

# Exploratory knowledge creation in agile software development projects

Tor Erlend Fægri

1) Department of Computer and Information Science, Norwegian University of Science and Technology. 2) SINTEF ICT, Department of Software Engineering, Safety and Security.

*Tor.E.Fegri@sintef.no*

**Abstract.** Agile Software Development (ASD) is a highly popular software engineering methodology that promotes flexibility, close customer collaboration, teamwork and evolutionary development. Promises include improved innovative performance. Strong industrial acceptance motivates a closer investigation of the merits of ASD in praxis. In this theoretical paper we investigate popular ASD methods from an organization theory perspective such as organizational learning and discourses of power to evaluate impacts of ASD on one aspect of innovative performance; learning and knowledge creation among the project team members with a particular attention to the two knowledge search strategies exploitation and exploration – both essential for organizational survival. We conclude that ASD in praxis may benefit exploitation but that its benefits for exploration are unclear.

## Introduction

Software organizations participate in the actual construction of advanced and sophisticated technological systems by supplying parts of, or complete software systems. Software systems are programs, similar to text, that precisely instruct computers what to do. Such software systems are in high demand because they provide essential support for increasing operational performance in many

different contexts (Cusumano 2004; Drucker 2007). Software is already an indispensable part of modern society – software is used for a wide range of everyday tasks: From mission-critical control systems, Internet infrastructure and Internet-based services, tools and games on personal computers and even the operation of cars and mobile phones. And the amount of software in our technological environment seems to increase still. It is difficult to imagine what society would be without software.

Software organizations participate in interorganizational networks consisting of governments, suppliers, customers, competing software organizations and numerous others. We can call such networks the environment of the organization (Hatch 2006). The environment is competitive. Multiple competing software organizations attempt to make a profit in the same market. Furthermore, because many different groups of people with different roles and needs may be using the software actively and directly, this contributes to a stream of requests for updates and changes. And more importantly, these groups of people participate in the social construction of the technological system through which the technology may attain different meanings and values to its users (Pinch and Bijker 1987). Via these relationships, the software organization is influenced and guided by a process that shifts between variation and selection. The software organization is forced to be sensitive to feedback from the users in order to produce attractive and useful products. Software organizations supply different variants of a product. Subsequently, the social groups using them select, or approve, the variants that best suit their technical requirements. Eventually, the social construction process stabilizes and the technical solution is “taken for granted” and the problem disappears (p.44). The key point here is that technological development deals with social processes involving many different actors trying to identify the best solution through conversation (Levin 1997). The software organization has intricate dependencies to its environment.

During periods of improvements technological systems are subjected to minor refinements and extensions that tend to make the system more complex and larger. As software systems are technological systems, their complexity and size tend to increase too. It can be very difficult to control such a system. Gradually, as it grows larger, the technological system gains a momentum, with its own goals, rate of growth and direction (Hughes 1987). In practice, in an attempt to manage the complexity and size it is common to subject software systems to decomposition whereby it is divided into multiple, more easily maintainable programs. Because of their non-physical nature, software systems are easily altered to obtain different results in use. Flexibility is in many ways the core value of software. It enables the tailoring of a software system to the particular requirements of a user. The tailoring might in fact be done by users themselves, before or during use. As such, software systems have a significant potential to be tailored to reflect feedback from the user groups. In the competitive software

industry it is widely accepted that the ability to quickly produce new versions and new functionality is the principal competitive advantage for any software organization (Baskerville, Ramesh et al. 2003; Cusumano 2004). This contributes to maintain a high frequency of new offerings from competing software organizations. Also, even minor changes in one software program can create demands for fundamental changes in a software program that has dependencies to it. For software organization delivering parts of software systems this creates high levels of turbulence in their environment. Even software organizations delivering whole systems face the challenge of the continuously rising complexity. Tomorrow, their system may be the subsystem of a larger system. In turbulent environments organizations should prepare to evolve in order to survive (Sawhney and Prandelli 2004). Therefore, software organizations must continuously try to adapt. Skills and knowledge are fundamental to all human activity within and outside the organization (Argyris and Schön 1996; Polanyi 2000). Adapting to change requires the acquisition of new skills and knowledge so that the organization can solve new tasks and problems. March (1991) distinguish between two types of knowledge search strategies: exploration and exploitation. The former deals with the search for radically new knowledge while the latter deals with refinement of existing knowledge. For organizations operating in turbulent environments it becomes crucial to attentively balance exploring emerging possibilities and exploiting past investments. A too high attention to exploring emerging possibilities reduce the ability to capitalize on previous investments. A too high attention to exploiting past investments will contribute to the continued obsolescence. Argyris and Schön (1996) discuss the great importance of organizational learning. In brief, they argue that fundamentally new knowledge can be obtained efficiently by double-loop learning. Double-loop learning is supported by openly questioning existing assumptions and values within the organization. They further suggest that this type of learning stimulates technical innovation because it can eliminate competence traps (p.245). The acquisition of skills and knowledge in the software organization therefore becomes the principal line of discussion in this paper.

Agile Software Development (ASD) is a development approach for software projects that draws on organizational learning theory and collaborative decision making based on mutual adjustment. ASD has gained significant interest within the software industry during the last 5-7 years. The supporters of ASD claim significant benefits from adoption, for example in terms of innovative performance and exploratory knowledge creation (Highsmith and Cockburn 2001; Nerur and Balijepally 2007). Rhetorically, ASD appears to support the challenges of software organizations in their efforts to acquire new skills and knowledge at increasing speeds. The primary focus of this paper is to critically discuss the promises of ASD in practice in the context of organization theory as it has been outlined in the preceding paragraphs. The paper takes a critical position to the

claim that ASD supports exploratory knowledge creation. The main argument of this paper is that ASD in praxis inhibits, or makes exploratory knowledge creation more difficult.

We have now introduced the problem area and the focus of the discussion. The remaining part of the paper is structured as follows: Next, we will explain some very important aspects of learning and change within the software organization. The following section gives an overview of Agile Software Development; its origin, theoretical foundation, key objectives, and industrial influence. Subsequently, we attend to how ASD is set into practice. In this practical context we critically evaluate how ASD practices affect the work of professionals with a particular attention to exploratory knowledge creation. The last section discusses implications of the evaluation.

## Skills, knowledge and learning in the software organization

Organizations are systems of coordinated action among individuals and groups whose preferences, information, interests or knowledge differ (March and Simon 1993 p.2). Coordination among members of the organizations is supported by organizational routines. The term routine is comparable to what we denote skill at the individual level (Nelson and Winter 1982 p.97). Similar to the way that our actions represents personal tacit knowledge (Polanyi 2000), organizational routines are part of the organization's knowledge – organizations remember by doing (Nelson and Winter 1982 p.99). However, there is much more to it than simply “knowing the routine”. It is also the dealing with what to do at what time. Different signals and different messages give input to the task and the organizational context influence how messages should be interpreted (p.102). Nevertheless, and even despite best intentions, decisions are made by people with less than perfect cognitive abilities, less than complete knowledge and a continuous lack of fully updated information. March and Simon discuss this non-optimal element of decision-making as a kind of bounded rationality (March and Simon 1993). An important consequence of this is that search for information, individual and group-oriented problem-solving based both systematic and compositional strategies, and a significant element of randomness will colour the organization's learning activities (p.197-203). The learning is in the action; as people interact and make new experiences they extend their skills and knowledge. Polanyi (2000) describes this by noting that knowledge has a “to-from structure”, it contains the things we already know and is extended through new experiences.

Learning is a fundamental aspect of work. Through learning we can acquire new knowledge and skills, refine existing knowledge or improve existing skills. Learning is the vehicle that enables new members of the organization to

understand its particular circumstances and activities, technologies, culture and structure. And without sufficient knowledge and skills among its employees, the organization cannot accomplish its activities in an effective manner. Organizational learning seeks to understand learning that occurs beyond the individual level. One can say that organizational learning is the way in which the organization continuously learns from the individuals and “teaches” new individuals this knowledge (Berger and Luckmann 1966).

Nelson and Winter notes that the ability of the organization to accomplish useful things are primarily rooted in the repertoire of skills among the organization’s members. They claim that skills, organization and technology is intimately intertwined in a “functioning routine” (Nelson and Winter 1982 p.104). Further, an important key is their ability to interpret messages to perform the tasks to the best for the organization. Part of the organizational memory is in the equipment, structures and work environment. Most importantly, the context of the information possessed by an individual is established by the information possessed by all the other individuals (p.105). The great importance of memory for problem-solving is also noted by March and Simon (1993 p.198). They suggest that most problem-solving depends on a restructuring of program details already in memory. Argyris and Schön (1996) make a similar claim by stating that “organizations directly represents knowledge in the sense that they embody strategies for performing complex tasks that might have been performed in other ways.” (p.13)

Software organizations depend on routines too. Software organizations frequently use software processes to prescribe a normative scheme for the coordination of the activities required to construct a software system (Sommerville 1996). Activities may include the gathering of user requirements, creating an overall system design, building a prototype, building the production software and organizing the testing and verification of the final system against requirements. The software process is, arguably, the most central among a set of organizational routines in the software organization.

Nelson and Winter notes that organizational routines can be compared to individuals’ skills in that little attention from higher level management is needed to complete them (Nelson and Winter 1982 p.125). Further, they suggest that organizations are poor at improvising in novel situations. An explanation for this is that bounded rationality have bigger impact on organizations than individuals in the sense that the tradeoff between capability and deliberate choice works less favourably towards the latter in organizations – one cannot assume that an organization is an “intelligent organism” that responds well to environmental changes. Nelson and Winter presents an evolutionary model that is substantially different from the orthodox models. They include aspects of flexibility, latent conflicts and other firms in the environment (p.127). The core of their argument is that organizational behaviour can be explained by the organization’s routines and

how they change over time (p.128). Supplementing this line of argument is Hughes' assertion that it is only with the occurrence of "reverse salients", or major problems, that opportunities for radical changes to technological systems are created (Hughes 1987). Argyris and Schön (1996) makes a similar point, suggesting that prior success may easily create competence traps by which organizations fail to recognize that skills and knowledge giving success in the past will eventually lead to obsolesce (p.19). The topic of competence traps has been discussed more fully by March and Simon (1993) who note that that innovation depends on a significant amount of searching for "raw" knowledge – knowledge that is not already in memory (p.198). This searching for raw knowledge corresponds to exploration as mentioned earlier. Considering the tendency of large technological systems to attain their own momentum, we can start to appreciate the difficulty of planning for skills and knowledge development.

Change require learning. Our knowing is in the action claims Schön (1983 p.49). Here, Schön argues that knowledge and action are inseparable. All work has both elements. He does, however, discuss knowledge at a personal level. Organizational change can also be viewed as a learning process (March and Simon 1993). Organizational learning discuss learning as a way by which organizations improve existing skills or acquire new skills. Organizational skills are reflected in the routines used by the members to achieve their tasks (Nelson and Winter 1982). Argyris and Schön (1996) discuss two different models of organizational learning – single-loop learning and double-loop learning. Single-loop learning is "instrumental learning that changes strategies for action or assumptions underlying strategies in ways that leave the values of a theory of action unchanged" (p.20) while double-loop learning is "learning that results in a change in the values of theory-in-use, as well as in its strategies and assumptions." (p.21) In order to transform espoused-theory into theory-in-use double loop learning is necessary. Greenwood and Levin (2007) put a similarly strong emphasis on focusing on the organization's practices through their action research project. Both projects are intimately based on an ambition to change through action. However, Greenwood and Levin go further in accepting a balanced relationship between the change actor and the organization than Argyris and Schön. The notion of organizational change through a evolution in the composition of routines, set forth by Nelson and Winter (1982), resembles the view of Argyris and Schön who suggest that the most concrete phenomena in an organization are its theories-in-use and that learning occurs when the organization's members try to change these theories-in-use. In this context, exploration can be compared to double-loop learning in which assumptions, norms and values are constantly questioned (Argyris and Schön 1996). But organizational learning has paid only modest attention to the perspective of power.

Indeed, power is a fuzzy and elusive concept being discussed and analyzed by a wide range of scholars over a long period of time. Four dimensions of power are nevertheless apparent within the literature. The first dimension is the pluralistic, liberal conceptualization of power as upheld by Dahl (1963). Essentially, this dimension states that A has power over B if A can make B do something that B would not otherwise do. In the two-dimensional view of power, proposed by Bachrach and Baratz, power also includes “non-decision making” whereby A’s influence over B takes the form of avoiding or restricting B in discussing topics that would affect A (Clegg 1989). In this view, power is discussed within an environment consisting of an elite that has influence over the common man. The three-dimensional view, even more sophisticated and sly than the preceding two, was proposed by Lukes (1974). This view of power maintains that the most effective control of people can be obtained if the use of power is invisible or is considered harmless by the people over which the power is exercised. Thus, A has power over B if A can control or influence the shaping of B’s perceived needs and demands. A simple example of this dimension of power is influencing the agenda for discussions. Last, in the fourth and most radical dimension of power, power is ubiquitous in all human collaborations and people are largely self-disciplined. The process of establishing respect for authorities and obey their will starts at the time of birth and continues throughout our lives, in schools, in the army and in the factories. Foucault refers to this dimension as disciplinary power (Clegg 1989 p.153). In Foucault’s view, people act according to values and norms that are developed by the ruling authorities. They are imprisoned.

Groups of people working together are prone to creating their own rules and power relationships. Studies of the working collective at an Norwegian paper producer in the 1960-ies showed that strong group norms, group censorship and behavior code developed in response to management objectives of efficiency and rationalization (Lysgaard 2001). These norms and codes helped to establish a strong group that could create a “barrier” towards management’s pressure. However, while these norms helped protect the group from outside pressures, the norms also created an inexorable and strongly conforming working environment for the group members. Barker (1993) discuss the power relationships of this social phenomenon in a small group setting. He concludes that the group pressures can create power relationships even stronger than the “iron cages” blamed on traditional hierarchical organizations. Taylor (1911) provides a different, yet simpler perspective on the similar phenomena that he called “systematic soldiering”. He viewed workmen as notoriously prone to resisting work at a pace he would denote a “fair day’s work”. Taylor’s view is solely that of soldiering as a problem for efficient management – a problem that should be eliminated. His grand project, scientific management, sought to demonstrate that performance of production could be dramatically increased if division of labor was exercised to the extreme. In his scheme, workmen did the work under close

monitoring from supervisors that followed scientifically and precisely developed plans. Rhetorically, Taylor justified his ambition by stating that he “wanted to develop each man to his highest state of efficiency and prosperity.” (p.43) But essentially, Taylor wanted to abandon the collegial team that Lysgaard promoted.

Social structures, such as those being interrogated and changed in organizational learning or action research programs, lend themselves directly to consideration from a power perspective. When Argyris and Schön attempts to identify and eliminate the defensive routines inhibiting double-loop learning in an organization (Argyris and Schön 1996 ch.7) and when the group studied by Brown and Duguid (1991) narrate, collaborate and create their common understandings for the problems they face in their work, there are naturally also relevant relations of power between these individuals. For example, it would be naïve to think that all members had the same norms and goals initially. The adjustment of individual’s norms and goals will be subject to a power struggle. Therefore, there are good reasons to assume that the power relationships described by Lysgaard (2001) and Barker (1993) will also exist in software engineering teams.

The socio-technical discourse, heavily influenced by the work of Trist and others at the Tavistock Institute in the postwar era, made a significant contribution to our knowledge regarding the interplay between work practices and technology. Trist and Bamforth (1951) made a compelling case by presenting the dangers of failing to consider the psycho-social aspects of work when introducing new production technology. What is more, the socio-technical discourse has had a direct impact on democratization of work and setting focus to the dangers of alienation in the work setting. Blauner (1964) found that alienation may easily result if the work is not associated with learning. Braverman (1974) was highly concerned that specialization of work and a capitalization of production means would bring management to a level of control that would ultimately hurt the workers. Trist (1981) set new attention to the benefits of empowering the worker, for example through autonomous teams. He found multiple benefits of autonomous teams; increased depth of commitment (p.18), improved work coordination (p.19), reduction in management overhead and cost (p.19), less boredom and alienation (p.27), improved group learning (p.34), increased problem-solving capability (p.34), better settlement of special needs (p.34) and flexibility (p.34). However, the fundamental empirical basis of the socio-technical discourse is in manufacturing organizations where workers have relatively well-defined tasks.

Naturally, a very important aspect of work is our understanding of the organization. How do people create meaning of the work organization and the tasks they do? Berger and Luckmann (1966) provide an important perspective for the discussion of social construction in their treatment of the sociology of knowledge. They accept Weber’s (2000) notion of ideal types but nevertheless

oppose the view of modernism and positivists that favor models and rigorous abstractions. Berger and Luckmann suggests that our reality is constructed in a continuous dialectic between individuals and society. Reality is socially constructed in an ongoing conversation between people, shaped by their history and common experiences. Individuals' subjective realities are externalized as objective reality through action while this objective reality is also being internalized by individuals into subjective reality. For example, when a new member enters the organization, we can easily appreciate the internalization process as the newcomer attempts to make sense of the roles, routines and other social practices. It is by the internalization process that we realize the social dimension of reality (Berger and Luckmann 1966 p.76). Weick, claims Hatch (2006), builds upon this view that reality is objectively constructed in his enactment theory. Weick suggests here that the organizational objective reality is greatly influenced by management in their gathering of information and by the decisions made from this information. In Weick's perspective, this creates rich opportunity for change (Hatch 2006 p.58). Berger and Luckmann sees this differently and asserts that organizations in society outlive individuals but individuals have a small influence in creating them. Organizations, claim Berger and Luckmann, represents hardened behavior that is externalized as routines. Changing them creates friction.

Working with software development involves problem-solving (March and Simon 1993), i.e. searching for solutions to problems. Brown and Duguid (1991) explain problem solving in terms of three activities; narration, collaboration and social construction within loosely connected, naturally occurring working groups. Through *narration* the workers create stories that helps to find solutions to the problems they are facing. The stories allow the workers to build common understanding and thereby assisting in the construction of a common story that ultimately lead to the solving of the problem. The workers *collaborate* and thereby integrates working and learning, as well as making individual learning inseparable from collective (group) learning. And lastly, through *social construction* the context surrounding this particular context is created. It is not a generic model of problem solving, but a construction regarding this particular problem. It is also through social construction that the workers build their identity. To foster working, learning and innovation an organization must close the gap between espoused and actual practice by conceiving itself as a community-of-communities, claim Brown and Duguid (p.53). To a degree, the project oriented way of working, that is very common in software organizations, reflects the notion of community-of-communities. Multiple, and sometimes parallel, projects are defined, maintained and completed throughout the life of the software organization. People participate in these projects without fully knowing what projects they will be part of next.

For many software organizations, work is primarily based in projects, with a matrix-like organizational structure. In the matrix organizational structure projects are dynamically manned from pools of people with functional roles (Hatch 2006). Project teams may be of almost any size but will normally have a limited lifespan. People may be allocated to multiple projects, in which case they are engaged in solving different tasks. This flexibility in work organization can contribute to effective and timely adaptation to changes in the organization's demands but shortage of people within certain functional specialization groups and conflicts of authority may arise (Hatch 2006). Within the project the team members collaborate and coordinate their work to accomplish the tasks. The project team thus becomes a sub organization within the organization, albeit with a limited lifespan. Large and complex software systems represents a breadth and depth of knowledge not embodied in a single developer. Therefore, developers must collaborate in teams to build such systems. For a software organization with a project-based structure, people will spend most of their time working on project-related activities. Thus, the primary context within which meaning is constructed and new knowledge is created is the shared practice within the software project team (Dybå 2003), Most decisions affecting the work will be taken within the team and therefore the software project team can be seen as a learning organization in itself.

## ASD Projects

The primary objective of this paper is to discuss critically the use of ASD methods in software organizations. As ASD is a project management approach it is natural to focus our attention first to the people being most affected by the ASD methods and those are the people working within the ASD governed projects. We now continue with a description of some fundamental aspects of software development projects and the principal activities that they must coordinate.

Software engineering is a discipline within engineering that deals with professional construction of software systems. The engineering of software is intellectually intensive (Humphrey 1997; Dybå 2000; Glass 2006; Slaughter, Levine et al. 2006) from which the produced software, the product, can be viewed as an explicit representation of knowledge (Leiponen 2006). We base our understanding of knowledge as having two dimensions, knowing-how and knowing-that (Gourlay 2006). The former is conformant to Polanyi's conceptualization of tacit knowing (Polanyi 2000) and Schön's knowing-in-action (Schön 1983). The latter dimension is knowledge represented in an explicit, symbolic, context-free form, for example in written documents (Gourlay 2006), or here, software programs. Both dimensions of knowledge are central to the work of software engineers. Explicit knowledge, as mentioned, present itself in the software and supplementary models, documents and artefacts that are used to

guide the development. The tacit dimension of knowledge, the knowing-how, is in the skills stemming from experience, turned into action in the personal situation of the developer when the software is created.

Most software engineering efforts are too complex to be undertaken by a single person and are therefore allocated to project teams. Teams of people with different skills and roles must collaborate in these projects. The project associates a goal vision with a set of resource constraints to accomplish the goal vision. Resource constraints may include people, money and time. The software process, one, for example based on ASD principles, deals with the organization of the project. Software processes can be viewed as an instrument of bureaucracy by a legitimization of authority through roles and rules (Weber 2000). Bureaucracies are efficient because they formalize, optimize and standardize repeating tasks through particular task knowledge (p.126). Another highly important aspect of work is the role we assume within the organization. Not only is the role important for the understanding of communication (March and Simon 1993 p.22), but the role will also influence the type of tasks performed and the knowledge and skills developed through carrying out the associated actions. While the process represents the institutionalization of organizational routines it also supports the development of specialist knowledge that gradually becomes embedded in the working roles and the types associated with those roles (Berger and Luckmann 1966 p.70). In a sense, the processes represents repeatable patterns of action, that may also relieve people from their attention to minute details and help them in exploration and reflection. Although software processes, due to their project organization focus, represent bureaucracy on a shorter term and of a smaller scale than organizational bureaucracy they will still have profound effects on the people working in the project. Because software project participants spend most of their time working in projects and subsequently are highly concerned with the activities governed by these projects. Similarly, the team will be the primary organizational structure for learning (Edmondson 2002). The project exists within the environment of the software organization and the project will normally exist in a relationship with other organizations or user groups outside the organization. Therefore, software engineering is a social construction process with actors both within the software organization and outside the organization, in the form of social user groups and other stakeholders.

As noted, knowledge creation and learning is a key activity in software engineering. While it might be argued that this constitute a liberation of work and the developer, Sørhaug (2004) suggests that it in practice simply is a more sophisticated form of alienation in which the “knowledge worker sells himself” (p.136). Slaughter, Levine et al. (2006), taking a rather less depressing view, suggest that software engineering is more similar to research and development than manufacturing. This view is supported also by Cusumano (2004) and Glass (2006). What is clear, however, is that any advantages in manufacturing

performance will be of less significance within software engineering than in other manufacturing activities. From this viewpoint knowledge creation is of higher importance for the software organization than for the manufacturing organization. This is not to say that software engineering is knowledge-intensive and other kinds of work are not. Skills and knowledge are fundamental to any type of work (Schön 1983). The point is solely that manufacturing plays an insignificant role in software engineering. The work is within the knowledge created among the team members which is translated into a technical system, the software. Making a copy of an existing piece of software is almost infinitely cheap. Therefore, the main focus of software engineering is therefore the creation of unique products. In order to accomplish this, the project team must acquire new knowledge, the team members must acquire new skills. Learning should be nurtured in the team because this is the primary arena for work.

March (1991) contributes to the discussion of knowledge creation with the notion of a spectrum of knowledge search strategies, with exploration and exploitation at the two extremes. Exploration represents the search for new and novel knowledge and is what we normally associate with innovation (see for example Hughes' (1987) discussion of innovation). Exploration, in this sense, will be a means to overcome the competence traps discussed earlier. In contrast, exploitation represents the refinement of existing knowledge and is associated with incremental improvement and adjustment. Both exploration and exploitation are necessary for long-term organizational survival (March 1991). Intuitively, organizations must engage in exploitation to reap the benefits of current investments. In parallel, exploration is necessary to create new opportunities as the current knowledge becomes increasingly obsolete and to avoid competence traps (March and Simon 1993). The key is thus a balance between the two strategies (March 1991). In order to approach this balance it is helpful for a software organization to know more about that consequences of the software processes that the organization's projects are using.

It is at this point useful to briefly turn our attention to the organizational context, surrounding the people working within ASD projects. This is important in order to understand how ASD can affect the organization's learning abilities. As most other organizations, the software organization has elements from the two ideal types action organization and political organization discussed by Brunsson (2002). While the action oriented organization creates a common ideology towards enthusiasm, conformity, solutions and specialization, the political organization fosters criticism, variety in ideology, talk and action resistance. In order to satisfy numerous, and often contradictory demands from its environment, the organization engage actively in hypocrisy. The hypocrisy enable the organization to satisfy its technical environment with results and to satisfy its institutional environment with talk and decisions. Thus, it is important to keep in mind that it is not only the production of action and tangible artefacts that matter.

Talk and decisions are also products of the organization. While one might think that the role of software developer, or “coder”, is the main interesting role with respect to the production of products in the software organization, this is too simplistic. The “product” of the software organization is not simply the software system, it is a combination of talk, decisions and actions. Actions, that ultimately leads to the construction of the software system, are undoubtedly very important for the organization as this generates the income. But talk, such as marketing activities that support the creation of demands for the organization’s products within different social groups, and decisions, for example creating illusions of organizational commitment to certain strategies and improvement programs, are also critical.

Software Process Improvement denote activities used to advance the routines used to create software systems. The history of SPI short. The first significant effort to formalize and standardize the software engineering process occurred in 1970, with Royce’s waterfall model (Royce 1970). The quest to provide increasingly more efficient process models continued during the 1980-ies and the 1990-ies (Sommerville 1996; Highsmith and Cockburn 2001; Boehm 2006). During the 1980-ies and the 1990-ies, the predominant view of good software project management was based on values in predictability, quality management, control, up-front planning and rigorous standardization of routines (Boehm 2006; Nerur and Balijepally 2007). The inspiration was to a large extent from the Japanese success in the 70-ies and early 80-ies within the manufacturing industry. Lead by these values, rigorous and heavy-weight process models such as RUP (Rational Unified Process) became popular. Furthermore, a lot of attention was given to more bureaucratic factory-based view of development with the hope that strictly formalized processes would enable the rationalization of large-scale development efforts through reuse of software components and standardization. Results from these endeavors were mixed, and showed that large up-front investments rarely paid off and that “over-the-wall” management created inefficiency and sustained inflexibility. Before the software was finished, the requirements had changed. There was a profound acknowledgement of the ever changing nature of the project’s goal vision that fuelled a renewed interest in more lightweight and less rigid alternatives (Dybå 2000). Interestingly, RUP was also claimed to be lightweight due to its configurable nature (Boehm 2006), but in practice, the enormous repertoire of practices and choices for configurations, caused that RUP was rarely configured to suit the organization, nor the projects. ASD, on the other hand, claims a anti-bureaucratic stance (Nerur and Balijepally 2007). For a long time, researchers and professionals had seen the problems of traditional development methods and proposed to adopt evolutionary development methods based in incremental and iterative development (IID) practices (Adams, Day et al. 1998; Highsmith and Cockburn 2001; Larman and Basili 2003; Trott 2004). These ideas resonated with many software practitioners

but the industrial adoption was moderate. This changed quite dramatically when a group of them gathered in 2001 to write the Agile Manifesto<sup>1</sup>. The Agile Manifesto can be seen as the ideological commonality and the origin of ASD. It lists four key guiding values: 1) flexibility before planning, 2) close customer collaboration before contract negotiation, 3) individuals and interactions before processes and tools, and 4) working software before comprehensive documentation. From these four values, the Agile Manifesto derives 12 shaping principles. These values and principles represent a substantial reordering of traditional development processes. Nerur, Mahapatra et al. (2005) has made a review of the agile ASD literature and produced the following brief summary of the main differences between traditional development and agile development, grouped into seven topics:

Topic	Traditional development	Agile development
1. Fundamental Assumption	Systems are fully specifiable, predictable, and are built through meticulous and extensive planning	High-quality adaptive software is developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change
2. Management style	Command and control	Leadership and collaboration
3. Knowledge management	Explicit	Tacit
4. Communication	Formal	Informal
5. Development model	Life-cycle model (waterfall, spiral or some variation)	The evolutionary-delivery model
6. Desired organizational form/structure	Mechanistic (bureaucratic with high formalization), aimed at large organizations	Organic (flexible and participative encouraging cooperative social action), aimed at small and medium sized organizations
7. Quality control	Heavy planning and strict control. Late, heavy testing	Continuous control of requirements, design and solutions. Continuous testing

**Table 1: Comparison of traditional versus agile development**

---

<sup>1</sup> <http://agilemanifesto.org/>

The Agile Manifesto and the characteristics identified by Nerur, Mahapatra et al. represents the cornerstone of most ASD methods. Worth noting is that neither the four guiding values or the twelve principles would constitute a software process model by themselves. Multiple software processes have been proposed, however, that claim adherence to the Agile Manifesto (Abrahamsson, Salo et al. 2002). Widely adopted ASD process models include XP and Scrum. Scrum, in particular, was inspired by a Japanese incremental and iterative production process used for non-software products (Takeuchi and Nonaka 1986). XP is also fundamentally linked to this Japanese process. During the last 5-7 years or so ASD has seen a lot of interest and strong rates of adoption (Dybå and Dingsøy 2008).

Following the Agile Manifesto, the product is developed in steps, and each step is “verified” by the customer. Nerur and Balijepally (2007) compares the element of repeated problem reframing in ASD, in part stimulated by planning reluctance and customer collaboration, with Schön’s model of reflection-in-action (Schön 1983) and the generative learning model of Argyris and Schön (1996). In the former perspective, the problem-solving is a cyclic conversation with the problem in which the problem is repeatedly reframed, a solution is proposed and the solution is evaluated according to its ability to solve the problem. Through such conversations between the problem-solver and the problem, learning occurs during reflection and action, in a dialectic between tacit knowing and explicit knowledge. In the latter perspective, ASD are effectively adherent to the generative learning model of double-loop learning by promoting learning through experimentation, action and problem reframing claims Nerur and Balijepally.

Judging from Table 1, one can say that ASD represents a shift in the governing values of organization of software projects from bureaucratic, predictability-seeking and plan-driven towards flexible, change-driven and humanistic problem solving. Autonomous teams, as they are promoted by the collaborative, participatory and cooperative aspects of ASD work (see Table 1) promise to liberate the team members from direct management control by authorizing the group to make their own decisions. Following Brown and Duguid’s (1991) arguments, these autonomous teams would appear like the loosely coupled communities that facilitated learning and innovation in their investigation.

Based on these arguments we are led to assume that exploratory knowledge creation may find a supportive environment in ASD projects. Indeed, at this point it would seem only fair to conjure with Highsmith and Cockburn (2001) as well as Nerur and Balijepally (2007), in their claims that ASD is beneficial for exploration. But there is no clear scientific knowledge based in empirical investigations available to support the claim that ASD is beneficial for exploratory knowledge creation (Dybå and Dingsøy 2008). Therefore, there are good reasons to question ASD’s support for exploration. The theoretical arguments given by Nerur and Balijepally (2007) appear solid however. In order

to understand the realities of ASD we need to investigate how ASD is set into practice. We must turn our attention to how ASD values and principles are implemented in praxis to uncover the realities of ASD.

## ASD in praxis

Firstly, we must ask, what is the praxis of ASD? The praxis of ASD is partly explained by the software methods claiming adherence to the Agile Manifesto. Examples include Scrum (Schwaber and Beedle 2002), XP (Beck 1999) and a number of others. The two mentioned are by far the most popular, however. Scrum and XP are easily accessible for industry, both methods are backed by commercially available training programs, books and active Internet forums. These method solutions go far in standardizing project organization, although it should be noted that Scrum is primarily a project management practice, and is not concerned with the detailed practices that the developers use to create the software. At least not at the detailed level of XP. XP on the other hand, deals not so much with project management as the techniques used by developers in their daily work. In fact, XP should be considered more as a collection of techniques than a process model. It is very common that projects following the Scrum approach adopt techniques from XP.

Empirical investigations on knowledge creation in ASD projects are limited. Bahli and Zeid (2005) found that adoption of XP was facilitated by “a high degree of knowledge creation enabled by mutual development of information systems developers and users.” The authors attribute the improvement in knowledge creation to the more frequent conversations with the customer and the problem. Their investigation did not address the exploratory or exploitative characteristics, however. In fact, from their discussion it appears that XP in this case primarily improved the team’s ability to understand and implement the customer’s need, leaving the climate for experimentation and exploration of new knowledge untouched. Gassman, Sandmeier et al. (2006) investigated the use of XP for innovative NPD projects and claimed that XP supported a good balance between creativity and resource control. However, two aspects reduce the relevance of this investigation for the present discussion. First, their selection of case projects is problematic. The case projects were selected because they were promoted as highly innovative by the organization and the customers before the projects started. Thus, it is difficult to say if it was the XP process, or the project’s funding policy or stated objectives, that stimulated the exploration of new knowledge. Secondly, XP practices were only used for the initial phases of the projects. Of main interest in this paper is an discussion of how the ASD process, when used in normal praxis, influence the exploration of new knowledge among the team members.

In short, the most important reason for a lack of support for exploration in ASD projects is its main intention. The main intention of ASD is to improve project performance within the organization by reducing uncertainty by empirically controlling the evolving system with the customer's requirements, reducing overhead, and pushing people harder by enforcing group commitment and elaborate power relationships. Each of these points will now be discussed with respect to organization theory and learning.

Schwaber, who together with Jeff Sutherland can be called the creators of Scrum, puts focus on performance and efficiency in his latest book on using Scrum in the large organization: "Scrum makes the whole enterprise more effective." (Schwaber 2007 p.xi) He claims that "Scrum takes people out of the comfort zone, trading commitment to customers for creative and enjoyable work." (p.69) Schwaber deserves credit for the good intentions regarding creative and enjoyable work. But how Scrum is supposed to sustain creativity is more unclear: "The ScrumMaster (the team leader in Scrum) is responsible for improving the lives of the development team by facilitating creativity and empowerment." (p.78) In XP, a customer representative is expected to be located together with the team at all times, giving continuous directions for the team. But how can the prevailing commitment to customers' uncertain requirements be transformed into an environment for creative work? This would fundamentally contradict the view of Hughes (1987) that noted that most radical innovations, for which exploratory knowledge creation is fundamental, originates from individuals with an "outsider's personality" that is not tightly bound to the mission-orientation of the organization. Furthermore, Amabile, Hadley et al. (2002) found that stress is inhibiting creativity. "Taking people out of the comfort zone" has a strong resemblance to increased stress levels when viewed in correspondence with the added discipline and control enforced. Therefore, it appears unclear how the efficiency focus of ASD may support exploration of new knowledge.

ASD intends to reduce uncertainty in software development. Scrum has its basis in empirical process control (Schwaber and Beedle 2002 p.24-25) in which the Product Owner decides what tasks are part of the iteration. In practice, these tasks tends to be rather small, because the group estimation process effectively shows the limitation of human cognition when it comes to predicting effort used to solve problems. XP seeks to reduce uncertainty by prescribing an on-site customer that is continuously available and can explain the exact requirements to the software. Therefore, ASD represents a market-oriented view to software development. If we contrast ASD with the open source paradigm (OSS), as explained by e.g. von Hippel (2005), this becomes clearer. OSS promotes open collaboration between any organization or individual to create new knowledge and technologies without the restrictions of producing continuous and immediately verifiable customer value. ASD, on the other hand, is, in fact, mostly about evolutionary delivery of immediately useful customer value. This is an

inhibitor to exploration. One key to this is that customers know roughly what they need, but they are rarely interested in, nor particularly knowledgeable about technological innovations that in fact may be irrelevant or disruptive to existing routines (Christensen 2003). The strong customer collaboration emphasis in ASD is thus a threat to exploratory knowledge creation. Exploratory knowledge creation is associated with radical innovations, for which markets are hidden or do not yet exist (Brown and Duguid 1991; Christensen 2003). Through its emphasis on close customer collaboration and frequent product handoffs, ASD fosters an environment for exploitative knowledge creation and refinement of existing products or technologies. OSS, on the other hand, does not restrict knowledge production to the producer/consumer constellation, but attracts users, academics and businesses alike, in a free, democratic collaboration. Such open collaboration models appear likely to support exploration better, also in software engineering (Tanayama 2002). Uncertainty reduction is a natural tendency in organizations. Taylor (1911), well-known for his work within scientific management, developed his practices from the assumption that the most efficient production comes from lack of uncertainty and autonomy. However, uncertainty is inherently linked with exploration. Accepting uncertainty is, a premise for exploration of new knowledge. Therefore, it appears difficult to stimulate exploration within ASD.

ASD aims to reduce overhead. The reason is found in ASD's fundamental opposition to the traditional, plan-based methods in which significant investments were made in upfront planning – such as specifications and designs (see Table 1). We see also the motive to reduce overhead in the evolutionary delivery model discussed in the previous paragraph. ASD methods do not want to accept uncertainty. If the development process is continuously monitored empirically the chance is higher for detecting mistakes. But mistakes are inherently part of exploration. If we decide to search for “raw knowledge” as March and Simon (1993 p.198) describes it, we must also accept the risk that the knowledge is not suitable for this particular task. Thus, from ASD's strong motive to reduce waste we can also assert that ASD in praxis will also reduce the opportunities for exploration of new knowledge. Berger and Luckmann (1966 p.70) noted that routines and habits could help people free time for exploration and reflection. Exploration is, in its essence, an activity associated with luxury, free time and reflection.

Our last argument regards power and individual performance. ASD promotes empowered autonomous teams that make decisions on behalf of the team members. Autonomous teams were strongly supported by Trist (1981) within the manufacturing organization due to the increased control that the team member achieved on the work. But the team members' power is mainly focused inward; they are given the power to sort out their own priorities and problems. This is the origin of concertive control and internal group norms and group pressures (Barker

1993). These power relationships can easily force the group to accept greater responsibility for the results. Additionally, there is a danger in knowledge-intensive work that people feel burnt-out due to the tendency of accepting a lot of responsibility (Sørhaug 2004 p.136). Which again is a potential danger for supporting exploratory knowledge creation in ASD teams. The emphasis of ASD on collaboration and interaction between team members, appears to resonate with the observations of Brown and Duguid (1991) that learning has a fertile environment in dynamic, loosely coupled communities of practice through narratives, collaboration and social construction. But the autonomous teams of ASD are not loosely coupled. In Scrum, for example, the team is typically rather static and do not differ substantially from ordinary teams in that respect. This is in line with Sørhaug's argument: "Organizations has a certain rigidity in the form of work diversification, standardization, routinization and so on. Organizations don't learn. They have already learnt." (Sørhaug 2004 p.254) Although the findings of Brown and Duguid (1991) are not made in an ASD context, they give some indications of ASD's properties for creating a learning environment. What is more important, is that autonomous teams may easily create strong levels of concertive control on its members (Barker 1993; Lysgaard 2001), which may actually inhibit member's willingness to participate in activities related to creating new and novel knowledge (Hoegl and Parboteeah 2007). This may also easily be emphasized in ICT based work environments due to added possibilities of discretion and control given by ICT (Powell and Snellman 2004), of which an ASD process would be clear example. Similarly, strong customer influence in the development work is likely to increase creation of such team-level norms which in turn may direct more attention to solve customers' most pressing needs, thus further inhibiting efforts to explore new territories of knowledge. Team-level norms, demanding equality in a range of ways fundamentally change the working environment of the individual (Lysgaard 2001). A substantive rationality develops naturally and almost invisibly within autonomous groups, claims Barker (1993). An important enabling psychological factor for double-loop learning is trust between the people in the group (Argyris and Schön 1996). Without this trust, it becomes impossible to uncover and question the real norms, values and assumptions that underlie the present status. It is difficult to see how such trust can be established in an ASD team under constant pressure from peers, management and customers to conform to the group's norms, showing continuous progress and deliver value-adding contribution to this iteration's version of the software system. Besides, we can assume that there are primarily commercial interests between the software organization and the customers. Such interests will complicate the development of a shared understanding of the deeper norms and values as commercial relationships draw inherent benefits from hiding information.

In these arguments lies the paradox. “The actions we take to promote productive organizational learning actually inhibit deeper learning.” (Argyris and Schön 1996 p.281) Any process has standardization, and variation avoidance, as a basic disposition. Creative, exploratory work in software engineering is most likely hindered by processes altogether (Armour 2001). Even if ASD, rhetorically, would allow for open collaboration and deeper learning in a wide community, the ASD methods stress collaboration with one group of people; the customer. This becomes more clear when we consider the practical consequence of topic 1 and 7 in Table 1 (“Fundamental Assumption” and “Quality Control”). In a hypothetical ASD work situation, this translates roughly to the following scenario: The customer explains the required functionality for the particular part of the software program, the developer creates this part of the software, demonstrates for the customer that it is indeed working, and listens for any further directions regarding what is needed to finish it. If it is deemed satisfactory, the developer continues with another part of the software. The developer, with the team surrounding, essentially works in a constant dialogue with the customer. In Scrum, a person filling the role Product Owner is representing the customer for the team. But in practice, it is very difficult to fill this role properly as the interaction can be intense and the Product Owner rarely has the knowledge of the developer that is required to solve the particular task at hand. Thus, communication and coordination easily bypass the Product Owner. For XP, an on-site customer is prescribed. Thus, the customer has physical access to the developers as well. While the basic motive for the close customer collaboration is quick and efficient clarification of the real requirements, it also reduces the freedom of the developer to take on the role of an “outsider’s personality” less influenced by the mission-orientation of the organization which is associated with exploratory behavior by Hughes (1987 p.58-61). Rather, this pulls the developers into the loop of continuous design, rapid feedback, rapid change and continuous control of design, requirements and solutions.

It has been noted by many that development processes must be matched with the project’s characteristics and that there are “home grounds” for different process models. Nevertheless, during the last 5-7 years or so ASD has gained significant and widespread interest in industry and academia despite lacking scientific knowledge regarding the factors that constitute the claimed benefits (Dybå and Dingsøy 2008). Researchers also continue to voice the need for careful consideration and adoption of ASD (Boehm and Turner 2004; Boehm 2006). More knowledge regarding the influence of ASD on project work will be useful in guiding process adoption. Nevertheless, most organizations adopting ASD methods will adopt their own version of the method. Multiple factors contribute to local variants of process adoptions; local resistance, misunderstanding, gradual process decay or simply the fact that certain elements of the method are deemed unsuitable for the organizational context. Besides,

although the team members may claim that they are following the method, there are likely to be significant differences between the espoused process and the actual practice (Argyris and Schön 1996).

## Implications

There will of course be many occurrences of exploratory knowledge creation in ASD projects. This will happen, irrespectively of the organization of the software project. Innovative software systems have appeared ever since the beginning of the industry, stemming from teams' ability to explore new fields of knowledge. Besides, it is not possible to objectively and accurately classify a ASD project from a traditional project. There is no "benchmark" to measure agility in this respect. We can naturally make subjective judgments for the agility in a project, but trying to measure this would be an impossible task. Further, we can see complex relationships at work. The collaboration within the teams, the attention to interaction and frequent discussions as well as emphasis on reducing formal bureaucracy through minimizing documentation may well contribute to establish a team climate for experimentation and exploratory knowledge creation. Also, the impact of concertive control in team members willingness to openly share ideas and reflect upon deeper level norms and values is probably greatly influenced by the organizational environment and management's attention. Besides, multiple researchers have noted that knowledge creation and innovation cannot be managed, it can only be supported (Trott 2004) or enabled (von Krogh, Ichijo et al. 2000). Therefore, it seems most useful to pursue increased understanding of the complexities of relationships in ASD praxis using qualitative and exploratory research methods.

We have argued that ASD, in praxis, will promote exploitation of existing knowledge much stronger than exploration of new knowledge. In fact, it seems likely that other development paradigms, such as OSS, support exploratory knowledge creation much better. But OSS is unlikely to fit all the demands for a software organization. OSS is essentially chaotic. There is little control as to what, if anything, will be produced. A compromise between the open knowledge creation model of OSS and the more structured ASD process might be an interesting path, for example in the shape of the Communities Of Creation model proposed by Sawhney and Prandelli (2004). Sawhney and Prandelli view organizational survival in turbulent environment as a matter of balancing order and chaos in an evolutionary manner. Their proposed COC model seeks to maintain order by legislating membership procedures. On the other hand, COC enables a certain amount of chaos by facilitating collaboration between people of widely different backgrounds.

For example, gathering ideas and collaborating in open knowledge exchange networks are excellent sources for exploring new territories of knowledge. But

the constant verification and check with customer demands might be a hindrance. So, rather than focusing on customer collaboration, the process should rather be focused towards open collaboration, including customers, for particular areas of the project. Another futile path might be a more strict policy regarding customer collaboration, for example by limiting it to the initial phases of development, which was proposed by Gassmann, Sandmeier et al. (2006). This could become rather complicated in software projects, however, due to the iterative nature of ASD, which would require careful planning of involvement in each increment and thus increase the pressure to formalize customer involvement. From previous investigations, flexible development practices have been found to complicate involvement of customers (Fægri and Hanssen 2007). Nevertheless, organizations innovate effectively by reshuffling existing, proven routines (Nelson and Winter 1982). From experience with ASD processes organizations are likely to gain knowledge regarding both positive and negative effects which again can contribute to the continuing search for still better software processes.

A continuing empirical investigation, in the form of an action research program among a set of Norwegian software organizations, focusing on concertive control, customer involvement and exploratory knowledge creation is currently being undertaken by the author.

## References

- P. Abrahamsson, O. Salo, J. Ronkainen and J. Warsta (2002). Agile software development methods - Review and analysis. *Technical report 478*, VTT Electronics: 112.
- M.E. Adams, G.S. Day and D. Dougherty (1998). "Enhancing new product development performance: An organizational learning perspective," *Journal of Product Innovation Management* **15**(5): 403-422.
- T.M. Amabile, C.N. Hadley and S.J. Kramer (2002). "Creativity under the gun," *Harvard Business Review* **80**(8): 52-+.
- C. Argyris and D.A. Schön (1996). *Organizational learning II: Theory, method, and practice*. Reading, Massachusetts: Addison-Wesley.
- P.G. Armour (2001). "The business of software: Matching process to types of teams," *Communications of the ACM* **44**(7): 21-23.
- B. Bahli and E.S.A. Zeid (2005). "The role of knowledge creation in adopting extreme programming model: an empirical study." *ITI 3rd International Conference on Information and Communications Technology: Enabling Technologies for the New Knowledge Society*
- J.R. Barker (1993). "Tightening the iron cage: Concertive control in self-managing teams," *Administrative Science Quarterly* **38**: 408-437.
- R. Baskerville, B. Ramesh, L. Levine, J. Pries-Heje and S. Slaughter (2003). "Is internet-speed software development different?," *IEEE Software*: 70-77.
- K. Beck (1999). "Embracing change with extreme programming," *IEEE computer* **32**(10): 70-77.
- P.L. Berger and T. Luckmann (1966). *Den samfunnsskapte virkelighet*. Bergen: Fagbokforlaget AS.
- R. Blauner (1964). *Alienation and freedom*. Chicago: The University of Chicago Press.
- B. Boehm (2006). "A view of 20th and 21st century software engineering." *Proceedings of the 28th international conference on software engineering*, Shanghai, China, ACM Press.

- B. Boehm and R. Turner (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Boston, MA: Addison-Wesley.
- H. Braverman (1974). *Labor and monopoly capital: the degradation of work in the twentieth century*. New York: Monthly Review Press.
- J.S. Brown and P. Duguid (1991). "Organizational learning and communities-of-practice: Toward a unified view of working, learning, and innovation," *Organization Science* **2**(1).
- N. Brunsson (2002). *The organization of hypocrisy: Talk, decisions and actions in organizations* (2nd ed.). Copenhagen: Abstrakt Forlag.
- C.M. Christensen (2003). *The innovators dilemma: When new technologies cause great firms to fail*. Cambridge, MA: HBS Press.
- S.R. Clegg (1989). *Frameworks of power*: Sage Publications.
- M.A. Cusumano (2004). *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know in Good Times and Bad*. New York, NY: Free Press.
- R. Dahl (1963). *Moderne politisk analyse*.
- P.F. Drucker (2007). *Management challenges for the 21st century*. Amsterdam: Elsevier.
- T. Dybå (2000). "Improvisation in small software organizations," *IEEE Software* **17**(5): 82-87.
- T. Dybå (2003). A dynamic model for software engineering knowledge creation. In *Managing software engineering knowledge*. A. Aurum, R. Jeffrey, C. Wohlin and M. Handzic, eds. Berlin, Springer-Verlag: 95-117.
- T. Dybå and T. Dingsøy (2008). "Empirical studies of agile software development: a systematic review," *Information and Software Technology*.
- A.C. Edmondson (2002). "The local and variegated nature of learning in organizations: A group-level perspective," *Organization Science* **13**(2): 128-146.
- T.E. Fægri and G.K. Hanssen (2007). "Collaboration, process control and fragility in evolutionary product development," *IEEE Software* **24**(3): 96-104.
- O. Gassmann, P. Sandmeier and C.H. Wecht (2006). "Extreme customer innovation in the front-end: learning from a new software paradigm," *International Journal of Technology Management* **33**(1): 46-66.
- R.L. Glass (2006). *Software Creativity 2.0* (2.ed): Developer .\* Books.
- S. Gourlay (2006). "Conceptualizing knowledge creation: A critique of Nonaka's theory," *Journal of Management Studies* **43**(7): 1415-1436.
- D.J. Greenwood and M. Levin (2007). *Introduction to action research: Social research for social change* (2nd Edition): SAGE Publications.
- M.J. Hatch (2006). *Organisasjonsteori: moderne, symbolske og postmoderne perspektiver*. Oslo: abstrakt forlag.
- J. Highsmith and A. Cockburn (2001). "Agile software development: The business of innovation," *IEEE Computer* **34**(9): 120-127.
- M. Hoegl and K.P. Parboteeah (2007). "Creativity in innovative projects: How teamwork matters," *Journal of Engineering and Technology Management* **24**: 148-166.
- T.P. Hughes (1987). The evolution of large technological systems. In *The social construction of technology systems: New directions in the sociology and history of technology*. W. E. Bijker, T. P. Hughes and T. J. Pinch, eds. Cambridge, Mass., MIT Press: 51-82.
- W.S. Humphrey (1997). *Managing technical people: Innovation, teamwork, and the software process*. Reading, MA: Addison-Wesley.
- C. Larman and V. Basili (2003). "Iterative and incremental development: A brief history," *IEEE Computer* **36**(6): 47-56.
- A. Leiponen (2006). "Managing knowledge for innovation: The case of business-to-business services," *Journal of Product Innovation Management* **23**(3): 238-258.
- M. Levin (1997). "Technology transfer is organizational development: an investigation into the relationship between technology transfer and organizational change," *International Journal of Technology Management* **14**(2-4): 297-308.
- S. Lukes (1974). *Power: A radical view*. London: Macmillan.

- S. Lysgaard (2001). *Arbeidskollektivet: En studie i de underordnedes sosiologi* (3.utgave). Oslo: Universitetsforlaget.
- J.G. March (1991). "Exploration and Exploitation in Organizational Learning," *Organization Science* **2**(1): 71-87.
- J.G. March and H.A. Simon (1993). *Organizations* (2nd ed.). Cambridge, Massachusetts: Blackwell Publishers.
- R. Nelson and S. Winter (1982). *An evolutionary theory of economic change*. Cambridge, MA: Harvard University Press.
- S. Nerur and V. Balijepally (2007). "Theoretical reflections on agile development methodologies: The traditional goal of optimization and control is making way for learning and innovation," *Communications of the ACM* **50**(3): 79-83.
- S. Nerur, R. Mahapatra and G. Mangalaraj (2005). "Challenges of migrating to agile methodologies," *Communications of the ACM* **48**(5): 72-78.
- T.J. Pinch and W.E. Bijker (1987). The social construction of facts and artifacts: Or how the sociology of science and the sociology of technology might benefit each other. In *The social construction of technology systems: New directions in the sociology and history of technology*. W. E. Bijker, T. P. Hughes and T. J. Pinch, eds. Cambridge, Mass., MIT Press: 17-50.
- M. Polanyi (2000). *Den tause dimensjonen*. Valdres, Norway: Spartacus Forlag.
- W.W. Powell and K. Snellman (2004). "The knowledge economy," *Annual Review of Sociology* **30**: 199-220.
- W.W. Royce (1970). "Managing the development of large software systems." *IEEE WESTCON.*, Los Angeles, CA.
- M. Sawhney and E. Prandelli (2004). Communities of creation: managing distributed innovation in turbulent markets. In *How organizations learn: Managing the search for knowledge*. K. Starkey, S. Tempest and A. McKinlay, eds. London, Thomson: 271-300.
- K. Schwaber (2007). *The enterprise and Scrum*. Redmond, Washington: Microsoft Press.
- K. Schwaber and M. Beedle (2002). *Agile software development with Scrum*. New Jersey: Prentice Hall.
- D.A. Schön (1983). *The reflective practitioner: How professionals think in actions*. New York: Basic Books.
- S.A. Slaughter, L. Levine, B. Ramesh, J. Pries-Heje and R. Baskerville (2006). "Aligning software processes with strategy," *MIS Quarterly* **30**(4): 891-918.
- I. Sommerville (1996). "Software process models," *ACM Computing Surveys* **28**(1): 269-271.
- T. Sørhaug (2004). *Managementlitet og autoritetens forvandling*. Bergen: Fagbokforlaget.
- H. Takeuchi and I. Nonaka (1986). "The new new product development game," *Harvard Business Review*.
- T. Tanayama (2002). Empirical analysis of processes underlying various technological innovations. *VTT Publications 463*: 115p.
- F.W. Taylor (1911). *The principles of scientific management*. New York, NY: Harper & Row.
- E. Trist (1981). The evolution of socio-technical systems. *Occasional paper No. 2*. Toronto, Quality of Working Life Centre.
- E.L. Trist and K.W. Bamforth (1951). "Some consequences of the Longwall method of coal getting," *Human Relations* **4**(1): 3-33.
- P. Trott (2004). *Innovation Management and New Product Development* (3.ed): Financial Times Prentice Hall.
- E. Von Hippel (2005). *Democratizing innovation*. Cambridge, Massachusetts: MIT Press.
- G. von Krogh, K. Ichijo and I. Nonaka (2000). *Enabling knowledge creation*. New York, NY: Oxford University Press.
- M. Weber (2000). *Makt og byråkrati*. Trondheim: Gyldendal.