able to do so by employing their experience to recognize problems that they have previously encountered and for which they already developed solutions. Experts use their experience to form mental simulations of the problem currently being encountered and use these simulations to suggest appropriate solutions.

There are certain implications of viewing daily meetings as an NDM process. First, the agile project manager needs to make sure that employees are trained to develop domain-specific expertise and collaborative teamwork skills. Second, because NDM relies on highly trained experts, the agile project manager needs to make sure the team consists of experts, not novices. If there are novices in the team, the agile project manager needs to determine how to move the novices through the stages until they become experts. Third, to make teams perform effective decision-making processes in such meetings, the project manager needs to make sure the team members have developed a shared mental model; that is, they must have a shared understanding of who is responsible for what and of the information and requirements needed to solve the tasks (Lipshitz et al. 2001).

11.4.2 Escalation of Commitment

One major source of error in decision-making is escalation of commitment (Stray et al. 2012). Escalating situations happen when decision-makers allocate resources to a failing course of action (Staw 1976). It is a general phenomenon that is particularly common in software projects due to their complex and uncertain nature. Keil et al. (2000), for example, found that 30–40 % of all software projects experience escalation of commitment.

Several studies have shown that decision-makers tend to invest additional resources in an attempt to justify their previous investments (Bazerman et al. 1984). Because groups have the capacity of employing multiple perspectives when making decisions, one might believe that escalating commitment situations should occur less frequently in agile teams than in traditional teams. However, several studies show that when having group decision-making, escalating tendencies will occur more often and will be more severe than in individual decision-making due to group polarization and conformity pressures (Whyte 1993).

To avoid situations of escalating commitment in agile projects, it is important to make sure that the team meetings do not become a place for defending decisions (Stray et al. 2012). Not only do the teams need to watch their internal process, they also need to consider which non-team members are allowed to participate or observe the team meetings. If team members start to defend their decisions or give detailed reports of what they have done because people outside the team are present in, for example, the daily meetings, we advise that these people outside the team do not participate on a regular basis.

Early signs of escalation in, for example, the daily meeting such as rationalizing continuation of a chosen course of action, and when team members start giving detailed and technical descriptions of what they have done since last meeting, must be taken seriously. Further, when the team becomes aware of the signs of escalating commitment, this needs to be addressed in the retrospective meetings (see next section).

11.5 Feedback and Learning

Agile software development is a type of knowledge work (see Chap. 7) where feedback and learning is particularly important. In order to adapt to changes in technology and customer requirements and to reduce the risk in development, agile development methods rely on frequent feedback loops both within the development team and with external stakeholders. The focus on having a "shippable product" leads to feedback on technical problems, and the demonstration of the product at the end of iterations leads to feedback on the developed functionality. The feedback represents opportunities for learning, which can lead to changes in product and personal skills, as well as changes in the development process.

With the focus in agile development on "working software" and "individuals and interactions," knowledge is managed in a very different manner than in traditional software development. Traditional development has often focused on managing explicit knowledge in the form of written lessons learned in knowledge repositories and documented procedures in electronic process guides (see Chap. 7). Agile methods focus on managing knowledge orally, which means that dialogue is the main method of transfer (Bjørnson and Dingsøyr 2008). Von Krogh et al. write that "it is quite ironic that while executives and knowledge officers persist in focusing on expensive information-technology systems, quantifiable databases, and measurement tools, one of the best means for knowledge sharing and creating knowledge already exists within their companies. We cannot emphasize enough the important part conversations play" (von Krogh et al. 2000).

A well-known theory of learning that focuses on feedback is Argyris and Schön's theory of single- and double-loop learning (Argyris and Schön 1996). Double-loop learning is distinguished from single-loop learning in that it concerns the underlying values. If an agile team repeatedly changes practices without solving their problems, this is a sign that they have not understood the underlying causes of

their problem and is practicing single-looped learning. Lynn et al. (1999) argue that learning has a direct impact on cycle time and product success, and have identified two factors that are central for learning: capturing knowledge, and a change in behavior based on the captured knowledge. Practices that must be in place for this to happen are, among others, recording and reviewing information, have goal clarity, goal stability, and vision support.

There are, however, often challenges in agile teams to make use of opportunities for learning. A study by Stray et al. (2011) reports that many teams spend little time reflecting on how to improve how they work and they do not discuss obvious problems. Some of the teams that carry out regular retrospective meetings struggle to convert their analysis into changes in action. Among those who try to remedy identified problems actively, several give up after seeing little change.

Learning is challenging but crucial. A stream of research has established that teams who have shared mental models about product, tasks, and process work more effectively. Further, developing overlapping knowledge, sometimes referred to as knowledge redundancy, is critical in turbulent environments where people need to work on tasks assigned by priority rather than competence of team members. In the following, we discuss some of the main arenas for feedback and learning in agile development: the project kick-off and retrospectives after iteration and release.

11.5.1 Agile Project Kick off

Kick-off is one of the most used tools of project management (Besner and Hobbs 2008). Typical activities in a kick-off meeting are describing a vision for the project, establishing roles, project stakeholders, and planning the project. A vision or overall goals of the project will usually be defined by the customer, what is referred to as the product owner in Scrum. Further, in agile methods, the team is seen as a self-managing team, with one person facilitating the work of the team. Thus, the only internal roles are the team facilitator and team members. However, companies often also appoint a project manager, especially in multi team projects. The project stakeholders are usually represented by one product owner, but can also be other people from the customer who have an interest in the product to be developed, or other development projects, for example, when sharing a common technical infrastructure. A picture from a project kick-off is shown in Fig. 11.3.

As for planning the project, one important decision is the duration of an iteration. If there are frequent changes in customer requirements or technology, this calls for shorter iterations, while a more stable environment calls for longer iterations. Normally an agile team will make a rough plan for several iterations and a detailed plan for the next one. The detailed plan can be made at the kick-off by making a product owner give priorities to the set of features that is to be developed. The features are estimated, for example, using the technique planning poker, which facilitates a discussion between team members on what tasks must be performed to develop a feature. The team then commits to what they will be able to deliver in the

Fig. 11.3 Project kick off with development team, team facilitator, and customer responsible

first iteration. The "plan" for the team is then a list of prioritized features, and who is to perform the tasks of developing the features is decided on during the iteration.

What is important in the kick-off meeting to enable feedback and learning? From studies of shared mental models, we know that teams need to establish shared knowledge on a number of areas to function effectively. Shared mental models comprise knowledge of the tasks, technology, team members' skills, and interactions. The planning poker technique is one way to improve shared mental models. Estimation discussions can provide knowledge of tasks at hand, the technology used in development, as well as demonstrating team member skills (Fægri 2010).

Planning poker is carried out as follows: Every individual is given a set of playing cards with values loosely in a Fibonacci sequence, usually 0, 1, 2, 3, 5, 8, 13, 20, 40, and ∞. For each task, individuals decide on a card that represents the amount of work; this can be in number of hours or relative to a standard task. All team members show their cards, and the person with the highest and lowest estimates is asked to explain their reasoning. The process is repeated until consensus, or if consensus is unlikely a number is set based on majority vote or average of votes. If there is much divergence, it might also be necessary to decompose a task into smaller tasks that are easier to estimate. See Chap. 3 for a further discussion of estimation, and there are also some studies available on the use of planning poker as an estimation technique (Molokken-Ostvold et al. 2008).

Finally, that everyone has a clear image of team interaction is accomplished by having clear work processes. Agile methods are simple and easy to remember, which makes it easy to function as a shared mental model.

11.5.2 The Retrospective

A retrospective (or postmortem review (Birk et al. 2002) or post-iteration workshop (Outi 2006)) is a collective learning activity after an iteration or release (Dingsøyr 2005). The main motivation is to reflect on what happened in order to improve future practice—for the individuals that have participated in the project and possibly also for the organization as a whole.

Researchers in organizational learning use the term "reflective practice" (Dybå et al. 2014), which is "the practice of periodically stepping back to ponder on the meaning to self and others in one's immediate environment about what has recently transpired. It illuminates what has been experienced by both self and others, providing a basis for future action" (Raelin 2001). This involves uncovering and making explicit results of plans, observation, and achieved practice. It can lead to understanding of experience that has been overlooked in practice. Kerth argues that a retrospective can help members of a community to understand the need for improvement and motivate them to change. The retrospective helps the community to become "master of its software process" (Kerth 2001). In addition, retrospectives are claimed to foster learning, growth, and participant maturity, and provides an opportunity to celebrate success. Derby and Larsen further suggest that retrospectives lead to improved productivity, capability, quality, and capacity; the purpose is "whole-team learning" (Derby and Larsen 2006).

A typical agile retrospective will be conducted with activities to gather data, generate insight, and make decisions (ibid). To gather data, exercises such as plotting important events on a timeline or just brainstorming on "what went well" and "what could be improved" are typical. Insights are generated by analysis of the material, through use of fishbone diagrams, structuring of data and prioritization (see Fig. 11.4). Decisions about changes are made on this basis and are planned as tasks in the next iteration.

Although retrospectives today is a very common practice, there has been little research on this topic. Most works concentrate on describing approaches to conduct retrospectives, with little focus on the effects. However, in a survey on essential practices in research and development companies, "learning from post-project audits" were found to be one of the most promising practices to yield competitive advantage (Menke 1997).

Kransdorff (1996) criticizes postmortems because people participating do not have an accurate memory, which can lead to disputes. He suggests collecting data during the project, for example, through short interviews, in an effort to get more objective material.

Fig. 11.4 A retrospective in a development team with a group of developers structuring the results of a brainstorming session

11.5.3 Visualizing Project Status

Many teams use visual boards, kanbans,[3] or "information radiators" as a central element for collaboration, coordination, and communication (Sharp and Robinson 2010). A board usually displays tasks on cards, and the placement of cards on a board shows their status. Teams have found that such boards make meetings efficient. Participants point at cards on the board to show what team members work on, and the board shows progress in the project.

Physical artifacts are easy to refer to, easy to annotate, and hard to ignore (Sharp et al. 2006). A physical board makes it easier to limit the amount of information unlike an electronic system, which is often the alternative. Such boards can help giving teams a shared mental model of the project status, importance of tasks, and how ready the product is for delivery.

A visual board can be set up quickly by placing a board in a relevant location, deciding on how to organize the board, and supplying cards to put on the board. Find a location, which is visible both for the development team and for others who have interest in the work of the team. The board could be placed with other visual information the team is using, for example, a burndown[4] chart, which shows the remaining work in this phase.

The board should show important information about status and progress of the work of the team. This can be done by dividing the board into relevant phases that

[3] A kanban is a visual card system for organizing production according to demand, central in lean production.

[4] A burndown chart shows the estimated remaining work in an iteration, and is updated daily when teams use the Scrum development process.

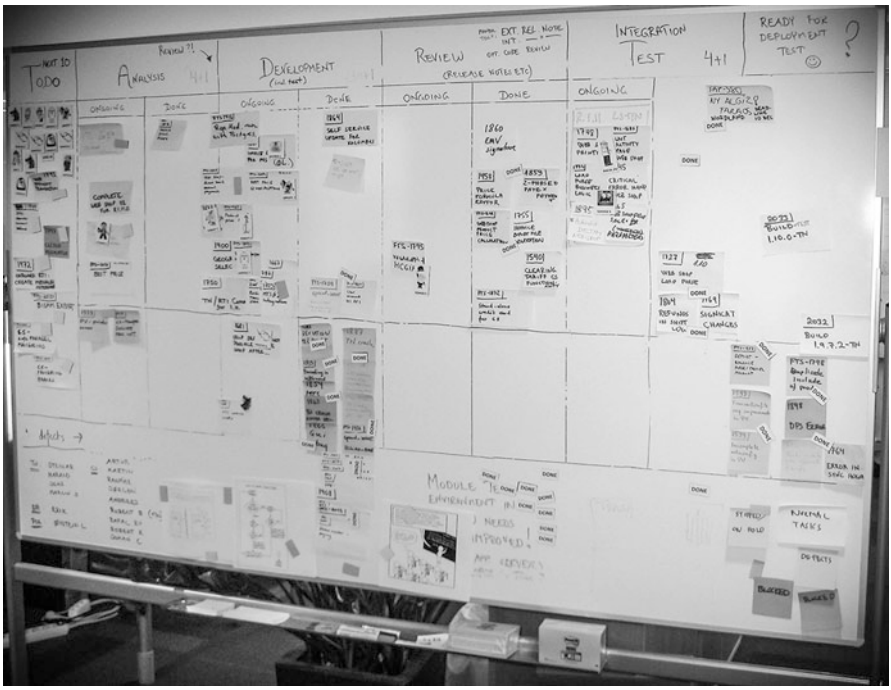

Fig. 11.5 Example visual board with areas for tasks "todo," "analysis," "development," "review," "integration test," and "ready for deployment test"

work tasks go through. Typical phases include "to do," "analysis," "development," "review," "integration test," and "ready for deployment test," as shown in Fig. 11.5. If your team has particular problems, for example, if it is unclear for developers whether a task is completed or not, you can add a phase for checking that either another developer or an external person agrees that the task is completed. Some also choose to mark problems in the project either through putting tasks in an area for problems or by marking tasks with a different color.

Physical artifacts like the card represent tokens of responsibility, and moving artifacts have been found to give more insight than electronic manipulation tools (Sharp et al. 2006), which is the alternative many teams use. A visual board makes it easy to discover common problems in a project like: tasks do not get completed, important tasks are not done, and if too many tasks are started at the same time.

11.6 Principles of Agile Project Management

A fundamental property of software is its nonphysical form; software code is essentially a large set of abstract instructions possessing unlimited complexity, flexibility, and revisability. Software exhibits nonlinear behavior and does not

Table 11.1 Principles of agile project management

Minimum critical specification	No more should be specified than is absolutely essential and critical to overall success
Autonomous teams	Autonomous teams are responsible for managing and monitoring their own processes and executing tasks
Redundancy	Team members should be skilled in more than one function
Feedback and learning	Feedback and learning are integral to project execution and the project's interaction with the environment

conform to laws of nature. One consequence is that it is inherently hard to build models of software that allow accurate reasoning about the system's qualities (Fægri et al. 2010). Agile project management addresses these basic properties of software and breaks away from the linear sequence of well-defined activities of traditional project management. It shifts focus from up-front planning to execution. In doing so, agile project management moves from traditional command and control structures to shared decision-making, self-management, and learning in software teams to deal with the complexity and unpredictability of the problem-solving activities of software projects.

Based on our extensive experience and studies of a multitude of agile projects during the last decade (Moe et al. 2009, 2010; Moe et al. 2012; Dybå and Dingsøyr 2008; Šmite et al. 2010; Dingøyr et al. 2012; Dingsøyr et al. 2010; Dybå 2011), we offer the following set of socio-technical principles of agile project management (see Table 11.1).

11.6.1 Minimum Critical Specification

This principle has two aspects; the first is that no more should be specified than is absolutely essential; the second requires that the team identify what is critical to overall success. This means that the system requirements should be precise about what has to be done, but not about how to do it, and that the use of rules, standards, and predefined procedures is kept to an absolute minimum. Focus should be on the larger system requirements and boundary conditions, leaving as many design decisions as possible to those closest to the work.

Understanding "the problem" that the system is intended to address is one of the keys to project success. Therefore, this principle is oriented toward the analysis and problem understanding that will help the project's stakeholders to focus on the nature of the overall problems and issues and come to some agreement about what these really are. It will also help the software team to understand the problems—rather than what they perceive as being the "problem"—the system is supposed to solve.

Additionally, complex and turbulent environments require software projects to be highly adaptable. Thus, specifying more than is needed closes options that should be kept open for as long as possible.

Successfully dealing with this principle requires that the project establishes shared mental models about the problem and its solution, as well as about tasks, technology, team member skills, and interactions. The project's kick-off meeting is crucial for achieving this.

11.6.2 Autonomous Team

This principle is based on the premise that autonomous, or self-managing, teams are a prerequisite for the success of innovative software projects. Such teams offer potential advantages over traditionally managed teams because they bring decision-making authority to the level of operational problems and uncertainties and thus increase the speed and accuracy of problem solving.

Members of autonomous teams are responsible for managing and monitoring their own processes and executing tasks. They typically share decision authority jointly, rather than having a centralized decision structure where one person makes all the decisions or a decentralized decision structure where team members make independent decisions.

However, there are important individual and organizational barriers and challenges to successfully applying autonomous teams in software development (Moe et al. 2009). Misalignment between team structure and organizational structure can be counterproductive, and attempts to implement autonomous teams can cause frustration for both developers and management. Shared resources, organizational control, and specialist culture are the most important barriers that need to be effectively dealt with in order to succeed.

Furthermore, autonomy at the team level may conflict with autonomy at the individual level; when a project as a whole is given a great deal of autonomy, it does not follow that the individual team members are given high levels of individual autonomy. It is a danger, therefore, that the self-managing team may end up controlling team members more rigidly than they do under traditional management styles. Thus, it is imperative to ensure that individual developers have sufficient control over their own work and over the scheduling and implementation of their own tasks.

For autonomous teams to thrive, it is thus necessary to build trust and commitment in the whole organization, avoiding any controls that would impair creativity and spontaneity. The team's need for continuous learning, not the company's need for control, should be in focus. So, make sure that both the organization and the teams know and respect the project's objective.

11.6.3 Redundancy

This principle is concerned with the overlap in individuals' knowledge and skills in order to create common references for people's creation of new knowledge; as the level of redundancy increases within the team, individuals will find it easier to share new knowledge and the project will be able to coordinate its work more effectively Therefore, this principle implies that each member of the team should be skilled in more than one function so that the project becomes more flexible and adaptive, which allows a function to be performed in many ways utilizing different people.

Having such redundancy, with team members cross-trained to do various jobs, increases the project's functional redundancy and thus the flexibility of the team in dealing with personnel shortages. Redundancy is also critical in turbulent environments where people need to work on tasks assigned by priority rather than the competence of team members.

A particular challenge, however, is that individual specialization, high levels of proficiency, and the ability to solve more complex problems are often more important motivations for people than to seek overlapping knowledge. It is essential, therefore, with a greater focus on redundancy at the organizational level surrounding the project; rather than viewing redundancy as unnecessary and inefficient, the organization must appreciate both generalists and specialists to build redundancy into its projects.

11.6.4 Feedback and Learning

Without feedback and learning, agile project management is not possible. The focus on project execution rather than on up-front planning in agile projects, leads to an intertwinement of learning and work, and of problem specification and solution. Viewing the software project as an open system that is continuously interacting with its environment also points to the importance of feedback and learning.

The complexity and unpredictability of software problems are typical of "wicked" problems (Rittel and Webber 1973; Yeh 1991), which are difficult to define until they are nearly solved. For such problems, requirements cannot be completely specified until most of the system is built and used. At the same time, the system cannot be built without specifying what is to be built. Furthermore, the problem is never really solved as improvements can always be made.

To deal with and manage software problems, therefore, the activities of requirements, design, coding, and testing have to be performed in an iterative and incremental way, which focuses on ongoing improvement of output value rather than on single delivery. The project should allow overlapping and parallel activities in a series of steps, making feedback and continual learning an internalized habit to reach a desirable result.

Together, these principles lay the foundation for successfully planning, executing, and monitoring the activities of a software project while allowing openness to define the details in each individual case according to the project's specific context.

11.7 Conclusions

The principles of agile project management have the potential to provide organizations and systems with emergent properties. However, organizations should be cautious in embracing these principles or in integrating them with existing practices. Agile management methods are ideal for projects that exhibit high variability in tasks, in the skills of people, and in the technology being used. They are also appropriate for organizations that are more conducive to innovation than those built around bureaucracy and formalization. Software organizations should, therefore, carefully assess their readiness before treading the path of agility.

The challenge in managing agile software projects is to find the balance between upfront planning and learning. Planning provides discipline and a concrete set of activities and contingencies that can be codified, executed, and monitored. Learning permits adapting to unforeseen or chaotic events. The two require different management styles and project infrastructure. Projects with low levels of complexity and uncertainty allow more planning, whereas projects with high levels of complexity and uncertainty require a greater emphasis on learning. Openness to learning is new to many software companies. But it is obvious from the many spectacular project failures that the time has come to rethink some of the traditions in software project management.

Agile project management has currently caught interest for small and co-located projects. However, in the future, agile project management might also solve some of the important challenges facing large-scale and global projects (see Chap. 12). The issues raised in this chapter are instrumental in making this move from traditional to agile project management; or in the words of Louis Pasteur: "chance favors only the prepared mind."

References

Abramson P, Oza N, Siponen MT (2010) Agile software development methods: a comparative review. In: Dingsøyr T, Dybå T, Moe NB (eds) Agile software development. Current research and future directions. Springer, Berlin, pp 31–59

Argyris C, Schön DA (1996) On organizational learning II: theory method and practise. Addison Wesley, Reading, MA

Augustine S, Payne B, Sencindiver F, Woodcock S (2005) Agile project management: steering from the edges. Commun ACM 48(12):85–89

Aurum A, Wohlin C, Porter A (2006) Aligning software project decisions: a case study. Int J Softw Eng Knowl Eng 16(6):795–818

Bazerman MH, Giuliano T, Appelman A (1984) Escalation of commitment in individual and group decision making. Organ Behav Hum Perform 33:141–152
Besner C, Hobbs B (2008) Project management practice, generic or contextual: a reality check. Proj Manage J 39:16–33
Birk A, Dingsøyr T, Stålhane T (2002) Postmortem: never leave a project without it. IEEE Softw 19(3):43–45, Special issue on knowledge management in software engineering
Bjørnson FO, Dingsøyr T (2008) Knowledge management in software engineering: a systematic review of studied concepts and research methods used. Info Softw Technol 50(11):1055–1168. doi:10.1016/j.infsof.2008.03.006
Cohen SG, Bailey DE (1997) What makes teams work: group effectiveness research from the shop floor to the executive suite. J Manage 23(3):239–290
Conboy K, Coyle S, Wang X, Pikkarainen M (2011) People over process: key challenges in agile development. IEEE Softw 28(4):48–57
Constantine LL (1993) Work organization: paradigms for project management and organization. Commun ACM 36(10):35–43
De Meyer A, Loch CH, Pich MT (2002) Managing project uncertainty: from variation to chaos. MIT Sloan Management Review Winter 2002:60–67
Derby E, Larsen D (2006) Agile retrospectives: making good teams great. The Pragmatic Bookshelf, Raleigh, NC
Dingsøyr T (2005) Postmortem reviews: purpose and approaches in software engineering. Info Softw Technol 47(5):293–303
Dingsøyr T, Lindsjørn Y (2013) Team performance in agile development teams: findings from 18 focus groups. In: Baumeister H, Weber B (eds) Agile processes in software engineering and extreme programming, vol 149. Springer, Berlin, pp 46–60
Dingsøyr T, Dybå T, Moe NB (2010) Agile software development: current research and future directions. Springer, Berlin
Dingsøyr T, Nerur S, Balijepally V, Moe NB (2012) A decade of agile methodologies: towards explaining agile software development. J Syst Softw 85(6):1213–1221. doi:10.1016/j.jss.2012.02.033
Dybå T (2000) Improvisation in small software organizations. IEEE Softw 17(5):82–87
Dybå T (2011) Special section on best papers from XP2010. Info Softw Technol 53(5):507–508
Dybå T, Dingsøyr T (2008) Empirical studies of agile software development: a systematic review. Info Softw Technol 50(9–10):833–859. doi:10.1016/j.inf-sof.2008.01.006
Dybå T, Maiden N, Glass R (2014) The reflective software engineer: reflective practice. IEEE Softw 31(4):32–36
Fægri TE (2010) Adoption of team estimation in a specialist organizational environment. In: Sillitti A, Martin A, Wang X, Whitworth E (eds) 11th international conference on agile software development, Trondheim, Norway, 1–4 June 2010. Springer, pp 28–42
Fægri TE, Dybå T, Dingsøyr T (2010) Introducing knowledge redundancy practice in software development: experiences with job rotation in support work. Info Softw Technol 52(10):1118–1132
Faraj S, Sambamurthy V (2006) Leadership of information systems development projects. IEEE Transact Eng Manage 53(2):238–249
Garel G (2013) A history of project management models: from pre-models to the standard models. Int J Proj Manage 31(5):663–669
Guzzo RA, Dickson MW (1996) Teams in organizations: recent research on performance and effectiveness. Annu Rev Psychol 47:307–338
Hackman JR (1986) The psychology of self-management in organizations. In: Pallack MS, Perloff RO (eds) Psychology and work: productivity, change, and employment. American Psychological Association, Washington, DC
Hewitt B, Walz D (2005) Using shared leadership to foster knowledge sharing in information systems development projects. In: Walz D (ed) Proceedings of the 38th Hawaii international conference on system sciences (HICCS), pp 1–5

Hoare CAR (1984) Programming: sorcery or science? IEEE Softw 1(2):5–16
Hoegl M, Parboteeah P (2006) Autonomy and teamwork in innovative projects. Hum Resour Manage 45(1):67
Humphrey WS (1989) Managing the software process. Addison-Wesley, Reading, MA
Keil M, Mann J, Rai A (2000) Why software projects escalate: An empirical analysis and test of four theoretical models. MIS Q 24(4):631–664
Kerth NL (2001) Project retrospectives: a handbook for team reviews. Dorset House Publishing, New York
Kirkman BL, Rosen B (1999) Beyond self-management: antecedents and consequences of team empowerment. Acad Manage J 42(1):58–74
Kozlowski SWJ, Bell BS (2003) Work groups and teams in organizations In: Borman WC, Ilgen DR, Klimoski RJ (ed) Handbook of psychology (vol 12): industrial and organizational psychology. Wiley-Blackwell, New York, pp 333–375
Kozlowski SWJ, Ilgen DR (2006) Enhancing the effectiveness of work groups and teams. Psychol Sci Public Inter 7:77–124
Kransdorff A (1996) Using the benefits of hindsight - the role of post-project analysis. Learn Organ 3(1):11–15
Lehman MM (1989) Uncertainty in computer applications and its control through the engineering of software. Softw Maint Res Pract 1(1):3–27
Lipshitz R, Klein G, Orasanu J, Salas E (2001) Taking stock of naturalistic decision making. J Behav Decis Mak 14(5):331–352
Lynn GS, Skov RB, Abel KD (1999) Practices that support team learning and their impact on speed to market and new product success. J Prod Innov Manag 16:439–454
Menke MM (1997) Managing R&D for competitive advantage. Res Technol Manage 40(6):40–42
Meso P, Troutt MD, Rudnicka J (2002) A review of naturalistic decision making research with some implications for knowledge management. J Knowl Manage 6(1):63–73
Moe NB, Dingsøyr T, Dybå T (2008) Understanding self-organizing teams in agile software development. In: 19th Australian conference on software engineering, pp 76–85
Moe NB, Dingsøyr T, Dybå T (2009) Overcoming barriers to self-management in software teams. IEEE Softw 26(6):20–26
Moe NB, Dingsøyr T, Dybå T (2010) A teamwork model for understanding an agile team: a case study of a Scrum project. Info Softw Technol 52(5):480–491
Moe NB, Aurum A, Dybå T (2012) Challenges of shared decision-making: a multiple case study of agile software development. Info Softw Technol 54(8):853–865
Molokken-Ostvold K, Haugen NC, Benestad HC (2008) Using planning poker for combining expert estimates in software projects. J Syst Softw 81(12):2106–2117. doi:10.1016/j.jss.2008.03.058
Morgan G (2006) Images of organizations. Sage, Thousand Oaks, CA
Nerur S, Balijepally V (2007) Theoretical reflections on agile development methodologies - the traditional goal of optimization and control is making way for learning and innovation. Commun ACM 50(3):79–83
Nerur S, Mahapatra R, Mangalaraj G (2005) Challenges of migrating to agile methodologies. Commun ACM 48(5):72–78
Outi S (2006) Enabling software process improvement in agile software development teams and organisations. VTT Publications, Espoo
Pearce CL (2004) The future of leadership: combining vertical and shared leadership to transform knowledge work. Acad Manage Exec 18(1):47–57
Pich MT, Loch CH, De Meyer A (2002) On uncertainty, ambiguity, and complexity in project management. Manage Sci 48(8):1008–1023
Raelin JA (2001) Public reflection as the basis of learning. Manage Learn 32(1):11–30
Rittel HWJ, Webber MM (1973) Dilemmas in a general theory of planning. Policy Sci 4:155–169
Schwaber K, Beedle M (2001) Agile software development with Scrum. Prentice Hall, Upper Saddle River

Sharp H, Robinson H (2010) Three 'C's of agile practice: collaboration, co-ordination and communication. In: Dingsøyr T, Dybå T, Moe NB (eds) Agile software development: current research and future directions. Springer, Berlin, p 13

Sharp H, Robinson H, Segal J, Furniss D (2006) The role of story cards and the wall in Xp teams: a distributed cognition perspective. In: Agile. Minneapolis, MN. IEEE Computer Society, pp 65–75

Šmite D, Moe NB, Ågerfalk PJ (2010) Agility across time and space: implementing agile methods in global software projects. Springer, Berlin

Staw B (1976) Knee-deep in the big muddy: a study of escalating commitment to a chosen course of action. Organ Behav Hum Perform 16(1):27–44

Stray VG, Moe NB, Dingsøyr T (2011) Challenges to teamwork: a multiple case study of two agile teams. In: Sillitti A, Hazzan O, Bache E, Albaladejo X (eds) Agile processes in software engineering and extreme programming, vol 77. Lecture Notes in Business Information Processing, pp 146–161

Stray VG, Moe NB, Dybå T (2012) Escalation of commitment: a longitudinal case study of daily meetings. In: Wohlin C (ed) Agile processes in software engineering and extreme programming. Lecture Notes in Business Information Processing. Springer, Berlin, pp 153–167. doi:10.1007/978-3-642-30350-0_11

Takeuchi H, Nonaka I (1986) The new product development game. Harv Bus Rev 64:137–146

Trist E (1981) The evolution of socio-technical systems: a conceptual framework and an action research program. Occasional paper no 2. Ontario quality of working life centre, Toronto, ON

Trist E, Bamforth KW (1951) Some social and psychological consequences of the longwall method of coal—getting. Hum Relat 4(1):3–38. doi:10.1177/001872675100400101

von Krogh G, Ichijo K, Nonaka I (2000) Enabling knowledge creation. Oxford University Press, New York

Whyte G (1993) Escalating commitment in individual and group decision making: a prospect theory approach. Organ Behav Hum Decis Process 54(3):430–455

Woodfield SN (1979) An experiment on unit increase in problem complexity. IEEE Trans Softw Eng 5(2):76–79

Yeh RT (1991) System development as a wicked problem. Int J Softw Eng Knowl Eng 1(2):117–130

Biography Tore Dybå received his MSc in Electrical Engineering and Computer Science from the Norwegian Institute of Technology and the Dr. Ing. in Computer and Information Science from the Norwegian University of Science and Technology. He is a chief scientist at SINTEF ICT and an adjunct professor at the University of Oslo. He has 8 years of industry experience from Norway and Saudi Arabia. His research interests include evidence-based software engineering, software process improvement, and agile software development. He is the author or coauthor of more than 100 refereed publications appearing in international journals, books, and conference proceedings. He is editor of the *Voice of Evidence* column in *IEEE Software and a member of the editorial boards of* Journal of Software Engineering Research and Development and Information and Software Technology.

Torgeir Dingsøyr works with software process improvement and knowledge management projects as a senior scientist at SINTEF Information and Communication Technology. In particular, he has focused on agile software development through a number of case studies, coauthored the systematic review of empirical studies, coedited the book *Agile Software Development: Current Research and Future*

Directions, and coedited the special issue on Agile Methods for the Journal of Systems and Software. He wrote his doctoral thesis on *Knowledge Management in Medium-Sized Software Consulting Companies at the Department of Computer and Information Science, Norwegian University of Science and Technology, where he is now adjunct associate professor.*

Nils Brede Moe works with software process improvement, agile software development and global software development as a senior scientist at SINTEF Information and Communication Technology. His research interests are related to organizational, socio-technical, and global/distributed aspects. His main publications include several longitudinal studies on self-management, decision-making and teamwork. He wrote his thesis for the degree of Doctor Philosophiae on *From Improving Processes to Improving Practice —Software Process Improvement in Transition from Plan-driven to Change-driven Development. Nils Brede Moe is also holding an adjunct position at Blekinge Institute of Technology.*