The Knowledge Management Strategy of Agile Software Development

Peter Wendorff, ASSET GmbH, Germany, P.Wendorff@t-online.de
Dace Apshvalka, Riga Technical University, Latvia, Dace.Apshvalka@gmail.com

Abstract
In software development the most important asset is knowledge. Much of this knowledge is tacit, and given its elusive nature, this creates a challenge for organisations. In practice most organisational KM strategies are codification strategies or personalisation strategies. We use this distinction to guide a review of typical practices in agile software development that address knowledge management. Based on this survey we conclude that agile software development implies a personalisation strategy.

Keywords: knowledge, management, strategy, codification, personalisation

1. Introduction

In software development (SD) the most important asset is knowledge (Rus and Lindvall 2002). One common distinction made in knowledge management (KM) is that between explicit knowledge and tacit knowledge.

Much of the knowledge in SD is tacit (Rus and Lindvall 2002). Organisations that develop software inevitably rely on corresponding tacit knowledge as part of their intellectual capital. Given the elusive nature of tacit knowledge this creates a challenge, as Rus and Lindvall (2002: 26) humourously illustrate: "The major problem with intellectual capital is that it has legs and walks home every day. At the same rate experience walks out the door, inexperience walks in the door."

Since the late 1990s a number of so-called "agile" software development methods have been proposed and have challenged the hegemony of the traditional, so-called "plan-driven" methods since then, leading to an ongoing dispute (Boehm and Turner 2004). For example, in Beck (2000) one of the most popular agile methods called "Extreme Programming" (XP) is enthusiastically described by one of its originators, while Stephens and Rosenberg (2003) voice harsh criticism of XP.

Glass (2001: 12) uses the term "methodology wars" to characterise the often acrimonious dispute between advocates of agile and plan-driven SD. He notes that with the publication of the "Manifesto for Agile Software Development" (Agile Alliance 2001) the dispute became more intense.


In this paper we look at key practices in agile software development that relate to KM and discuss the assumptions that motivate them. Our first goal is to achieve a clear characterisation of the KM strategy of agile SD. Our second goal is to identify conditions that make the successful implementation of this strategy likely.

In section 2 we shortly explain the role of KM in SD. In section 3 we characterise two fundamental strategic choices for KM: personalisation and codification. In section 4 we investigate typical practices in agile SD that relate to KM. In section 5 we identify the implicit KM strategy in agile SD and derive some conditions that make its successful implementation more likely.

2. Knowledge Management in Software Projects

The most critical asset in software engineering is intellectual capital. Rus and Lindvall (2002) point out that many practices in software engineering are closely related to KM, although this is often not noticed, for example:
• design patterns
• pair programming
• prescriptive SD methods

Rus and Lindvall (2002: 33) note that decisions in SD are often governed by personal "gut feeling" and experience, "but because software development is such a complex and diverse process, gut feelings might not be sufficient, and not all managers have extensive experience". In the 1990s these concerns have led to an emphasis on documented, prescriptive, plan-driven SD processes.

This focus on process has often resulted in a lack of attention for the unique abilities of the human mind, as Highsmith (2002: 373) points out: "However, beginning with the heavy emphasis on software process engineering [...] we’ve sacrificed the very characteristic that creates great software - or any great product, for that matter - reliance on individual strengths."

In KM "gut feeling" and personal experience represent tacit, personal knowledge. Organisations have two opposing strategic options how to deal with the tacit, personal knowledge of their members. They can either leave it tacit and personal, or they can try to turn it into explicit, organisational knowledge (Jashapara 2004).

3. Knowledge Management Strategy

Interestingly, many SD methods do not formulate an explicit KM strategy (Rus and Lindvall 2002). Instead they define numerous practices that relate to KM. One defining difference between agile and plan-driven methods is their attitude towards knowledge (Boehm and Turner 2004).

In practice organisations have two fundamental, opposing strategic options to deal with knowledge. They can use a codification strategy or a personalisation strategy (Hansen, Nohria and Tierney 1999; Jashapara 2004).

Typical elements of a KM codification strategy are:
(1) reliance on sophisticated technology (e.g., processes and tools)
(2) orientation towards explicit knowledge (e.g., solution templates)
(3) knowledge transfer through documents (e.g., formal document workflows)
(4) channelling of information (e.g., access to databases)
(5) focus on efficiency (e.g., through standardisation)

Typical elements of a KM personalisation strategy are:
(1) reliance on proficient people (e.g., individual development)
(2) orientation towards tacit knowledge (e.g., emergent solutions)
(3) knowledge transfer through conversation (e.g., dialogue and discussion)
(4) channelling of expertise (e.g., access to experts)
(5) focus on effectiveness (e.g., through flexibility)

4. Agile Software Development Practices

In this section we use the five elements of a KM personalisation strategy given in section 3 to identify and group typical agile SD practices.

4.1 Reliance on Proficient People

Advocates of agile software development assume that project success is mainly dependent on people factors instead of technological factors, and accordingly they focus on the creation of a learning environment where individuals can grow.

For example, Cockburn (2002: 43) claims: "Purely people factors predict project trajectories quite well, overriding choice of process or technology." This view is clearly shared by the Agile Alliance (2001), that proclaims the value of "Individuals and interactions over processes and tools" and urges "Build projects around motivated individuals. [...]".
Beck (2000) defines a set of four values: communication, simplicity, feedback and courage. Communication does not address technical skills, instead it clearly addresses social skills.

It is important to note that all agile approaches place an equally high value on social skills as on technical skills. In this sense Boehm and Turner (2004: 46) remark that "agile methods tend to need a richer mix of higher-skilled people."

This emphasis on social skills is in agreement with many prominent authors, for example DeMarco and Lister (1999: 4), who claim that in software engineering, "the major problems of our work are not so much technological as sociological in nature."

Boehm and Turner (2004) note that agile methods strongly rely on tacit knowledge, and that accordingly they depend on the ability to cultivate and share it. Therefore, the ability of knowledge workers to manage their own knowledge is a major concern in agile methods.

Cockburn (2002: 67) identifies a number of "success modes" of humans in SD, for example, "being good at looking around", "being able to learn", "being malleable", etc. He suggests that agile SD methods promote conditions where these success modes can take effect.

Emphasis on human learning and growth is prevalent in all agile methods. The key assumption is that individuals have a natural inclination to learn and grow, and that their level of proficiency can increase considerably while they are on a project. Therefore, an important aim of agile approaches is to create conditions where individual learning is encouraged.

This is in tune with Highsmith (2004: 182), who reminds: "Ongoing staff development is every manager's job, whether she is a project manager or team leader. While project managers strive to get the right people, with both the right skill set and attitude, there is also a pervasive belief in agile projects that we can always improve performance, that we continually strive for excellence."

One way to institutionalise learning in SD is pair programming, where two developers work together on one computer (Beck 2000). Describing the effects of a junior partner regularly doing pair programming with senior partners, Beck (2000: 100) observes: "In a couple of months, typically, the gap between the partners is not nearly so noticeable as it was at first. The junior partner is typing more regularly. The pair notices that each of them has strengths and weaknesses. Productivity, quality, and satisfaction rise."

4.2 Orientation towards Tacit Knowledge

Organisations can attempt to turn the tacit knowledge of their members into explicit knowledge, or they can choose to leave it tacit, and try to develop, harness and leverage it.

Cockburn (2002) uses the metaphor of a cooperative game of invention and communication to characterise human interaction during software development. In this context he reviews the need for documentation and claims that it is an art to guess "how much can be bound in your group's oral tradition and how much has to be committed to archival documentation. [...] The correct amount of documentation is exactly that needed for the receiver to make her next move in the game. Any effort to make the models complete, correct, and current past that point is a waste of money." (Cockburn 2002: 175)

From a KM perspective, Cockburn's statement about knowledge in agile software engineering can be understood as follows. In a software project explicit and tacit knowledge always coexist. If a person tries to understand an issue he or she needs enough explicit knowledge to activate his or her corresponding tacit knowledge.

Tacit knowledge that is owned by a knowledge worker can be made explicit, but the creation of explicit knowledge (e.g., through documentation) produces costs. Lack of explicit knowledge obstructs understanding and produces costs as well. In an attempt to minimise the resulting total costs, agile SD methods aim at creating the minimum amount of explicit knowledge to enable the activation of necessary tacit knowledge.
Apart from these economic considerations Highsmith (2002: 98) points out another reason for the orientation towards tacit knowledge in agile software development: That because of the nature of software engineering much tacit knowledge cannot be made explicit. He states: "But whether you are a professional golfer or a professional software developer, the essence of your skill is tacit - it can't be written down and plunked into the next recruit. The essence of talent and skill isn't what we record on paper; it's defined by proficiency, mastery, virtuosity, savvy, and artistry. Transfer of tacit knowledge takes time, energy, and face-to-face apprenticeship."

Lindvall and Rus (2002: 36) claim that most knowledge in software engineering is not explicit, but tacit. In accordance with this view many leading agile methodologists, for example, Beck (2000), Cockburn (2002; 2005) and Highsmith (2002; 2004), believe that tacit knowledge must inevitably play an important role in software development. Therefore, they regard its successful cultivation as one key to effectiveness.

Boehm and Turner (2004: 35) observe that many agile practices "are as much about developing the team's shared tacit knowledge base as they are about getting work done". Their remark emphasises the development of shared mental models among stakeholders as a key focus of agile software development to enable the effective use of tacit knowledge.

4.3 Knowledge Transfer through Conversation

A central requirement in agile software development is knowledge transfer through human conversation, preferably face-to-face, instead of knowledge transfer based on documentation.

This preference for conversation is expressed as a value proposition of agile SD by the Agile Alliance (2001): "The most efficient and effective method of conveying information to and within a development team is face-to-face conversation."

Reliance on conversation is characteristic of KM in agile SD, as Highsmith (2004: 123-124) writes: "In the early to mid-1990s knowledge management practices placed an emphasis on explicit (written) knowledge, but in more recent years the pendulum has swung toward an emphasis on people and tacit knowledge - knowledge shared through interaction. [...] The agile philosophy is that when documentation is used to replace conversation, it is misused. When it is used to complement conversation, it is used correctly."

Cockburn (2002) provides an extensive discussion of the pros and cons of knowledge transfer through conversation in SD. He uses the term "osmotic communication" to refer to opportunistic communication within a collocated team where "information flows into the background hearing of members of the team, so that they pick up relevant information as though by osmosis" (Cockburn 2005: 24). He thinks that osmotic communication is a powerful way to cultivate frequent feedback and fast knowledge dissemination, and accordingly osmotic communication is one of the three core properties of his agile SD method Crystal Clear (Cockburn 2005).

In agile software development it is the responsibility of all stakeholders to communicate relevant information when it is needed, not when doing so is convenient. As Cockburn (2002: 99) remarks, it is important that "a person cares to notice and communicate" valuable information to others.

This duty is brought to attention in XP, for example, by introducing "communication" as an explicit, central value, because: "Problems with projects can invariably be traced back to somebody not talking to somebody else about something important" (Beck 2000: 29-30). Listening is another central element in agile software development. In XP, for example, "listening" is one of the four basic activities (Beck 2000).

Disagreement can often be overcome by reflecting on different cultural assumptions. Cockburn (2002: 109) makes the practical suggestion that conflicting parties on a project should first "pretend that the other person's sentences, however crazy they may sound to you, make sense in the other culture's value system. Listen that way first, and then decide if you still need to disagree."

This practice is similar to the use of dialogue in team learning, that is characterised by Jashapara (2004: 62) as "the free and creative exploration of complex issues involving active listening and
suspending one’s own view”. Another key technique in effective team learning is discussion. Jashapara (2004: 62) provides the following explanation: "Discussion is complementary to dialogue and is best employed in situations of convergent thinking and decision making [...]"

There is agreement among agile methodologists that the use of conversation to coordinate a software development team requires good social skills or some form of facilitation. In XP, for example, a coach is required "whose job it is to notice when people aren't communicating and reintroduce them" (Beck 2000: 30).

One of the core practices of XP, pair programming, is characterised by Beck (2000: 100) as "a dialog between two people [...] It is a conversation at many levels, assisted by and focused on a computer". He explains that pair programming is effective, because the continuously changing composition of pairs helps to disseminate information within the team, and notes that "the information becomes richer and more intense as it spreads and is enriched by the experience and insight of everyone on the team." (Beck 2000: 101)

This central role of dialogue and interaction is generally acknowledged in KM. For example, Jashapara (2004: 203) expresses the following conjecture: "The holy grail of competitive advantage may be closely related to exploiting the cognitive and social aspects of tacit knowledge. It is often through dialogue and interaction with others that we are able to contact the inner depths of our tacit knowledge and generate new ideas and insights."

Robinson and Sharp (2004) present empirical studies of programming teams using XP. They conclude that respect and trust are important prerequisites for the successful implementation of knowledge sharing through conversation.

This view is also expressed by Cockburn (2002: 101) who states: "Amicability is the willingness of people to hear the thoughts of another person with goodwill and to speak without malice. Amicability is the weaker cousin to trust. [...] I always watch the amicability level in an organization to learn to what extent information is being revealed versus concealed in conversations."

4.4 Channelling of Expertise

In agile SD channelling of expertise through direct access to experts is clearly preferred over channelling of the same information through archival means.

Channelling of expertise in agile software development becomes most visible in the strong preference for a full-time, on-site customer representative. Boehm and Turner (2004: 32) note: "Agile methods strongly depend on dedicated, collocated customer representatives to keep the project focused on adding rapid value to the organization. This generally works very well at the project level."

Channelling of expertise requires that the experts can make decisions to avoid delays in the project. Boehm and Turner (2004: 44) present the result of an empirical field study of customer representatives in agile software development projects and "found that success depends on having customer representatives who are Collaborative, Representative, Authorized, Committed, and Knowledgeable (CRACK) performers."

A call for channelling of expertise is given by the statement "Business people and developers must work together daily throughout the project" (Agile Alliance 2001). Cockburn (2002: 221) remarks on this statement: "The best links are through onsite business expertise and daily discussions, which is what the statement calls for. The word 'daily' refers to the sweet spot, where discussions are ongoing and occur on demand."

Cockburn (2002) presents a number of cases of software projects where he compares channelling of information in the form of documentation and channelling of expertise in the form of face-to-face communication with experts. From these studies he identifies face-to-face communication with on-site experts as a critical "sweet spot" of agile software development. As a result he urges for easy access to expert users, instead of mere documentation, as a highly desirable property of projects using his agile Crystal methods (Cockburn 2004).
Channelling of expertise in XP becomes particularly obvious in the role assigned to external experts on a project. Beck (2000: 147) defines the team's goal: "Their goal is to get you to teach them how to solve their own problem. So, one or two team members will sit with you as you solve the problem. [...] And when you are done, they will most likely throw away everything you have done and do it over themselves."

An important practice used to channel expertise within an XP development team is pair programming (Beck 2000). Naturally this use of pair programming can only succeed if senior project members are willing to pair with less senior members. To ensure this, Beck offers a simple solution: The willingness to provide expertise is a binding commitment. He notes, "The rule is, if you're asked for help you have to say 'yes.'" (Beck 2000: 7)

The planning game in XP is an obvious mechanism for channelling of expertise between software developers and business domain experts. Beck provides a detailed discussion of the planning game in Beck (2000: 85-96).

4.5 Focus on Effectiveness

Koontz and Weihrich (1990: 10) define: "Effectiveness is the achievement of objectives. Efficiency is the achievement of the ends with the least amount of resources". In management there are usually trade-offs between effectiveness and efficiency. In agile SD the focus is firmly set on effectiveness as primary goal.

In the Manifesto for Agile Software Development and the corresponding Principles behind the Agile Manifesto, the Agile Alliance (2001) tends to focus on effectiveness: The delivery of working software is proclaimed as primary objective of an agile software development project. However, there is no stated commitment to contribute to the organisation's infrastructure beyond the project level.

A look at the writings of leading agile experts, e.g. Beck (2000), Cockburn (2002; 2005) and Highsmith (2002; 2004) confirms that they mainly discuss issues of short-term effectiveness within the scope of a single project. The primary goal is to provide a product that meets the customer's requirements as much as possible.

There is often a downside to the focus on effectiveness, that is brought to attention by Boehm and Turner (2004: 29): "Agile methods concentrate on delivering a specific software product, on time, that completely satisfies a customer. As such, the scope of concern is focused on the product at hand and generally ignores problems that may occur later. There is little, if any, concern with the organization beyond the project except as supporting or interfering with the development."

Typical practices of many plan-driven methods to increase SD efficiency at the scope of an organisation are conspicuously absent in agile methods. For example, there is no attempt to collect performance data on projects to optimise project management on later projects, and there is no requirement to document lessons learned during one project for use on other projects, etc. (Boehm and Turner 2004)

Standardisation of development processes is another way to increase efficiency in SD. However, this strategy is usually not pursued in agile SD. For example, Cockburn (2004: xviii) encourages teams to develop their own methods "to fit the particular personalities on the team, the current local working environment, and the peculiarities of the specific assignment."

Paulk (2002) notes that this focus on a single project can be highly effective for a single project, but warns that it makes the institutionalisation of good engineering and management practices across an entire organisation difficult. As a result effectiveness achieved on a particular project may cause inefficiency at the organisational level.

These concerns resonate with results of field studies of XP in large organisations, presented by Lindvall et al. (2004: 30), who point out that frequently some XP practices turned out to be unsuitable in an organisational context, because often "the challenges lie not in the agile project itself and the new practices it puts in place, but in the interface between the new and existing practices. In a large
organization, a project cannot be truly independent, but must interact with and follow the rules of the organization overall."

5. Knowledge Management Strategy in Agile Software Development

Our survey in section 4 shows that many key practices in agile software development are closely related to KM and imply a personalisation strategy. The quotations from the Manifesto for Agile Software Development (Agile Alliance 2001) presented in section 4 as well as the ideas expressed by leading agile methodologists, e.g., Beck (2000), Cockburn (2002), Highsmith (2002), fully corroborate this impression.

Kneuper (2002) provides a comprehensive overview of KM in plan-driven SD. His analysis focuses on plan-driven approaches, and referring to Hansen, Nohria and Tierney (1999) arrives at the conclusion that plan-driven SD implies a codification strategy.

Based on Kneuper (2002) and our own analysis in this paper we can summarise as follows:

- Plan-driven and agile SD differ significantly in their implicit KM practices
- Agile SD implies a personalisation strategy
- Plan-driven SD implies a codification strategy

Our conclusions are in agreement with the findings of Boehm and Turner (2004) as well as Paulk (2002) who compare plan-driven and agile software development and note that plan-driven approaches generally prefer codification of knowledge while agile approaches mainly attempt to cultivate tacit knowledge.

One of the most pressing questions in contemporary software engineering is to find criteria that can guide an appropriate selection of plan-driven and agile software development approaches for a given project (Glass 2001). This managerial decision problem is the core subject of Boehm and Turner (2004), and they present a number of criteria, based on theoretical considerations, empirical research and personal experience, that may help managers to make an informed choice.

Unlike the general perspective adopted by Boehm and Turner (2004) we have adopted a specific KM perspective in this paper. Based on our conclusion that agile software development requires the implementation of a personalisation strategy for KM we will now review the decision criteria presented by Hansen, Nohria and Tierney (1999) to address the managerial decision problem of SD method selection for a given project. Hansen, Nohria and Tierney (1999) frame their criteria in three questions, that we have adapted here to software engineering, and that managers should consider:

- Do you plan to develop families of similar software products (e.g., software product lines) in the future? If this is the case then the exploitation of reuse economics may be the better strategic choice, and this rather suggests plan-driven software development.
- Is the required application domain knowledge already available in suitable, codified form? In many mature application domains, e.g., law, accounting, engineering, etc., large bodies of proven, codified knowledge are readily available and point to a tradition of careful codification. If this is the case, and the planned software project is conventional rather than innovative, then a codification strategy and plan-driven development may be more appropriate.
- Do the stakeholders of the software project feel more comfortable with codified knowledge? KM strategies are closely related to individual preferences and organisational culture. If there is a clear preference on these grounds for the use of codified knowledge, this is unlikely to be changed easily in the course of a software project. In that case plan-driven software development is more advisable.

From our detailed examination of agile SD practices in section 4 some additional criteria that address the managerial decision problem of SD method selection can be derived. Again we frame these criteria as questions that managers should consider:

- Do you have a team with a high level of social skills? Agile SD methods critically depend on a team of very proficient people who are able to create a collaborative culture based on openness, trust, respect, discipline, responsibility and autonomy. Only if the critical social skills needed for a
self-organised team building process are present, is it reasonable to assume that this essential process will take place at the start of a project.

- Do you know and accept the disadvantages of tacit knowledge in an organisation? Agile SD is strongly oriented towards tacit knowledge that stays hidden in heads and walks out of the office every evening. It may be very tempting to focus on the stimulating effect that skilful orientation towards tacit knowledge can cause in human collaboration, but huge problems will almost certainly arise from a naive, unskilful reliance on it. This applies to software development in particular, where often key members on the project are temporarily contracted consultants. The tacit knowledge of these people walks out of the door every evening, and frequently at the end of the project it walks out of the door forever.

- Do you believe that conversation is enough for you to manage a project? One central belief in the agile community is the assumption that face-to-face communication is the most effective and efficient way to communicate within a team and on a project. Many experts doubt this agile tenet. Whatever the opinion of a manager may be, an irreconcilable clash of communication styles would lead to frustrations and problems.

- Do you know the costs that channelling of expertise may cause? An inevitable cost driver in agile SD is the need for a CRACK customer representative. There is good reason to believe that a non-CRACK representative is unsuitable for agile SD. Often it will be difficult to identify a CRACK representative at all, but it may even be more problematic for an organisation to burden such a valuable and versatile person with an IT project.

- Do you prefer short-term effectiveness over long-term efficiency? Many experts agree that agile practices can be very effective at the project level. However, some of these practices achieve effectiveness in a single project to the detriment of overall organisational efficiency. It is important to realise that agile software development is mainly concerned with short-term project effectiveness and not with long-term organisational efficiency.

6. Conclusion

Currently there is an intense discussion in software engineering about the defining characteristics of plan-driven and agile SD and criteria that can help to make an appropriate choice between the two competing approaches.

In this paper we have used a KM perspective to look at typical practices in agile SD. Using the established distinction between codification and personalisation strategies in KM we found that these agile practices imply a personalisation strategy. Other researchers have found that plan-driven SD implies a codification strategy.

Agile SD is critically dependent on the successful implementation of a personalisation strategy. Based on this realisation we could derive some conditions that make success more likely.

These conditions correspond well to findings that have been reported before by other researchers. To make these conditions more accessible we framed them into questions that managers should consider before they choose between plan-driven and agile SD.

Our analysis adds insight into the nature of agile SD because it is firmly based on established concepts and robust theories that have been developed in KM. In this way our work promotes integrative, interdisciplinary research.

The practical value of this theory-based approach becomes apparent in the clarity of the questions for managers. We are confident that these questions can help managers make informed decisions based on solid scientific research.

References