Barriers to Learning in Agile Software Development Projects

Jeffry S. Babb[1], Rashina Hoda[2], and Jacob Nørbjerg[3]

[1] Department of Computer Information and Decision Management, West Texas A&M University, 2403 Russell Long Blvd. Canyon, Texas USA, 79016
jbabb@mail.wtamu.edu
[2] Electrical and Computer Engineering, The University of Auckland, 38 Princes St, Auckland, New Zealand
r.hoda@auckland.ac.nz
[3] Department of IT Management, Copenhagen Business School, 60 Howitzvej, 2000 Frederiksberg, Denmark
jno.itm@cbs.dk

Abstract. The adoption of agile methods promises many advantages for individual, team, and organizational learning. However, environmental, structural, and organizational/cultural constraints often find teams adapting agile software development methods rather than engaging in full adoption. We present results from two qualitative studies of teams and organizations that have, in many cases, adapted agile software methods to suit their needs through the omission or alteration of aspects of the method. In many cases, aspects of an agile method that are most related to learning were those that were modified or omitted. This paper utilizes the results of these studies to identify common and emergent barriers to learning. Often these barriers to learning exist according to organizational culture and the extent to which that culture influences attitudes, norms, and behaviors pertaining to learning. We present these barriers to learning and provide insight to the causes, effects, and potential ameliorations for these barriers.

Keywords: Agile software development, learning, organizational culture, XP, Scrum, Dialogical Action Research, Grounded Theory.

1 Introduction

Organizations and development teams adopting agile principles and practices are often faced with dilemmas governing the degree to which these practices should be adopted [5]. Moreover, given that many agile methods stress principles and practice over plan and prediction, Beck [6] asserts that, while the whole adoption of an agile method, such as XP, will realize a synergy that is greater than the sum or parts, strict orthodoxy in the use of the methods is not prescribed or mandated. As such, a wide range of choices, complications, and barriers exist for those who adopt an agile method. This paper is concerned with the implications of partial and modified adoption,

H. Baumeister and B. Weber (Eds.): XP 2013, LNBIP 149, pp. 1–15, 2013.

and the issues surrounding the effective utilization of agile methods as pertains to organizational culture and learning.

Some of the research questions we aim to answer are: How are XP and Scrum practices related to learning? Which practices support learning about the customer's needs and how the product under development will help meet those needs, and which practices support skills development (improving developers' capability, their knowledge about tools, change/improve processes and practices etc.)?

In this paper we share experiences from two studies, both conducted via engaged, action-oriented, and evidence-grounded methods, whereupon we seek to illuminate the issues highlighted in our research questions. This paper offers a post-hoc reflection on the outcomes of these studies as they relate to our research questions. Our first study focuses on the adoption issues related to individual and team learning in the adoption and adaptation of XP into the software practices of a small shop in Virginia, USA. Our second examines agile adoption and use among 58 practitioners in 23 different organizations, in both New Zealand and India. In both studies the analytic processes of Glaser's grounded theory research techniques are used to distill the patterns of attitudes and behaviors which inform our observations regarding the barriers to learning in the use of agile methods.

The common patterns which emerge between the two studies suggest that barriers to learning can be classified into four main areas: Multiple Goals (Projects), Excessive Iteration Pressure, Level of Customer Involvement, and Organizational Culture. As individual, team, and customer/organizational learning are each high probability outcomes of the utilization of agile methods, the emergence of barriers to this learning may be counter intuitive, given the inherent propensity for learning as a result of adoption [29]. However, in the two studies highlighted there were indeed various barriers to learning which emerged as the result of the adoption, and in some cases adaptation, of agile methods.

The rest of the paper is structured as follows: Section 2 presents the related work in this area, followed by the Research Design in section 3. We then present the Results of the two studies in section 4, followed by a discussion of the results and the Conclusion.

2 Related Works: Agile Methods, Knowledge and Learning

The agile approach to software development emphasize team-work, a situated, iterative and emerging solution process, and personalized knowledge capture and sharing [3,8, 9, 15, 18, 26, 27, 29]. Chau et al. [8] compare the strategies of agile and so-called traditional or Tayloristic software development and discuss how agile practices and principles support a personalized and team based approach to knowledge sharing and learning in software development.

Melnik and Maurer [25] argue the importance of direct personal relationships for effective knowledge exchange. Based on an experiment they argue that intermediaries and documentation based communication lead to distortions and information loss in when passing information from requirements analysts to software designers.

The implication is that agile software development projects must have direct personal interactions between different stakeholders; e.g. as expressed in the principles of customer-on-site (XP) and frequent and direct customer contact (Scrum).

Studies of agile practice show, however, that agile teams rarely adopt all agile practices, and that they will often adapt the practices they use to local circumstances and contexts [3, 7, 16, 17, 24, 31]. The question, therefore, arises how the adaptation, manifested as either non-use or modification, of particular agile practices impact knowledge building, sharing and learning in agile software development teams and projects. In this paper, we will explore this question through the analysis of agile practices.

Based on Schön's [30, 31] theory of the reflective practitioner, as well as on earlier research about knowledge in software development, we identify three kinds of knowledge building, sharing and reflection in agile software development projects: 1) knowledge about the product being developed, 2) the skills and experience required to build the product, 3) and ongoing reflection on the process itself. In the next section we will give a brief account of the underlying theory and concepts, and elaborate on our understanding of the three types of knowledge.

2.1 Knowledge and Learning in Agile Software Development

The agile development process is iterative, with several releases of intermediate products towards the culmination in a final product. The process embraces change as the developers' and the customer's understanding of the problem and the desired qualities of the solution emerge. Thus, agile software development processes resemble Schön's [30] description of how skilled designers solve difficult problems in areas such as architecture, management and industrial design [2, 13, 26]. According to this view on design and problem solving, problems are complex and multi-faceted and the properties of the "correct" or "best" solution are not easily determined beforehand. Nor can the solution be found through a pre-determined set of steps; i.e. a method. To solve such problems, the skilled designer, the *reflective practitioner,* engages in *reflection-in-action*, an "ongoing conversation with the situation", sketching and testing solutions against his understanding - or framing - of the problem, changing and evolving both the problem framing, the understanding of the desired properties of the solution, and the solution itself, in the process [30, 31].

Reflection-in-action depends upon the knowledge and skills of the practitioner and at the same time adds to this knowledge. As the process progresses, his understanding and framing of the problem and the solution change and deepen. In the process, he draws upon his *repertoire* of previous problems, fragments of solutions, tools and techniques. The *repertoire* influences both how he frames the problem, his solution process, and the solutions he develops in the process.

This description of design and problem solving, and how it relies on the knowledge and skills of the designer, resonates well with studies of software design practice. In a study of how software developers solve a difficult design problem, Guindon [12, 14] describes how previous experience and familiarity with specific programming techniques and design patterns formed the designer's appreciation of the problem and

choice of a solution. She also observes how the designers' understanding of the problem and hence of the requirements for the solution, change and deepen as they repeatedly assess the evolving solution against the problem.

The agile development process is strongly related to *reflective* practice as stated above. One should bear in mind, however, that Schön [29] discusses the problem solving behaviour of the individual practitioner, whereas agile development is carried out by teams of developers in close collaboration with a customer. Thus, an agile development process must enable sharing of experience and skills within the developer team, as well as between the team and the customer. Agile software development practices, such as *customer-on-site, frequent releases, planning game, pair programming, user stories*, *acceptance tests*, and *refactoring* all support this goal [2, 15, 24, 26].

A software process that is not constantly monitored and improved risks *process erosion* [8] and ongoing learning, reflection and improvement are an intrinsic part of an agile team's responsibilities [8, 31]. Schön [30] uses the term *reflection-on-action* to describe how the practitioner reflects upon, and improves his solution process and its outcome. Like *reflection*-in-*action*, *reflection*-on-*action* is a personalized process that builds on and improves the individual practitioner's experience and expertise, but team-based *reflection-on-action* is implicitly supported by agile practices. Hazzan and Tomayko [14] demonstrate how the dialogue among developers and between developers and customers in agile practices such as *planning* game, *pair programming* and *refactoring* can induce team-based *reflection-on-action*. Likewise Babb and Nørbjerg [3] suggest adding techniques and tools to agile development practices in order to explicate *reflection-on-action* in agile development teams.

We derive three types of knowledge building, knowledge sharing, and learning from this account of the agile development process and reflective practice. We will use these three types in our discussion of barriers to knowledge sharing and learning in agile teams.

First, there is the knowledge about the problem and the solution that the developers and customer(s) build and share in the course of the agile project; i.e. the understandings and insights that evolve as they engage in *reflection-in-practice*. This includes an understanding of the problem and how the solution - the software - contributes to solving that problem. Note that this knowledge is not static but changes and evolves as the project progresses. Note also that this knowledge is not genuinely shared among all stakeholders: the customer cannot and should not understand all the technical details of the evolving software product, nor can the developers expect to completely share the customer's understanding of his world. Hence there is the need for ongoing dialogue between the development team and the customer.

The second type of knowledge concerns the expertise and skills that the stakeholders bring to the process - their *repertoire* Schön [30]. In the software development case this includes the developers' knowledge of software development techniques and tools, knowledge of previous solutions to "similar" problems, and their familiarity with software development practices. By sharing their *repertoire*, developers can learn from each other and thus increase the team's joint capabilities.

Finally, the agile development team must engage in *reflection-on-action* in order to learn from experiences and improve performance. The team may, for example,

discuss how to improve the accuracy of their estimates during a Sprint planning meeting, or two programmers may reflect upon whether the process they just used to solve a problem can be transferred to other situations [15].

3 Research Design

This paper is based on the findings from two separate studies of agile practices: The first is a longitudinal Action Research study in a small software company in the USA [2, 3]. The other is a Grounded Theory study of agile practices in 23 different organizations in India and New Zealand [16, 17, 18].

At the time of study, all the organizations had introduced or were in the process of introducing agile practices based on either XP, Scrum, or a combination thereof. They would also adapt the agile practices based on local needs or constraints. In this context, adaptation means that the agile team may adopt a practice, but modify it or choose to not use it at all.

For the present paper we have identified adapted practices in both studies and the underlying causes; e.g. lack of customer involvement may be a cause for adaptation of the original "customer-on-site" practice. Through identification of common patterns, we were then able to group individual causes into a smaller set of common causes which are presented later in the results section. We now describe the research set-up of the two studies.

3.1 The Longitudinal Small-Shop Study

The small shop study is a longitudinal study of the introduction of agile practices and learning tools and techniques into SSC (a fictive name), a software company in the eastern USA consisting of the owner/founder and 4 developers.

The study used Dialogical Action Research [21] which uses practitioner and researcher dialog as the principle means by which interventions are introduced into the practitioner's setting. Dialogical AR - as action research in general - proceeds in cycles, each cycle contributes to solving the practitioner's problem as well as to the researcher's knowledge and thorough reporting to the research community in general.

During the practitioner-researcher partnership, the researcher conducted interviews with the practitioners and observed them in their daily work. Interviews and observations were documented in transcripts, supplemental documents, and field notes. During the 9 months of fieldwork the researcher was present onsite twice per week on average. Each visit usually lasted for a period of 1-4 hours. The data collected during this period consisted of:

- 26 recorded and transcribed dialogs with the company owner and lead developer, and various combinations of the team based on progress in a given iteration of the Dialogical AR cycle.
- Internal SSC documents
- Field notes taken while observing the practitioners' work

The data collected during the initial phase of the partnership were coded, using open, axial, and selective coding, in order to derive common themes related to SCC's method use. This initial analysis resulted in the recommendation to introduce XP into SSC and the method was introduced incrementally over the following months. The researcher continuously documented and analyzed how and why the practitioners adopted and adapted XP throughout this part of the project. It is the results from this latter analysis which forms the basis for the discussion in the present paper.

In the final phase of the Dialogical AR partnership, the researcher introduced explicit tools and techniques to support XP at SSC with on-going *reflection-on-action* at SSC. This is, however, beyond the scope of this paper and is described elsewhere [2].

3.2 The Grounded Theory Study

The Grounded Theory (GT) study was carried out over a period of 4 years using Glaser's classic GT method [11]. Using GT, the researcher – one of the authors of this paper – conducted iterative rounds of data collection and constant comparison method of text analysis. Data was collected from 58 agile practitioners in 23 different organizations in New Zealand and India through face-to-face, semi-structured interviews of approximately an hour each, using open-ended questions as well as observations of the workplaces and practices.

All participants used Scrum or a combination of Scrum and XP. All participants were practicing fundamental Agile practices such as iterative and incremental development (with varying iteration lengths), iteration planning, estimation and planning of user stories and tasks, testing, status report meetings (such as daily standup), frequent release of working software, and some form of retrospective meetings. A majority of the participants engaged in test-driven development and pair programming (on demand). Some participants were certified Scrum Masters. Participants belonged to organizations ranging from as small as 10 people to as large as 300,000 employees. Their domains ranged from health, telecommunications, entertainment, agriculture, energy, to software product development for multiple domains. The project duration varied from 2 to 12 months and team sizes ranged from 2 to 20 people on different projects. Participants varied in their levels of experience of using agile practices from novice to mature with several years' experience.

Data was analyzed using GT's open, selective, and theoretical coding procedures. Codes arising from one interview were constantly compared to those arising from all other interviews using the constant comparison method. This led to identifying common themes or patterns in data at increasing levels of abstraction. A number of findings were made from the GT study with respect to agile practices and have been described elsewhere [16, 17, 18].

We discovered a number of barriers to learning across the dimensions of *reflection-in-action*, *repertoire*, and *reflection-on-action* in the findings from both the -studies. We have analyzed examples of non-adherence or modifications to agile practices in order to identify the underlying causes of the adaptation/non-use. In each case we also identify the effect on learning and knowledge sharing. We describe these barriers in the next section.

4 Results: Barriers in Practice

In this section, we describe the categories of "barriers in practice" as identified in the two studies. These are: Multiple Goals (Projects), Excessive Iteration Pressure, Level of Customer Involvement, and Organizational Culture.

4.1 Multiple Goals (Projects)

In the longitudinal study, SSC is a small company with 20-40 individual customers at any given time. Furthermore, the company's projects spanned from as little as a week or two to three months (and beyond). The high number of customers and - very short - projects had implications for the adoption and adaptation of the practices of *pair programming* and *customer as team member*.

It was difficult for SSC to fully embrace the idea of *pair programming* although both the manager and the developers understood that this technique increases reflection and awareness among the developers and hence may contribute significantly to productivity. With each developer working on several projects simultaneously, and often only one person being active on a given project at a given point in time, it was infeasible in practice to apply *pair programming* in a systematic way. Programmers would, however, team up to explore and solve difficult programming problems and to create s*pike solutions*. In this way the programmers used *pair programming* to improve their individual and shared knowledge of programming techniques and tools. This adaptation of *pair programming*, however, curtailed the adoption of c*ollective code ownership* and sharing.

SSC could not adopt the principle of *customer as team member* in the way prescribed by the XP method. With anywhere from 20 to 40 projects underway at varying stages of completion or maintenance, augmenting the “team” by 20 to 40 members was not realistic. Instead Daphne, the founder/owner, would act as a proxy for the customer and write u*ser stories*, and later a*cceptance tests*, based on her notes or her memory of a client’s intentions. Thus, the customers' needs and intentions were not communicated directly to the developers in the customers' own language, but were mediated through Daphne's interpretations and language.

In the Grounded Theory study, we discovered a co-relation between team members being split across multiple projects and their ability to perform group programming – working together in an open-plan workspace while sharing the same code-base and collaborating closely. Group programming in agile teams provides opportunities for learning among team members.

“I think in our business, software developing, it's a complex subject and it's impossible for one person to know about everything, so it's a day-by-day thing...This is a normal step and everybody is learning each day.” – Participant P14, Developer, New Zealand

Continuous learning involves different types of learning: learning Agile practices, learning new or complex technical skills, learning cross-functional skills, and learning from the team's own experiences - all of which fuel self-improvement. Where team members were split across multiple projects, their ability to perform group programming was curtailed.

"What I think a affected our project...[the developer] was working on another project, he didn't have enough time, so he didn't have the space to chat with anybody, to discuss ideas with anybody, to work with anybody, so he was really just on his own, and I think that really impacted a lot of the work he did in the last few months ... When you're working in a team like this [Agile team] and you've got to work quite closely, the individuals in the team matter." – Participant P21, Customer Rep, New Zealand

As a consequence the benefits of group programming, such as team-based reflection-in-action as a result of working together, were diminished. On the other hand, some other teams where members were largely dedicated to single projects at a time provided a strong learning environment, especially for new-comers who would pair up with more experienced members to learn new technologies.

"I had never worked on the Spring framework before, but in this project it's completely related to Spring framework, and Spring transaction management and all, so I started learning it...we were pairing each with other, that time it was beneficial because the other person was quite okay...and he knew about the Spring frame-work and he had done it before in some other project. So it helped me to learn it more faster, because he used to say: 'okay, you have to go with this stuff, and you can do it'. So that was a major advantage." – Participant P16, Developer, New Zealand.

It was obvious that while dedicated resources on projects performing group programming were able to benefit from enhanced learning opportunities, resources split across multiple projects suffered from diminished opportunities for learning.

4.2 Excessive Iteration Pressure

We defined "iteration pressure" as the pressure to deliver to a committed team goal every iteration. Iteration pressure, in itself, is not detrimental to the team, in fact some amount of iteration pressure is necessary to motivate teams to deliver their goals. Short iteration lengths or an extremely high and unsustainable development velocity, on the other hand, can cause excessive iteration pressure. For instance, in the GT study, a developer found one week iterations to be very demanding:

"I'm always feeling the need to rush, rush, rush!...after one week [iteration], we want to remove all these stickies [tasks] from the wall. So it's always pressure...if you have [longer] development time, then I can adjust my work like if we spent a little bit longer than we expected, I can catch up next week." – Participant P15, Developer, New Zealand

Creating and maintaining a continuous learning environment requires teams to set some explicit time aside for learning each iteration. Excessive iteration pressure, on the other hand, implies they may not have any extra time to spare for learning:

"I'd be interested to learn various agile techniques for requirements gathering, such as events and themes, and I'd love to try and use some of them in an Agile project. It's just [that] I haven't really had a lot of time to think about it. [Scrum] is very action oriented." - Participant P4, Business Analyst, New Zealand

"You need to actually allow time for other team members to learn what you do and for you to learn what they do. Often we tend to fill up our sprints with so much that a good teaching environment isn't necessarily there...they can see what you're doing but you need to be able to take the time to explain in really good detail." - Participant P8, Tester, New Zealand

Excessive iteration pressure was, therefore, found to be a barrier to learning in agile teams.

4.3 Customer Involvement

The original XP and Scrum practices to support customer collaboration are the 'on-site customer' and 'product owner' respectively. In practice, several factors contribute to less than ideal levels of customer involvement. These include skepticism towards agile practices, geographic distance including off-shoring setups, inability or unwillingness to collaborate, etc. [17].

In the longitudinal small-shop study, the change to XP increased the level of ongoing interaction with customers. Negotiating requirements with a key customer had previously been the responsibility of a key partner who was the main contractor. This had led to estimation and quality issues. After the introduction of XP, Daphne, the founder and CEO of SSC, insisted on engaging directly with the customer, using user stories to capture user requirements iteratively and respond to change. A drawback of this setup was that she was the prime – sometime only – liaison with customers. As such the repertoire of learning that can be derived from interactions with the customer was limited to Daphne, while the rest of the team did not get a chance to learn about the customer domain, business cases, and requirements in the same way.

Where the teams were suffering from inadequate customer involvement, a single team representative coordinating with the customer – or a *coordinator* role – emerged in most of the relatively new agile teams in the GT study [17]. It was mostly played by a business analyst or by developers. The *coordinator* was responsible for capturing customer requirements and relaying them to the team. Similarly, they would pass on questions from the team to the customer and elicit clarifications on requirements or prioritization. Another role identified in the GT study was that of a *translator* – a person responsible for understanding and translating between business language used by customers and the technical terminology used by the team, to improve communication between the two [18]. In relatively new agile teams, the *translator* role was mostly played by a single individual usually also playing the *coordinator* role. Both these roles involved close learning and in-depth understanding about the customer domain and requirements.

Lack of these roles altogether or where these roles are limited to individuals, becomes a barrier to learning for the whole agile team. Where the *coordinator* and *translator* roles were played by single individuals on new agile teams, they were useful in overcoming the challenges of inadequate customer involvement, however, it provided limited opportunities for other members to learn about the customer domain and requirements. In more mature teams – practicing agile for more than a year – most members of the team were able to play the *coordinator* and *translator* roles and interact

directly with the customer. This provided better opportunities for all members of the team to develop a repertoire of learning about the customer and their requirements.

4.4 Organizational Culture

Organizational culture has been defined as "*a standard set of basic suppositions invented, discovered or developed by the group when learning to face problems of external adaptation and internal integration.*" [18]. Senior management has a strong contribution in setting up overall corporate vision and values and maintaining the organization culture. Agile teams require organization structures that are informal in practice, where the boundaries of hierarchy do not prohibit free flow of information and feedback. In an informal organizational structure, the senior management encourages a strong learning environment with active mechanisms for knowledge management across the board.

In the longitudinal study, at the time when XP was introduced to SSC, the founder/CEO had all important knowledge about the company's processes, customers and products, and she would work hard to "mold" new employees into her ways of thinking:

"And even though we haven't written a formal methodology, which I guess is just in my head, and I have conformed Fred [a developer] to what's in my head...Luckily, he has been trainable and has listened to what I do... Fred was content with delivering back to me exactly what I asked for so I've molded Fred into my way of thinking, so I guess the methodology is in my head." – Daphne, CEO, SSC

This knowledge transfer process took time away from other important tasks and she hoped that having a formal method which everyone knew and could use, would take some of that pressure away from her. Her strong belief that her own knowledge and capabilities held the key to productivity and quality did, however, create obstacles for the kind of team learning intended in XP and other Agile methods. She saw the method as a more effective way to codify and transfer her ideas of best practices to the developers, rather than a vehicle for genuine knowledge sharing and skill development. This was evident in her approach to *pair programming, daily stand-up meetings,* and *user stories*. She immediately valued *pair programming* for skills development and saw developer pairing as a means for skills transfer from herself to one deveoper and through him to the next, and so on. Citing her own higher skill and experience level, Daphne saw *spike solutions* and *pair programming* as means to elevate her employees' skills until parity with her own skills was reached. On the other hand she was less confident to let the developers pair on their own without her guidance and support. She would be concerned that developers would "reinvent the wheel" and spend time finding solutions to problems she had solved already.. As a consequence - and also because of the resource and structural issues discussed above - she would neither support, nor endorse *pair programming* as a practice to be used across the board.

This view on learning as transferring knowledge and skills from management to developers, creates barriers to the team's own reflection and. It is feasible that c*ollective code ownership* will remain unachievable in SSC in practice.

Daphne's influence on the introduction and use of XP also had some positive impact on learning. To her, both *user stories* and *daily stand-up meetings* became means to monitor productivity and progress. She would, therefore, actively engage in these practices, thereby reinforcing their effect on learning and reflection.

In the GT study, we found that agile organizations, where all the teams operate using agile software development, are characterized by informal organizational structures. Informality in organizational structure promotes openness. Openness was one of the most common traits mentioned by participants, that made the organizational culture conducive for agile teams. In such organizations, team members are free to voice opinions, raise concerns, and freely share knowledge within and across teams. This was achieved in a few cases through knowledge repositories in the form internal project wikis where all important project information, domain knowledge, business cases, and technical tasks were recorded.

5 Discussion

Our discussion and analysis of the results of the two studies, as they related to the emerging theme of barriers to learning, will be presented in two steps: 1) an overview of adapted practices and the implications for learning, using the, and 2) discussion of the underlying causes of the adaptation/non-use.

We first summarize results, as they pertain to each identified barrier and the effect this has on learning and knowledge sharing within the agile teams. (See table 1)

Many of the barriers to learning and knowledge sharing emerge as the result of conflict and friction between constraints endogenous to the development team and, in some cases, the organization in which the development team is located. Company size, organizational culture, principle industry type, and team size, each play an influencing role concerning the adaptation of the agile method. Whereas agile methods are generally effective, they are not so codified that complete orthodox adoption is necessary. However, while ample instruction exists on the learning cycles inherent in XP and Scrum, experiences from both studies suggest that engagement in the reflective and learning-oriented practices are not always followed or are not sufficiently institutionalized. Thus there are structural and contextual hindrances for, knowledge sharing, reflection and learning in agile projects. Perhaps as profit is largely attached to the delivery and acceptance of working software, the learning cycles that improvement team and personal development may be eschewed in favor of moving forward to the next opportunities for billable hours, progress on projects, and productivity.

The learning that arises from the use of agile methods is also manifold. Some learning is related to improvements in a team's ability to understand the requirements of the project and to adapt the changes in requirements. Since these are productive aspects of agile methods which enable the delivery of working software (and thus revenue), this type of learning would be encouraged. Other learning relates to the improvement of developers' and teams' skills and expertise, and is perhaps, although counter-intuitively, reduced or omitted in favor of learning activities which are more directly related to the bottom line.

Table 1. Summary of Barriers to Learning and Knowledge Sharing in the Adaptation of Agile Methods

Barrier	Agile practices affected	Effect on learning and knowledge sharing
Multiple goals (projects)	Customer-on-site Pair programming Collective code ownership	**The product being developed:** Developers did not have direct access to the customer's needs and requirements. Developers did not share knowledge about the emerging product.
Excessive iteration pressure	Daily stand-up Retrospectives Pair programming	**Skills and experience:** The developers had insufficent time to experiement with new techniques and exchange new ideas. **Reflection-on-action:** Not enough time to reflect upon and improve practices and results.
Customer involve-ment	Customer-on-site	**The product being developed:** Developers did not have direct access to the customer's needs and requirements.
Organiza-tional culture	Pair programming Spike solutions User stories Stand-up meetings	**Skills and experience:** Skills and experience transferred from a strong "expert" to other developers. No collective sharing **Reflection-on-action:** One person's view dominates reflec-tion and improvement activities. Reduced team-based re-flection and improvement.

Similar to Hazzan and Tomayko [15], we observe, that although practices in XP and other agile methods support reflection and learning, practical and contextual issues create barriers to learning which are not simple to resolve. That is, the learning that is possible from the utilization of agile methods is challenged by the natural entropy inherent in the particulars and context faced by a given team or organization. We conveniently characterize the constraints and proclivities of a given team as part and parcel of their organizational and/or managerial culture. Therefore, as agile methods emphasize knowledge sharing and learning at the team level, the team is situated within, influenced by, and perhaps, bound by, an organizational culture. There are many agile principles and practices, such as collective code ownershp, stand-up meetings, and retrospectives in which organizational culture is driving the barriers we identify from the highlighted studies.

Many of the barriers to learning, which resulted from ommission or partial engagement of learning-oriented agile practices, resulted from individual, team, and management dilemmas [33]. In the case of the longitundial study at SCC, many decisions made which were detrimental the realization of the full benefit of agile practices were cognizant of the potential costs. The dilemma was typically a matter of prioritizing other short-term or existential constraints. However, organizational culture can influence the degree to which a team benefits from an inherent culture or disposition towards the indoctination of learning practices.

Argyris and Schön [1] characterize the problem of cultural threats to learning as a Learning Paradox, wherein disconnect arises in what is discussable and

not-discussable in the context of organizational norms. In the case of a small shop, such as SCC, there may be an involved owner who, rightfully insists on maintaining discretion on practices. Overall, organizational culture may have great impact on whether the team will engage in reflection-in-action and double-*loop learning* depending on what is *discussable*. This was evident in the Dialogical AR partnership at SSC where certain persistent problems related to one of their strategic partnerships precluded organizational learning for the team. This was so as SSC was too constrained by and codependent with their strategic partner to allow the not-discussable to enter into their reflections and learning.

This matter of organizational culture constitutes a particularly wicked problem as a learning culture - in spite of its seemingly problematic nature - may not be easily removed, since it is tied into the company's history, and - ultimately - the funder/owner's ambition to keep the company afloat. These sort of deeper structural problems which - in practice - limit the application of certain XP elements, are perhaps unavoidable. Evidence from the Grounded Theory studies also suggest many structural constraints which, at times, revealed the pace required for constant revenue-generating and forward-moving action, left some developers in a state where the productive trains of Scrum were dutifully engaged, while some of the more reflective and learning-oriented practices were curtailed.

We are left with a central question which is only partially answered by the evidence from our studies: what is the opportunity cost of trading the organizational and team learning aspects of agile methods for their productivity and adaptability aspects? In both studies, cracks and fissures in team learning were apparent. Certainly the implications these tradeoffs have for the long-term effectiveness of the team are worth further study. Perhaps metrics for learning and greater discipline for the specification of learning could be made more integral to the agile methods. However, it may be somewhat antithetical to the premise of agile methods that learning becomes a quantifiable metric. In any case, the importance of individual and team learning as a byproduct of agile method use is quite established, however, the mechanics to ensure that learning outcomes are inculcated as being concomitantly and equally important as product outcomes are perhaps not as integral to agile methods as we purport.

6 Conclusion

Even though software companies and practitioners aim to follow agile practices, they (or their managers) face challenging conditions as identified in our two studies. The answer to this is not, however, to insist that practitioners follow agile practices to the letter or abandon the agile practices altogether. Agile practitioners often end up adapting agile practices to different contexts and constraints, thus creating barriers to knowledge sharing and learning. For example, inadequate customer involvement leads to the emergence of adapted practices of using the coordinator and translator roles in place of ‘on-site customer’ or ‘product owner’. When played strictly by single individuals on relatively new agile teams, these adapted roles limit the rest of the team’s ability to acquire knowledge and enable learning about the customer domain.

Awareness of the barriers to learning described in this paper will help agile practitioners better grasp the risks associated with adapting agile practices and consciously include opportunities for learning and knowledge sharing when practicing agile. However, awareness alone may be insufficient as many of these barriers are structurally tied to the organizational context. The dilemmas and constraints that many agile teams and practitioners face may result in practices that forestall the learning mechanisms inherent in many agile practices. While more study into this area is needed, it seems that concepts related to reflective practice hold the most promise in allowing the individual practitioner opportunities to individually react and adjust to barriers to learning in the use of agile methods.

References

1. Argyris, C., Schön, D.: Organizational Learning II – Theory, Method, and Practice. Addison-Wesley, Boston (1996)
2. Babb, J.J.: Towards a reflective-agile learning model and method in the case of small shop software development: evidence from an action research study. PhD., Virginia Commonwealth University (2009)
3. Babb, J.J., Nørbjerg, J.: A Model for Reflective Learning in Small Shop Agile Development. In: Molka-Danielsen, J., Nicolajsen, H.W., Persson, J.S. (eds.) Engaged Scandinavian Research. Selected Papers of the Information Systems Research Seminar in Scandinavia, Molde, Norway, vol. 1, pp. 23–38. Tapir Akademisk Forlag (2010)
4. Bansler, J.P., Bødker, K.: A Reappraisal of Structured Analysis: Design in an Organizational Context. ACM Transactions on Information Systems 11(2), 165–193 (1993)
5. Boehm, B., Turner, R.: Balancing Agility and Discipline: A Guide for the Perplexed. Addison-Wesley, Boston (2004)
6. Beck, K.: Extreme Programming Explained. Addison-Wesley, Boston (2000)
7. Cao, L., Mohan, K., Xu, P., Ramesh, B.: A framework for adapting agile development methodologies. European Journal of Information Systems 18, 332–343 (2009)
8. Chau, T., Maurer, F., Melnik, G.: Knowledge Sharing: Agile Methods vs. Tayloristic Methods. In: Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2003. IEEE Computer Society (2003)
9. Cockburn, A.: Agile Software Development. Addison-Wesley, Boston (2002)
10. Coleman, G., O'Connor, R.: Investigating software process in practice: A grounded theory perspective. J. Syst. Softw. 81(5), 772–784 (2008)
11. Fitzgerald, B., Russo, N.L., Stolterman, E.: Information Systems Development. Methods in Action. McGraw-Hill (2002)
12. Glaser, B., Strauss, A.L.: The Discovery of Grounded Theory. Aldine, Chicago (1967)
13. Guindon, R.: Designing the Design Process: Exploiting Opportunistic Thoughts. Human-Computer Interaction 5, 305–344 (1990)
14. Guindon, R.: Knowledge exploited by experts during software systems design. International Journal of Man-Machine Studies 33, 279–304 (1990)
15. Hazzan, O., Tomayko, J.: The Reflective Practitioner Perspective in eXtreme Programming. In: Maurer, F., Wells, D. (eds.) XP/Agile Universe 2003. LNCS, vol. 2753, pp. 51–61. Springer, Heidelberg (2003)
16. Hoda, R., Kruchten, P., Noble, J., Marshall, J.: Agility in Context. In: Object-Oriented Programming, Systmes, Languages and Applications Conference, OOPSLA 2010, Reno/Tahoe, NV. ACM (2010)

17. Hoda, R., Noble, J., Marshall, S.: Agile Undercover: When Customers Don't Collaborate. In: Sillitti, A., Martin, A., Wang, X., Whitworth, E. (eds.) XP 2010. LNBIP, vol. 48, pp. 73–87. Springer, Heidelberg (2010)
18. Holz, H., Maurer, F.: Knowledge Management Support for Distributed Agile Software Processes. In: Henninger, S., Maurer, F. (eds.) LSO 2003. LNCS, vol. 2640, pp. 60–80. Springer, Heidelberg (2003)
19. Humphrey, W.S.: Managing the Software Process. Addison-Wesley, Reading (1990)
20. Kautz, K., Madsen, S., Nørbjerg, J.: Persistent Problems and Practices in Information Systems Development. ISJ (2007) (accepted for publication)
21. Lee, A.S., Mårtensson, P.: Dialogical Action Research at Omega Corporation, Richmond, VA, pp. 1–39 (2004)
22. Madsen, S., Kautz, K., Vidgen, R.: A framework for understanding how a unique and local IS development method emerges in practice. European Journal of Information Systems 15, 225–238 (2006)
23. Mangalaraj, G., Mahapatra, R., Nerur, S.: Acceptance of software process innovations – the case of extreme programming. European Journal of Information Systems 18, 344–354 (2009)
24. Mathiassen, L., Pries-Heje, J., Ngwenyama, O. (eds.): Improving Software Organizations. From Principles to Practice, The Agile Software Development Series. Addison-Wesley, Boston (2002)
25. Melnik, G., Maurer, F.: Direct Verbal Communication as a Catalyst of Agile Knowledge Sharing. In: Agile Development Conference (ADC 2004). IEEE Computer Society (2004)
26. Moe, N.B., Dingsøyr, T., Dybå, T.: A teamwork model for understanding an agile team: A case study of a Scrum project. Information and Software Technology 52, 480–491 (2010)
27. Nerur, S., Balijepally, V.: Theoretical Reflections on Agile Development Methodologies. Commun. ACM 50(3), 79–83 (2007)
28. Rubin, K.S.: Essential Scrum: A Practical Guide to the Most Popular Agile Process. Addison-Wesley, Boston (2013)
29. Schein, E.H.: Oraganizational Culture and Leadership, 1st edn. Jossey-Bass Publishers, San Franciso (1985)
30. Schön, D.A.: The Reflective Practitioner. How Professionals Think in Action. Basic Books (1983)
31. Schön, D.A.: Designing as reflective conversation with the materials of a design situation. Knowledge-Based Systems 5(1), 3–13 (1992)
32. Senapathi, M., Srinivasan, A.: Understanding post-adoptive agile usage: An exploratory cross-case analysis. The Journal of Systems and Software 85, 1255–1268 (2012)
33. Steiner, L.: Organizational dilemmas as barriers to learning. The Learning Organization 5(4), 193–201 (1998)
34. Stolterman, E.: How System Designers Think about Design and Methods. Some Reflections Based on an Interview Study. Scandinavian Journal of Information Systems 3, 137 (1991)