XP in a Small Software Development Business: Adapting to Local Constraints

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

[1] Department of Computer Information Decision Management, West Texas A&M University, 2403 Russell Long Blvd. Canyon, Texas, 79016, USA
[2] Department of Electrical and Computer Engineering, University of Auckland City Campus, Auckland, New Zealand
[3] Department of Computer Science, University of Aalborg Selma Lagerlöfs Vej 300, 9220 Aalborg Ø, Denmark
jbabb@wtamu.edu, r.hoda@auckland.ac.nz, jacobnor@cs.aau.dk

Abstract. While small software development shops have trended towards the adoption of Agile methods, local conditions and high iteration pressure typically cause adaptations and appropriations of Agile methods. This paper shares evidence from a study concerning how a small software development company adopts and adapts, XP to suit their business. Based on a Dialogical Action Research project, the study reflects on the conditions leading to Agile process adaptation, and why ad hoc and "a la carte" approaches may be problematic. Limitations and drawbacks to aspects of XP are also discussed. The Agile practices most sustainable for small shop teams, with process maintenance and viability as a goal, are highlighted.

Keywords: Agile methods, method adoption, method adaption, local conditions, process evolution.

1 Introduction

Most software development organizations have less than 10 employees. 80% of US software companies and 89% of the IT consultancy and service companies in the Copenhagen and southern Sweden regions belong in this segment [1, 2]. Research into the practices, conditions, specific needs, constraints and challenges of small software development shops is, however, scarce [1, 3-5].

For the purpose of this paper, a small shop is not necessarily identical to a small team. A small team can exist in a small company (hence the term "small shop") or within a larger company. We use the term "small shop" as a term of convenience and also as a term that the practitioners self-identified with in the case. Usually, a small team can be considered as mostly autonomous where the small-shop team would be the most autonomous. However, a small software shop faces specific constraints and conditions which induce particular requirements and limitations on its methods and processes [6]. Unlike the small team operating in a larger company, a small shop does not have access to a support infrastructure; e.g. a quality management department

T.H. Commisso et al. (Eds.): SCIS 2014, LNBIP 186, pp. 14–29, 2014.

which is responsible for adapting new methods and techniques [3, 7, 8]. The economic constraints of a small company more profoundly enforce short intense release cycles with an often-reduced capacity to share knowledge and reflect upon experiences. Furthermore, a dynamic environment together with small diversified projects leaves little room for institutionalizing processes [6].

The Agile approach seems to fit well with the small shop setting [7], but Agile methods are rarely adopted "off the shelf", but rather adapted to fit local circumstances and constraints [4, 9-12]. Previous research into agile adaptation has, however, examined the context larger organizations (50 or more employees) but there is little research into agile adoption and adaptation in small software shops. Larger organizations have the resources and capabilities required for a planned approach to adaptation, but a small software shop will often choose a pragmatic and/or ad hoc approach to process adaptation due to their small size and resource constraints. Therefore, the resulting process may be shaped more by personal idiosyncrasies rather than analysis of the company's needs and capabilities [4].

In this paper we will describe and analyze how a small software shop adopted and adapted the agile method Extreme Programming (XP) in order to address specific issues and challenges in the company. The paper is the outcome of a Dialogical Action Research (DAR) study in a small software shop in the USA. XP was introduced into the small shop in several iterations after an initial diagnosis of the company's characteristics and needs. We characterize the use (adoption and adaptation) of the method and discuss which elements were used and which were discarded or heavily modified. We discuss how the resulting combination of software development work in the case is related to the constraints and conditions of the small software shop. An analysis of the case shows that agile methods, while helpful for small shops to improve their processes and productivity, are modified to meet the unique conditions of a small software shop.

Previous studies have identified a number of organizational and contextual factors influencing the adoption and adaptation of agile methods [10-12, 23, 24]. We supplement these studies by showing how the les-often-studied small shop context shapes XP adoption. In fact, key XP practices and recommendations, such as "having a customer on site" may be structurally incompatible with the small shop environment – and perhaps even in the context of small teams in larger organizations.

This paper also seeks answers to the expressed need for empirical studies of agile development [13, 14]. The study contributes in this regard as it is based on a longitudinal episode of DAR investigating XP adoption and use in a small-team/small-shop context.

The rest of the paper is structured as follows: first, we characterize small shop software development. Next, the process of both adopting and adapting an agile method, XP, is considered. The research method used to investigate the case, Dialogical Action Research, is outlined in terms of how this method was used to achieve the research outcomes. We next characterize how the small shop in the case, SSC, proceeded to adapt and adopt XP. The paper then proceeds to discuss the implications of agile software process adaption, with consideration for implications for theory and practice. The paper then concludes with future directions for the work.

2 Small Shop Software Development

What is now commonly called "traditional" approaches to information systems development, software engineering, and software project management grew out of the needs of large-scale industrial and military software projects; however, currently a significant portion of software products and services are increasingly produced by small teams [3]. Small teams and small projects face conditions and challenges that require methods and processes that sometimes differ from what is required in larger teams and projects [3]. Among these conditions and challenges are: (1) A lack of wider institutional/organization sanction or reinforcement of their practices and habits; (2) Less time or fewer resources to engage a wider community of professionals for reinforcement of good practices and amelioration for less effective habits; (3) A constant focus on delivery due to financial constraints, leaving less time for reflection and learning; (4) Fewer “degrees of freedom,” or avenues, from which new practices can be gleaned.

There has been little research into small team software development practices prior to the rise in the importance of personal computer and microcomputers in the 1980s and of Internet-related applications and technologies in the 1990s and 2000s. With the onset of agile software development methods, however, a considerable amount of research and field reports have concentrated on agility in software development with some reports concluding the utility of agile methods is greatest in a small-team context [7].

Small software shops, here defined as software development companies with less than 10 developers, share several of the characteristics of small teams. Like the small team within a larger company, small shops have little need for elaborate plans, change management procedures or document-based coordination and communication in general [14, 15]. However, a small team working in a large company often has access to resources not found in the small company setting. Large companies, for example, may be more apt to provide infrastructure and resources for method and tool support, quality assurance, testing, configuration management, and documentation. A larger organization may also provide training opportunities and access to a pool of knowledge and expertise, as well as institutionalized processes that novice developers can adopt.

Often, the small software shop cannot approach or sustain the support infrastructures and resources of a larger organization [8]. Typically, there are only a few developers with constrained (or non-existent) access to the breadth and depth of experience and knowledge that can be found in a larger organization. Furthermore, practitioners in the small shop may lack a formal background/training in the computing disciplines conducive to software development. Often, the small software shop operates in a highly competitive environment and has few financial resources to spare for training, education, and development beyond what developers undertake in their own time (and using their own resources). Small shop survival hinges on short delivery cycles that produce immediate value to the customer.

Under these circumstances, comprehensive planning, detailed requirements analysis, documentation, and elaborate quality assurance activities mandated by most

software development methods cede to release-and-patch strategies, incremental deliveries, and rapid (yet inconsistent) customer feedback [1, 4, 6].

The sum of the constraints faced by small software shops implies that the methods, tools, and techniques traditionally recommended in software engineering, which have been evolved over the long term in the context of the large team and/or large company, do not fit well with the small software company [3, 14, 15, 16]. This does not imply, however, that a small software shop should not strive towards a systematic and professional software development process [6, 14, 15, 16], but that methods and processes used must be adapted to the small-shop environment and constraints; i.e. designed around frequent and incremental deliveries, and a workable channel for rapid customer feedback for revised requirements [3]. The family of agile software development methods, which research has shown to be suited to the small team environment, seems to fit these conditions of the small shop software shop [14, 15].

The paper now turns to the means by which small shops can appropriate innovations, such as Agile methods, given their operating constraints. The small software shop has little time to spend on adopting and adapting a method; i.e. finding the right method, training developers, and creating feedback loops for reflection and adjustments to method use [4]. Thus, we ask how and why these small software shops adopt and adapt innovations such as Agile methods in a manner that is plausible, beneficial, and sustainable to their situation?

3 Adopting and Adapting Agile Methods

Software developers rarely adopt systems and software development methods outright, but rather filter and combine elements from these methods to fit their needs. The criteria for selecting method elements, and the adaptation process itself appears to be complex and dependent on a number of factors, including: 1) the background and experience of the developers; 2) how and with what training a method is introduced to the developers; 3) the organizational context and managerial culture; 4) the application domain, and so forth [17-22]. However, the small shop is usually consumed with existential issues related to product delivery and a steady pipeline of customers will often dictate the terms of adoption and adaptation.

Developers also commonly adapt Agile methods to suit local contexts and conditions [10-12, 23, 24]. Popular agile methods like Scrum and XP recommend regular collaboration with the customer [25, 26]. Research shows, however, that it is very difficult to fulfill this requirement in practice, and many development teams revert to traditional document based communication with the customer or use a "customer proxy" from their own organization who may or may not have access to first-hand knowledge about the customer's needs [10, 23].

Mangalaraj, Mahapatra, and Nerur [12], develop a model to explain organizational acceptance of agile practices. According to the model, individual, team, technological, task, and environmental factors are key influences on organizational acceptance of XP practices. Using the model to analyze two instances of agile practices in the same

organization, the authors show how conditions and constraints can vary across teams in the same organization, leading to different ways to implement agile practices.

Cao et al. [24] use Adaptive Structuration Theory (AST) to analyze "[h]ow agile software development methodologies [are] adapted for use in different contexts?" (p. 333). Similar to [23] they find that the development teams will implement a number of adaptation strategies in order to overcome challenges to agile method adoption. The purpose of the adaptation is to maintain the rationale or "spirit" behind the agile methods; e.g., working software, open communication, iterative progress, honest plans, and team agency; within the constraints set by the developers' capabilities and expertise, as well as the conditions of the development project, such as; e.g. complex architectural requirements, the lack of access to a customer, or organizational and management constraints.

Senapathi and Srinivasan [11] explore how innovation, sociological, technological, and organizational factors influence the breadth and depth of agile usage in an organization and the effectiveness of agile methods for the organization.

Any agile practices may be changed or omitted to fit local constraints and conditions [25, 26, 32, 33]. Such adaptations do not, in and by themselves make the resulting practices non-agile, rather the adaptations are motivated by the development teams' ambition to maintain the "spirit" of agile development within the conditions and constraints they are facing [23, 24]. If, for example, documentation level and details are controlled by regulatory demands as in e.g. finance applications, then developers can fit the agile methods to the documentation requirements by producing "just enough" documentation up-front, and finalizing documentation post-hoc, when the product is reasonably stable, instead of abandoning agile practices altogether [24].

4 Research Method

In order to understand the phenomenon of adoption and adaptation of agile methods in a small-shop setting, the research was conducted with Dialogical Action Research (DAR) [27-29]. DAR presents a researcher/practitioner partnership that allows for a reflective dialog to explore and shape change in an organization and also specify learning for a scientific community. For this research, a small software shop, SSC, entered into a DAR partnership with an aim to adopt a suitable software development method and the researcher's aim was to understand the adoption/adaption process from an empirical perspective. Taking the DAR approach requires that the researcher engage in dialog with the practitioner to identify unexpected issues and react accordingly with suggestions for action-taking to effect change. This emergent and engaged approach for both research and practice is designed to foster greater understanding of practical phenomenon, such as the adoption and adaption of agile methods [30]. DAR, as is the case with agile methods, is intended to be engaged iteratively such that a practitioner's real-world problem can be addressed through dialog, action planning, action taking and assessment [27].

The engagement with SSC took place over a period of nine months from June 2008 to February 2009. In the first phase of the research the following steps were carried

out (1) plausibility; (2) diagnosis; (3) prescription. This resulted in a diagnosis of the main problems experienced by the practitioners in SSC. Understanding SSC's desire for a valid software development method, the researcher suggested and confirmed the suitability of agile methods and XP in particular within the DAR partnership. Also during the DAR partnership, the researcher monitored the progress of two major projects, and myriad smaller projects, to better understand SSC's adopted and adapted XP practices.

The practitioner-researcher partnership at SSC was used as a source of data that primarily consisted of research field notes, interview and meeting transcripts and supplemental documents taken from the practitioners' work environment. Using HyperResearch these texts were coded, using open, axial, and selective coding, in order to derive common themes related to SSC's method adoption, adaptation and use.

5 SSC's Adoption of XP

The DAR partnership at SSC revealed a development process largely consistent with the "waterfall" Systems Development Lifecycle (SDLC). However, the lead developer and owner, Daphne, felt the need to adopt a method that projected greater professionalism to customers. Dialog also revealed Daphne's perspective that she was the primary arbiter of institutional and problem-domain knowledge and that this knowledge was not effectively shared with the rest of the organization. Daphne characterized this as an immense pressure to be involved in almost all activities in the company, including training new employees and overseeing their work. This became less and less feasible as the company started to grow.

Further dialog and diagnosis culminated in the adoption of XP as a systematic framework for development. The researcher saw value in XP for its reflection and learning components. XP was selected to address the need for a shared and systematic approach to development, and to improve knowledge transfer, learning and reflection within the small shop. Thus, XP was selected for two primary reasons: (1) it was the most appealing to SSC's practitioners after a presentation of alternative Agile mehods, and (2) it presented Daphne with an opportunity to work closely with the three other developers and one graphic artist.

Over time, the practices and techniques of XP were introduced gradually into SSC by the researcher. The first to be introduced were Daily Standup Meeting and User Stories. According to Daphne, these activities provided the team with increased focus and increased productivity. Spike Solutions were also useful to SSC as they afforded justification for experimentation and testing. Next, SSC tried pair programming mainly as a means to create Spike Solutions whenever uncertainty was encountered.

Figure 1 shows the major processes that SSC made an attempt to adopt. In Figure 1-3, the elements outlined in solid black are those that SSC were able to fully and successfully adopt. The elements outlined as dashed are those that were partially adopted or adapted. Lastly, any element outlined as dotted are those where were not adopted at all.

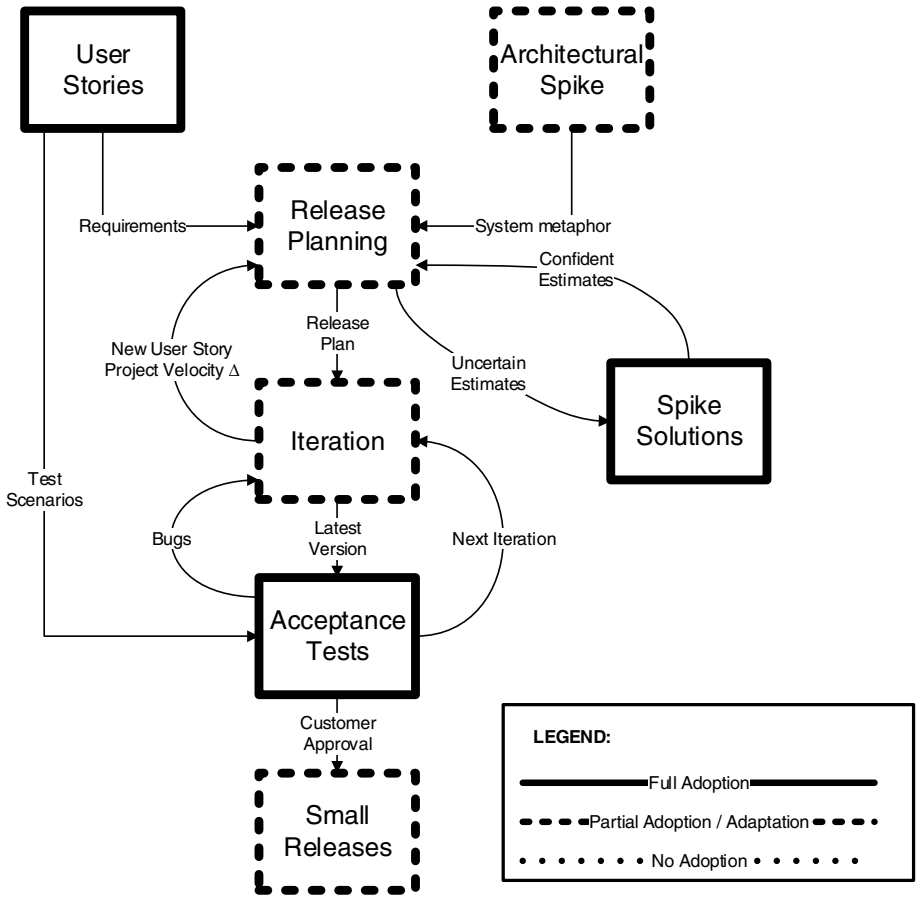

Fig. 1. SSC's Adoption of XP Processes

Initially, SSC focused on adopting the XP processes related to Iteration Planning. Figure 2 shows the Iteration activities SSC were able to fully or partially adapt. As a next step, SSC had progressed further with learning and adopting XP and responded with positive feedback concerning the benefits of adopting XP. Toward the end of the study, SSC had adopted the Collective Code Ownership portions of the XP method. At this stage differences in the nature of SSC's web development projects and other impediments arose which prevented SSC from fully adopting the XP activities within Collective Code Ownership. When the researcher suggested that XP encourages all production code to be written using Pair Programming, SSC responded that it would not be possible to do so given the vast number of projects simultaneously underway.

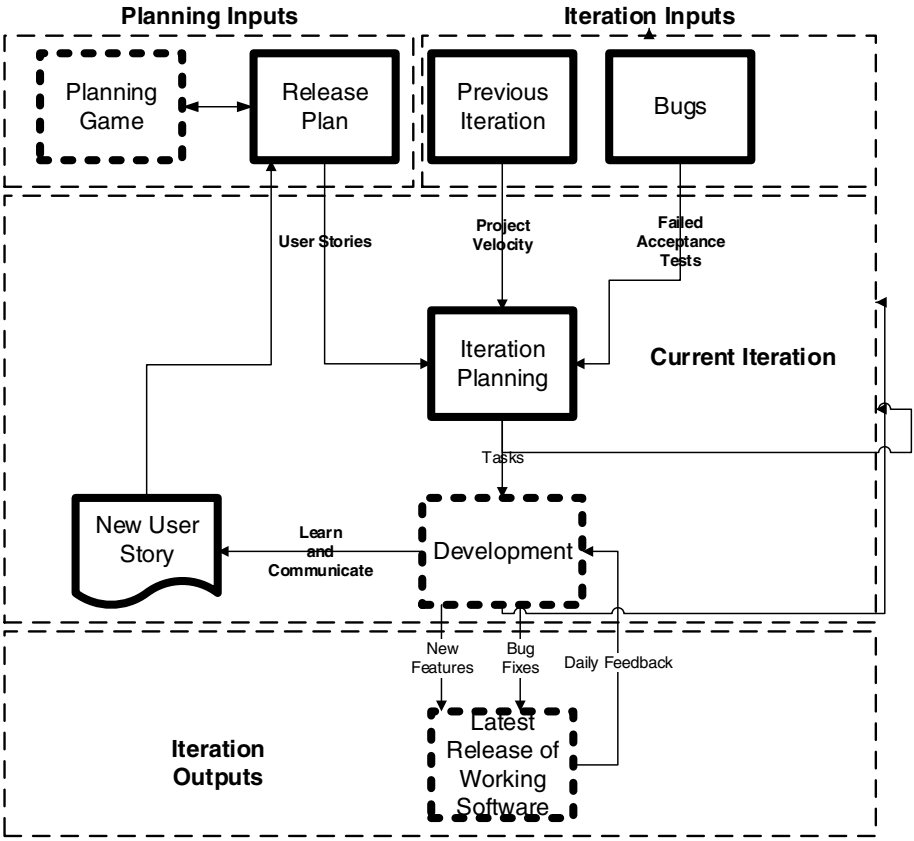

Fig. 2. SSC's Adoption of the Iteration Activities of the XP Method

Similarly, toward the end of the study, the researcher saw no direct evidence that Unit Testing was implemented in the manner suggested by XP. This was despite the fact that the researcher demonstrated to the SSC developers how to do Unit Testing with the NUnit software package. As shown in Figure 3 (dotted line means no adoption), the principle of Move People Around was not adequately considered or attempted save for the times when two developers would "pair" to create a Spike Solution. Lastly, none of the principle of Release Planning activities, such as Acceptance Testing and automated Unit Testing procedures, were witnessed or documented by the researcher.

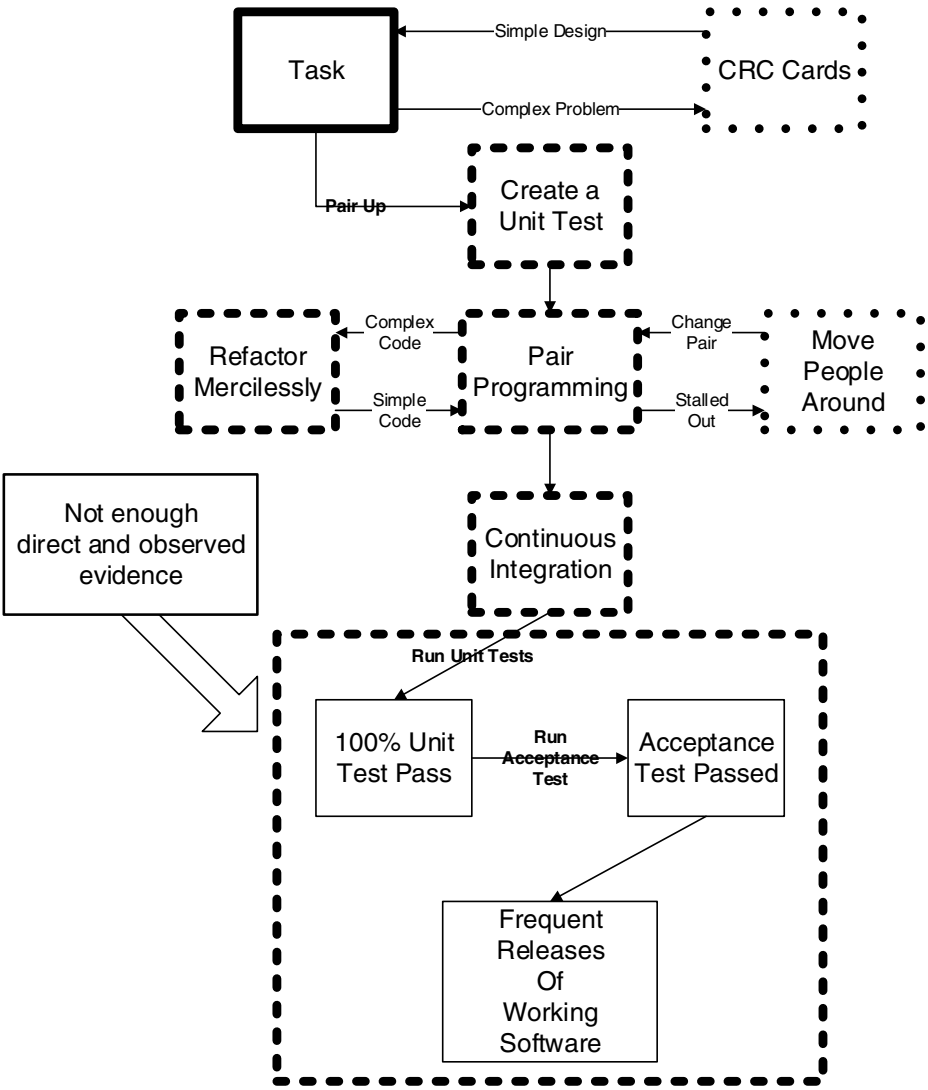

Fig. 3. SSC's Adoption of the Collective Code Ownership Activities of the XP Method

By the end of the study period, SSC had learned and/or tried the majority of XP practices and offered the researcher substantial positive feedback about their successes with using the method and also reported areas where the method was not going to "fit" them. Figure 4 summarizes SSCs adoption and adaptation of XP.

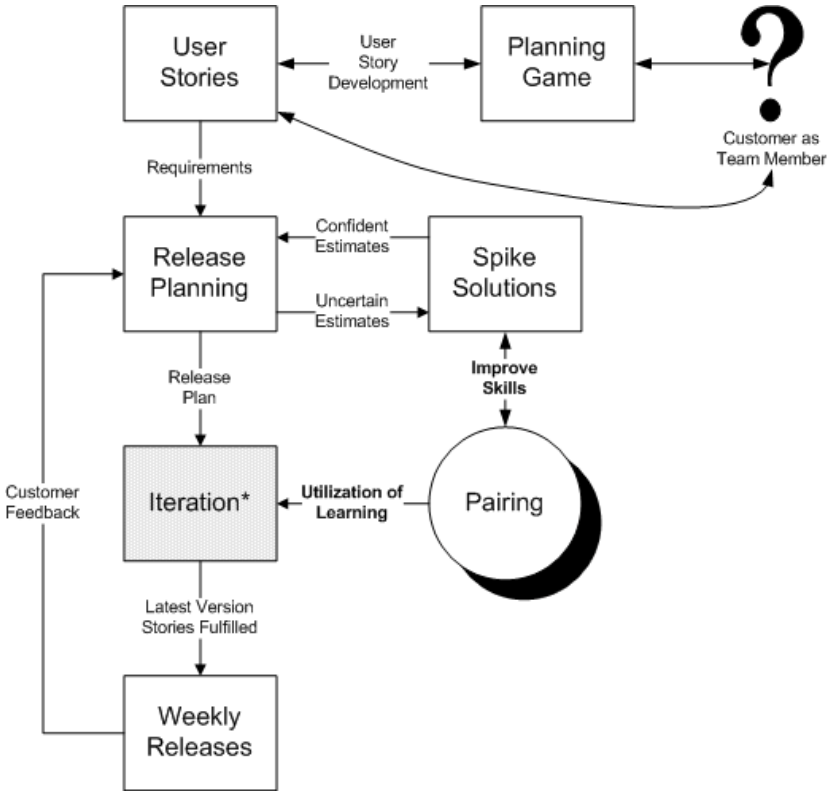

Fig. 4. XP Processes as Adapted by SSC

6 Analysis and Discussion

SSC's adoption and adaption are consistent with suggestions that a team that appropriates XP should "disaggregate" the elements of XP and subsequently critically examine the value of each element [25, 33, 34]. Moreover, local adaptation is also predicted and encouraged as part and parcel of a team taking responsibility for its own process management and evolution [25]. Coleman [22] suggests that small companies will tend to rely on tacit knowledge and less formal means of communication and documentation. In most cases, the burden of the formalizations of method will be difficult to adopt in the small shop setting. It is important to reiterate that both informal learning structures, a natural tendency for Daphne to keep the "reins of control" in tight, and high iteration pressure were primary influences on the adoption and adaptation experiences of SSC.

In Table 1 below, we summarize the outcomes of SSC's adoption and adaption of XP. SSC was particularly interested in the XP practices that were perceived to be immediately useful and/or consistent with their overall desire to adopt a method: the ability to explain their process to customers, and as a vehicle for knowledge sharing and learning. Prior to XP adoption/adaptation, SSC felt as though they were wasting

time by not learning from past projects and that they had generally fallen into a routine that followed a plan-code-test-release cycle without any formalization. Given this low bar, it is possible that any structured approach offered would have been welcomed, agile or not. However, in the case of SSC, the structure and discipline of XP was innovation enough.

Table 1. Summary of SSC's adoption and adaption of XP

XP Practice	Outcome	Implications
User Stories	Adopted	Valued as a means for feature documentation.
Planning Game	Adopted	Used to iteratively refine user stories. Maintained physically on cards.
Architectural Spike	Adapted	System metaphor discussed inconsistently. It is not likely this practice survived.
Release Planning	Adapted	High number of simultaneous projects prevented consistent care in release planning.
0Spike Solutions	Adopted	Perhaps more adopted. Proof of concept spike projects were used for junior developers to prove feasibility to lead developer.
Weekly Release	Adapted	High number of simultaneous projects prevented consistent weekly releases.
Daily Standup	Adopted	Eventually valued for task assignment and coordination. Also used for reflection activities.
Unit Testing	Adapted	Not consistently used but was incorporated in a few projects. It is not likely this practice survived.
Refactor Mercilessly	Adapted	Discussed intermittently during standup. Dedicated tracking and use of tools not observed. It is not likely this practice survived.
Pair Programming	Adapted	Used as a means of training junior developers in “how things are done.” Use for bidirectional learning and reflection not observed.
CRC (Class, Responsibilities, Collaboration) Cards	Omitted	While user story cards were seen as useful, the team thought these to be cumbersome.
Move People Around	Omitted	High number of simultaneous projects and team size made this prohibitive.
Continuous Integration	Adapted	Inconsistently implemented. Many projects would go for weeks without integrating “done” software into the overall project.
Customer as Team Member	Omitted	High number of simultaneous projects and nature of the business made this very difficult/impossible.

Whereas XP values working software, open communication, iterative progress, honest plans, and team agency, a small shop like SSC ultimately valued the quicker cycles and frequent review of projects to establish their progress. Team learning and agency were often less of a focus. During dialogs, which provided a time for reflection for the team and the individuals, aspects related to learning and empowerment felt compelling to the team. However, during normal working routines, iteration pressure was paramount and the reflective aspects of XP were less engaged.

Adaption raises questions regarding the consequences for deviations from "canonical" XP. Particular interest arises as to why SSC adopted, adapted, or omitted XP practices, and what implications this has for SSC's effectiveness, technical growth, learning, and process maintenance/evolution. Our analysis will focus on two important factors influencing SSC's adoption and adaption of XP: Management attitude and commitment and the characteristics of small software shops.

6.1 Management Attitudes and Commitment

XP adoption at SSC was heavily influenced by the experience and attitude of Daphne, the company owner. She saw the Daily Standup Meetings as a convenient way to monitor project progress, but they were also an important vehicle for sharing knowledge and experience. Similarly, and citing her own higher skill and experience level, she saw Spike Solutions and Pair Programming as means to elevate her employees' skills until parity with her own skills was reached. Further, since she fully expected the skills of her employees to eventually surpass her own, she saw Pair Programming as a means of transferring skills when new problems required her expertise. Thus, an important goal of the DAR partnership was met for Daphne: relief from the pressure of her previous monopoly on knowledge and expertise.

However, while Daphne understandably values control, methods such as XP may pose the risk of Daphne losing direct control over creativity and innovation at SSC. Thus, while Daphne did not support Pair programming as a general practice for knowledge development and Shared Code Ownership, she did support it as a means for transferring her skills to the developers. The impact of management attitudes and commitment is also reported in other research into agile adoption; c.f. [11]. It is possible that this perspective would have evolved over time.

6.2 Being a Small Software Shop

SSC context, as a small software shop, was the primary reason for modifications and adaptation of several elements of XP. Among the more challenging aspects of XP to adopt was the concept of Customer as Team Member. SSC had, at any given time, anywhere from 20 to 40 projects underway; each at varying stages of completion or maintenance. However, SSC did see the utility in this approach in the case of larger projects and in use with their strategic partnerships.

The impracticality of including a customer as team member on every project had follow-on effects and implications for User Stories, Acceptance Testing and the Planning Game. Daphne would write User Stories, and later Acceptance Tests, based

on her own notes or her memory of a client's intentions rather than capturing these intentions directly in the user's words (and in their own hand-writing if possible). In short, it was readily apparent that SSC would not be able to follow the orthodoxy of XP in this regard.

The large number of simultaneous projects had effects on Iteration Planning and Release Planning as well. SSC was forced to provide frequent releases of working software on a schedule that is more rigid than XP intends. Furthermore, SSC, did not adopt "test first" or "test driven" Unit Testing. This was despite the fact that the developers did recognize and profess value in the approach – one aptly called it "testing memory." However, it is very probable that SSC will move on towards a "test first" approach over time as they mature with the process.

The SSC developers made significant strides with Pair Programming during the latter period of the study. However, given the very small size of the SSC development team, it was nearly impossible to use Pair Programming to create all production code, as is prescribed by "orthodox" XP practices [25, 32, 33].

XP practices such as Customer as Team Member and Pair Programming are almost always tweaked or adapted one way or the other [36] for a variety of reasons. In the case of SSC, however, we see the adaption of these practices as directly related to the small size of the company.

Other reports of agile development teams relate the adaption (or omission) of the practice of Customer as Team Member to issues in the customer or user organization: the customer organization does not want to commit a person due to resource constraints or a lack of understanding of what it takes to get involved in agile projects or, it is not possible to identify the right representative in the case of; e.g., product development [10, 23]. In the small shop setting, however, the adaption of Customer as Team Member is caused by the large number of individual customers. It would simply not be feasible, given that each developer worked in small increments on different projects for several customers, to have the customer on site.

Regarding Pair Programming, the characteristics of SSC as a small company similarly creates barriers to adopting this practice. Others have related adaption or omission of Pair Programming to lack of interest [24] or to an uneven distribution of knowledge and skills among developers and the need for specialized skills in different parts of the software [12]. In, SSC, however, the practice of pair programming (and other aspects of collective code ownership) was ultimately abandoned because of the nature of the company's projects and customers. Most projects were relatively small and carried out by one developer only. This eroded the need for shared knowledge about a specific project or customer; or, at least, made it very uneconomical to insist on Pair Programming. Note, however, that Daphne would use Pair Programming to ensure other developers were in-sync with her wishes/techniques.

To summarize, the DAR experience affords insight on how certain aspects of XP appear less compatible with the small shop than with teams in a larger organization. While many agile methods may seem "informal" compared to plan-driven methods (such as CMMI), agile methods may present a higher degree of formality than a very small team, such as SSC's developers, can bear. Practices such as Customer on Site

and Pair Programming are quite difficult when there are only 3 to 5 developers. While agile may not be “plan-driven” it is certainly an endeavor that requires discipline. Thus, a small shop may naturally “teeter” on the precipice of agility and devolution into ad hoc cowboy coding [35]. It is clear from their adoption and adaption choices that SSC desires the cohesion of team activities, but the characteristics of SSC make consistent teamwork within the XP method a challenge.

Even with XP as a guide, the process formation (adoption and adaption) at SSC was certain to be shaped and tailored by local circumstances and constraints. Coleman [22] proposes that this method tailoring is a normal aspect of process evolution. SSC’s challenge will be to establish a means for reflective monitoring of this evolution.

7 Analysis and Discussion

In this paper we have presented a detailed account of how a small software business adopted and adapted XP practices in their development. Of interest in this paper are the motivations for adopting and adapting XP. In the case discussed in this paper, method adaptation happened nearly immediately and in result of the DAR partnership. Some aspects of XP (pair programming in particular) were roundly rejected and only begrudgingly incorporated, sporadically, as a means of engaging in reflection and/or training from manager to developer.

SSC's adoption and adaptation of XP shares many characteristics found in other studies of agile practice. We do, however, also observe how specific circumstances and constraints related to the small size of the company affect XP adoption and use. As such, this paper extends previous research on the adoption and adaptation of agile methods into the domain of small software shops. Understanding the motivations for adaption, and whether adaption choices may miss some of the helpful aspects of an agile method, is also a primary interest and contribution of this paper.

The outcomes and analyses of the DAR partnership has implications for both research and practice: We now have more empirical context for which practices are suitable in a small software shop, but further longitudinal inquiry into adoption and adaption in small shops and small teams is required. There are more questions in need of answering. For instance, during adoption and adaption, are developers aware of the "agile values" and principles or are they "just" performing the steps? How much indoctrination of agile principles and values is needed in order to create sustained support for agile development? What is the role of self-organization in method adoption and adaptation in the small shop? This research also questions the meaning of "team" in small shop agile development. The developers in SSC worked on different projects and thus hardly formed a "team" in the traditional sense. Nevertheless they successfully used team-based agile practices, such as daily stand-up, as a coordination and knowledge sharing mechanism. Further empirical study is required to develop answers, and theorizing, according to these questions.

References

1. Babb, J.S.: Towards A Reflective-Agile Learning Model and Method in the Case of Small-Shop Software Development: Evidence From An Action Research Study. Information Systems, PhD, pp. 449. Virginia Commonwealth University, Richmond VA (2009)
2. Hansen, P.A., Serin, G.: The Structure of the ICT Sector in the Øresund region. ÖresundIT (2010)
3. Fayad, M.E., Laitinen, M., Ward, R.P.: Thinking Objectively: Software Engineering in Small Companies. Commun. ACM 43, 115–118 (2000)
4. Pedreira, O., Piattini, M., Luaces, M.R., Brisaboa, N.R.: A Systematic Review of Software Process Tailoring. ACM SIGSOFT Software Engineering Notes 32, 1–6 (2007)
5. Pino, F.J., Garcia, F., Piattini, M.: Software process improvement in small and medium software enterprises: a systematic review. Software Quality Control 16, 237–261 (2008)
6. 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, vol. 1, pp. 23–38. Tapir Akademisk Forlag, Molde (2010)
7. Dybå, T.: Factors of Software Process Improvement Success in Small and Large Organizations: An Empirical Study in the Scandinavian Context. In: ESEC/FSE, pp. 148–157. ACM, Helsinki (2003)
8. Laitinen, M., Fayad, M.E., Ward, R.P.: Guest Editors' Introduction: Software Engineering in Small Companies. IEEE Software 17, 75–77 (2000)
9. Nerur, S., Mahapatra, R., Mangalaraj, G.: Challenges of migrating to agile methodologies. Commun. ACM 48, 72–78 (2005)
10. Hoda, R., Noble, J., Marshall, J.: The impact of inadequate customer collaboration on self-organizing Agile teams. Information and Software Technology 53, 521–534 (2011)
11. Senapathi, M., Srinivasan, A.: Understanding post-adoptive agile usage: An exploratory cross-case analysis. The Journal of Systems and Software 85, 1255–1268 (2012)
12. 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)
13. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology 50, 833–859 (2008)
14. Abrahamsson, P., Conboy, K., Wang, X.: 'Lots done, more to do': the current state of agile systems development research. European Journal of Information Systems 18, 281–284 (2009)
15. Lester, N.G., Wilkie, F.G., McFall, D., Ware, M.P.: Investigating the role of CMMI with expanding company size for small- to medium-sized enterprises. J. Softw. Maint. E 22, 17–31 (2010)
16. Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., Murphy, R.: An exploratory study of why organizations do not adopt CMMI. J. Syst. Softw. 80, 883–895 (2007)
17. 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)
18. 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)
19. Fitzgerald, B., Russo, N.L., Stolterman, E.: Information Systems Development. Methods in Action. McGraw-Hill (2002)

20. 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)
21. Kautz, K., Madsen, S., Nørbjerg, J.: Persistent Problems and Practices in Information Systems Development. ISJ (2007) (accepted for publication)
22. Coleman, G., O'Connor, R.: Investigating software process in practice: A grounded theory perspective. J. Syst. Softw. 81, 772–784 (2008)
23. Hoda, R., Kruchten, P., Noble, J., Marshall, J.: Agility in Context. Object-Oriented Programming, Systmes, Languages and Applications conference (OOPSLA2010). ACM, Reno/Tahoe (2010)
24. Cao, L., Mohan, K., Xu, P., Ramesh, B.: A framework for adapting agile development methodologies. European Journal of Information Systems 18, 332–343 (2009)
25. Beck, K., Andres, C.: Extreme programming explained: embrace change. Addison-Wesley Professional (2004)
26. Schwaber, K., Beedle, M.: Agile software development with Scrum. Prentice Hall Upper Saddle River (2002)
27. Mårtensson, P., Lee, A.S.: Dialogical Action Research at Omega Corporation. MIS Quarterly 28 (2004)
28. Sein, M.K., Henfridsson, O., Purao, S., Rossi, M., Lindgren, R.: Action Design Research. MIS Quarterly 35 (2011)
29. Costello, G.J., Donnellan, B., Conboy, K.: Dialogical Action Research as Engaged Scholarship: An Empirical Study. In: ICIS (Year)
30. Van de Ven, A.H.: Engaged Scholarship: A Guide for Organizational and Social Research: A Guide for Organizational and Social Research. Oxford University Press (2007)
31. Lee, A.S., Mårtensson, P.: Dialogical Action Research at Omega Corporation, Richmond, VA, pp. 1–39 (2004)
32. Jeffries, R., Anderson, A., Hendrickson, C.: Extreme programming installed. Addison-Wesley Professional (2001)
33. McBreen, P.: Software craftsmanship: the new imperative. Addison-Wesley Professional (2002)
34. Glass, R.L.: The state of the practice of software engineering. IEEE Software 20, 20–21 (2003)
35. Wood, W.A., Kleb, W.L.: Exploring XP for scientific research. IEEE Software 20, 30–36 (2003)
36. Arisholm, E., Gallis, H., Dyba, T., Sjoberg, D.I.: Evaluating pair programming with respect to system complexity and programmer expertise. IEEE Transactions on Software Engineering 33, 65–86 (2007)