UNDERSTANDING AGILITY IN SOFTWARE DEVELOPMENT THROUGH A COMPLEX ADAPTIVE SYSTEMS PERSPECTIVE

Journal:	*17th European Conference on Information Systems*
Manuscript ID:	ECIS2009-0561.R1
Submission Type:	Research Paper
Keyword:	Information Systems Development (ISD), Complexity / Complex adaptive systems, Agile computing, Case Study

UNDERSTANDING AGILITY IN SOFTWARE DEVELOPMENT FROM A COMPLEX ADAPTIVE SYSTEMS PERSPECTIVE

Xiaofeng Wang, Lero, the Irish Software Engineering Research Centre, Limerick, Ireland, xiaofeng.wang@ul.ie

Kieran Conboy, National University of Ireland, Galway, Ireland, kieran.conboy@nuigalway.ie

Abstract

Agile software development methods have emerged in recent years and have become increasingly popular since the start of the century. While much research claims to study agile methods, the meaning of agility itself in software development is yet to be fully understood. Agility is viewed by some as the antithesis of plan, structure discipline and bureaucracy. This study aims to develop a better understanding of agility, using the key concepts of Complex Adaptive Systems as a theoretical lens. The study explores agility from several different angles, including autonomous team, stability and uncertainty, and team learning. A multiple case study research method was employed. The findings of the study emphasize that agility is manifested as stability and discipline, which are just as desirable as flexibility, and context sharing is of the same value and importance as knowledge sharing. In addition, the collective nature of learning is underlined.

Keywords: agility, complex adaptive systems, autonomy, stability, team learning

1 INTRODUCTION

The last ten years or so has seen the emergence of agile software development methods as a response to the inefficiency of existing software development methods in rapidly changing environments (Highsmith 2002), e.g. eXtreme Programming (XP) (Beck 1999) and Scrum (Schwaber & Beedle 2002). A brief reflection on the history of the agile software development movement, however, reveals that agile methods originated as a set of techniques and practices, and the term agile is more a post-rationalization to justify a set of existing "light-weight" methods. Agility in software development has been interpreted in many different ways in practice. Skepticism and criticism of agile methods place agility to the opposite of plan, structure and discipline which are generally considered the core components of more traditional waterfall methods (Rakitin 2001, Stephens & Rosenberg 2003).

To clarify the meaning of agility, Conboy and Fitzgerald (2004) conduct a review of the literature on agility across several disciplines including manufacturing, business and management, and carefully distinguish several intertwined concepts, including flexibility and leanness. Based on the comparison and contrast of these concepts, they provide a broad definition of agility as "the continual readiness of an entity to rapidly or inherently, proactively or reactively, embrace change, through high quality, simplistic, economical components and relationships with its environment" (Conboy & Fitzgerald 2004, p.40). Lyytinen and Rose (2006) explore agility in an information systems development (ISD) context. They claim that ISD agility is concerned with why and how ISD organizations sense and respond swiftly as they develop and maintain information system applications. They outline a theory of ISD agility drawing upon a model of Information Technology (IT) innovation and organizational learning which adopts March's (1991) concepts of exploration and exploitation. Their empirical study shows that the concept of ISD agility is more multifaceted and contextual than conceived so far in the literature. It relates to being nimble in terms of the velocity to absorb base innovations and innovate with IS products; the velocity to shift from one innovation regime to another (organizational flexibility); the velocity to learn from experiences (trial and error learning); and the velocity to deliver IS solutions. Each one of these demands different competencies and expects managerial shaping of alternative organizational goals and incentives. Their findings suggest that the dynamics and interactions between these four types of agility form different ecological niches. Each one follows a different organizing logic. Managers must view the meaning of agility differently in each niche.

While these studies help to understand agility, and do highlight the lack of theoretical foundation regarding agility in an ISD context, they do not address specifically how agility is manifested in software development environments. Based on this observation, this study investigates the meaning of agility in software development using the lenses of Complex Adaptive Systems (CAS), an important branch of the complexity study which provides insights of how a system can be adaptive to its environment. (Note that in the following sections the full phrase complex adaptive system is used to refer to an instance of a complex system that demonstrates an adaptive nature, while CAS is used to refer to the study and theory of such systems.) The empirical part of the study employs a multiple-case study approach. The remaining part of the paper is organized as follows. Section 2 introduces the key concepts of CAS and builds a conceptual framework based on CAS which guides the empirical investigation; Section 3 describes the research method and the context of the empirical study; then the findings are presented in Section 4 and discussed in Section 5. The paper ends up with a conclusion section where the implications and limitations of the study are reviewed and the future work summarized.

2 A COMPLEX ADAPTIVE SYSTEMS PERSPECTIVE ON AGILITY

A complex adaptive system, roughly defined, consists of a large number of agents, each of which behaves according to some set of rules. These rules require agents to adjust their behaviour to that of other agents. They interact with, and adapt to, each other. CAS seeks to identify common features of

the dynamics of such systems or networks in general (Stacey 2003). There is no single and definitive account of CAS. Anderson (1999), Mitleton-Kelly (2003) and Stacey (2003) provide valuable introductions to CAS in the context of organization and management. Four key concepts of CAS in the centre of these accounts are of particular relevance to this study: inter-connected autonomous agents, self-organization, the edge of chaos and emergence. These key concepts provide a new perspective to investigate different facets of agility as a desirable property for software development teams in constantly changing environments.

The concepts of inter-connected autonomous agents and self-organization suggest that, to be agile, a software development team should be composed of autonomous members who have their own schemata, which generally refer to norms, values, beliefs, and assumptions that are held by individuals (Senge 1990, Schein 1997). Team members are interconnected in such a way that a decision or action by any individual may affect related individuals and the team. A team composed of autonomous but inter-connected members can spontaneously come together to perform a task (or for some other purpose); the team decides what to do, how and when to do it; and no one outside the group directs those activities (Mitleton-Kelly 2003). To do so, a team needs energy imported into and constantly flowing within it, which can be interpreted, partly, as the sharing of information, knowledge or other resources needed to sustain self-organized activities.

The edge of chaos provides organizations "with sufficient stimulation and freedom to experiment and adapt but also with sufficient frameworks and structure to ensure they avoid complete disorderly disintegration" (McMillan 2004, p. 22). Brown and Eisenhardt (1998) contend that, to compete at the edge, organizations must understand what to structure and what not to structure, to foster communication and to capture cross-business synergies. The edge of chaos concept suggests that being agile is neither chaotic nor static. It needs stability but not so much that order prevails and innovation is stifled. It is a delicate balance of both.

The concept of emergence sheds new light on learning, which can be seen as a collective behavior of creating new patterns of thought at the team level based on the interaction of individuals, instead of often seen exclusively as the provision of individual training. Learning means not only training or the acquisition of new skills, but also the gaining of insight and understanding which leads to new knowledge and behavior. When learning leads to new behavior, the team can be said to have adapted and evolved (Mitleton-Kelly 2003). An agile team facilitates team learning and generation of new knowledge. In addition, new knowledge needs to be shared to generate further new learning, knowledge and behavior.

In summary, this study investigates the meaning of agility from three facets: autonomous but sharing team, stability with embraced uncertainty and team learning, as shown in Table 1.

Facets of Agility	Underlying CAS Concepts	Relevant Studies
Autonomous but sharing team	Inter-connected autonomous agents Self-organization	Anderson 1999; Choi et al. 2001 ; Mitleton-Kelly 2003
Stability with embraced uncertainty	The edge of chaos	Brown and Eisenhardt 1998; Stacey 2003
Team learning	Emergence	Mitleton-Kelly 2003; Stacey 2003

Table 1. Agility through the CAS perspective

3 RESEARCH APPROACH

This study adopts an interpretivist stance, emphasizing that agility are situational and can be better understood through the understanding and sense making of people who are involved in software development. In particular, this study employs a qualitative approach, treating agility as a qualitative

property of a software development team that can be better studied through words and the meanings people ascribe to them rather than numbers or frequencies. The specific research method used in this study is case study, which is an appropriate approach when a research phenomenon is investigated in its real-live context (Yin 2003). A multiple-case design is employed. Given the research focus of the study, the level of inquiry is at the team level, so it seems appropriate to take a software development team as a case. The unit of analysis is the software development team. Three software development teams - XPTeam A, XPTeam B and WaterfallTeam - from two different companies were chosen as the cases. XPTeam A is a representative case; XPTeam B is a confirming case of the first one; and WaterfallTeam is a contrasting case, following the strategy suggested by Yin (2003). The profiles of the three cases are shown in Table 2. XPTeam A is a software development team in SecureSoft, a small software house specialized in network security and management systems development. XPTeam B and WaterfallTeam are software development teams in WorldTech, a major IT company providing both IT projects and services.

	XPTeam A	XPTeam B	WaterfallTeam
Team size	4	8	5
Team composition	3 developers, 1 project manager	6 developers, 1 test manager, 1 project manager	4 developers, 1 project manager
Development method	XP	XP	Waterfall style mixed with some agile elements
Years of method use	4.5 - 5 years	11 months to 1.5 years	More than 5 years
Location	Co-located in an open office space	Co-located in an semi-open office space	Collocated in an semi-open office space
Software developed	Application for external customer	Web application for internal use	Backend application for internal use

Table 2. The profiles of the three cases

Two rounds of data collection are conducted. The interval between the two rounds is six months. The main data collection method used is semi-structured face-to-face interviews. The questions are all open-ended. The members of each team are interviewed. Each interview lasts between 30 minutes to two hours. In all the cases, most interviewees are interviewed twice. Table 3 lists the people interviewed in each team. Documents regarding the development processes of the case teams are collected when available. Some non-participative observations are conducted as the opportunities occur. Field notes are taken during both rounds of data collection.

	XPTeam A	XPTeam B	WaterfallTeam
First round interviews	1 group interview (with the 4 team members below), 4 individual interviews - Project manager - Coach - Developer A - Developer B	5 individual interviews - Project manager - Team lead - Tech lead - Developer A - Test manager	1 individual interview - Project manager
Second round interviews	2 group interviews (with the team members below), 3 individual interviews - Coach - Developer A - Developer B	6 individual interviews - Project manager - Team lead - Tech lead - Developer B - Developer C - Test manager	3 individual interviews - Project manager - Developer A - Developer B

Table 3. Two rounds of interviews

The data analysis includes two steps: within-case analysis and cross-case comparison (Eisenhardt 1989). The emphasis is on the cross-case comparison, in which an analysis tactic suggested by

Eisenhardt (1989) is used: the three cases were divided into two groups, XPTeam A and XPTeam B in one group as the cases using agile approach, while WaterfallTeam in the other group as the case that uses waterfall approach. XPTeam A and B are compared firstly for similarities and differences, and then they as a group are contrasted with WaterfallTeam for similarities and differences.

4 MANIFESTATION OF AGILITY IN THE THREE TEAMS

This section presents how agility has been manifested (or shown to be absent) in the three cases.

4.1 Autonomous but sharing team

Team autonomy in XPTeam A and B firstly is shown as competences relevant to software development being distributed among team members. The members of the two teams are involved in all development activities of their projects, and all have to deal with the customers, analyse user requirements and write code together. There are no traditional roles such as system analyst, designer or programmer. Each team member is able to assume all the roles, since comprehensive competences are required to work with user stories, the implementation of which is self-contained and encapsulates different development activities:

> *"The problem is not to have three persons for analysis, or two persons for design, but a user story inside has to resolve analysis, developing, and, etc., everything." (Project manager/XPTeam A)*

For example, when XPTeam B started the project, there were big gaps among team members in terms of Java related knowledge and skills. With the project going on, the developers with less Java experience learnt quickly from those more experienced, and the team members reached fairly the same level of competence. As a result, there is no dependency on a particular individual, since each team member gets exposure to different areas of a project. Distributed competence is shown in the case of WaterfallTeam too, although the team uses waterfall approach. Like the other two teams, there are no specific roles like analyst, designer or coder in the team. The developers are not specialized on specific tasks. Everybody has chances to do different things.

Team autonomy is also manifested as a disciplined team in XPTeam A and B, which is seemingly contradictory to the idea of autonomy. However, both teams reckon the importance of disciplines. As a member of XPTeam B describes, disciplines are necessary components of an agile process, and they come from the process the team uses:

> *"There is a set of rules really, and you may not adopt them, you probably adopt most of them, and those rules kind of direct you really, it's like you need to formalize it so you can be more flexible." (Test manager/XPTeam B)*

Team autonomy does not mean the team members are working on their own; instead, there is constant sharing among them. XPTeam A considers sharing an important aspect of team working. They believe that, as a team, they have to face every moment in any case without barriers. Sharing is also seen as a contributor to a team's agility by WaterfallTeam who works with the waterfall approach. The difference is that sharing in the two teams using agile processes goes beyond simply knowledge sharing. It extends to context sharing and the sharing of achieved results. What is shared among the team members is not only the technical knowledge related to different areas of a project, which helps to distribute competences among them. It is also the knowledge about who knows what, which is particularly important for a bigger team like XPTeam B, and helps the team members self-organize to implement tasks:

> *"I think the ten o'clock stand-up meeting is definitely good, because you know what everybody else on the project is working on, and you might say 'I'm working on this and I'm not sure how to'… and someone says 'oh yeah actually I did it yesterday'." (Developer B/XPTeam B)*

In XPTeam A and B, the developers are attentive to what happens around them, with the help of the open space the teams are working in:

> *"When you are doing something, you have to listen what the pair, or the single one if you are in pair, what he's doing, what they are saying, you have one ear in this way and the other (in the other way)." (Developer A/XPTeam A)*

Collective ownership of results is another kind of sharing. The two teams using the agile processes both endorse the collective ownership of code, as suggested by XP. A developer of XPTeam B, however, warns that collective ownership can become collective irresponsibility sometimes, which means no one claims to be responsible if there is some problem with a piece of code. In the case of XPTeam A, in addition to collective ownership of code, the team also owns collectively other forms of working results, such as designs, solutions, etc., which helps the team to have a sense of common achievement.

4.2 Stability with embraced uncertainty

Stability for software development is a desired property by all three teams, which is seen as an indispensable component in responding to change:

> *"There has to be some limitation of what you are doing, you cannot be so flexible that things are chopping and changing every single day." (Test manager/XPTeam B)*

Stability first of all is demonstrated as a short-term certainty in all the three teams. The short-term certainty means a team has a very clear idea of what they have to do in a short time frame, such as one-week or two-week iterations in the cases of XPTeam A and B. WaterfallTeam also realizes the importance of the short-term certainty to deal with constant changes from the management:

> *"Well I guess in terms of uncertainty, you don't know really tomorrow you are going to work on the same project, so on a phased approach you can complete one phase and then this is done. And say after tomorrow, let's say the next phase is cancelled because of the management decision, then you still have a product that works." (Developer A/WaterfallTeam)*

In the cases of XPTeam A and B, stability is also shown as a sense of frequent achievement and satisfaction. The two XP teams realize that, with their agile process, the team members can be motivated more easily than with the waterfall method, since the developers can see the result of their work at the end of each iteration, rather than working for six months without anyone has ever seen or used the code produced as what can happen in traditional processes. There is evidence to suggest that the developers of WaterfallTeam also recognize the importance of motivating people, and believe that a satisfied and motivated team is a source preventing a project from falling apart:

> *"If someone is not happy with what he's doing, he's not going to do his job well. If he doesn't like it, he doesn't like to co-operate, if he's not happy with people, he wasn't going too far… So the main thing is with people, keep them happy… because if people are unhappy, the project falls apart." (Developer A/WaterfallTeam)*

In addition, a team focused on working is also a sign of stability. A focused team has several meanings in the two XP teams: one meaning is to focus on work in a short but appropriate amount of time. It can be an iteration, as in XPTeam A and B. Another meaning of being focused is to focus on current work, not wasting time to do future-proof work, which has been emphasized particularly in XPTeam B. The third meaning is to focus on development activities and not to mix them with personal desires of learning new things. For example, XPTeam A is very attentive of keeping the team focused on development activities by reserving daily studying time to satisfy the developers' desires to learn.

Last but not least, stability shows as team working at a sustainable pace, with ease and without anxiety, is another aspect of the stability for development. A developer of XPTeam A associates this working state with agility directly:

"I think agility is a state of mind… you don't have to feel anxiety, you have to be relaxed when you approach a problem, and XP or Scrum is just a method to obtain this kind of relaxity… If you are happy on what you are doing, if you are not stressed, I think you can say you are agile." (Developer B/XPTeam A)

Stability co-exists with uncertainty which is unavoidable in the teams using agile methods. Uncertainty needs to be embraced. Embraced uncertainty is manifested firstly as the probability to change directions in the cases. All the teams believe that the iterative nature of their processes gives them more possibility to change directions when needed, including WaterfallTeam, since they use iterative phases within the waterfall process. But the probability to change should be complemented by having a whole picture of the project, which has been emphasized in the two XP teams. XPTeam B observes that having a whole picture of the project occurs not only to the developers, but also to their onsite customer.

4.3 Team learning

XPTeam A understands that learning means doing things differently. If a team wants to be adaptive and evolve, they have to learn. In the two XP teams, learning happens as team learning rather than individual learning, which means a team as a whole acquires new knowledge and competences, and the results of learning are shared among team members. Compared with WaterfallTeam, team learning happens continuously and mutually, through using agile practices in the two XP teams. It happens in daily development activities. It is a continuous experience for the team members. Meantime, since learning happens through interactions among the developers, it is generally bi-directional. A developer of XPTeam B comments:

"I think it (XP) is a very good way of learning as well, because with pair programming which is part of it, you are learning from somebody different every day, and likewise you're able to teach somebody else for you've been doing the day before … it gives a sense of shared, the project is shared… There's more, definitely more knowledge been shared." (Developer C/XPTeam B)

Besides, learning is not a daunting experience due to the fact that the teams using the agile processes generally work on small pieces of tasks. The developers learn gradually through implementing them, sometimes with the help of others. The team lead of XPTeam B observes that:

"Because it is down to granular level, it's easier to put better workload over people and also easier for people to get involved, it's also easier for people who don't have skill learn gradually on the smaller story rather than having to develop something big on their own, so I think it's easier to get a higher level skill without being overly complicated… They are not huge chunk of piece to take on." (Team lead/XPTeam B)

Due to these attributes, team learning is seen more efficient than individual learning:

"The learning, when we do pair programming it's more efficient. In one year I learn a lot of things that I didn't think (I could do) when I was in the university." (Developer B/XPTeam A)

Table 4 summaries the findings.

5 DISCUSSION

As shown in this study, agility in the context of software development is highly multifaceted and ambiguous. In this section the different facets of agility demonstrated in the cases are discussed by drawing on relevant agile literature.

5.1 An autonomous but sharing team

Despite the suggestion by advocates of agile that software development processes should be organized to improve and distribute both technical and social competences continuously (Cockburn & Highsmith

2001), few empirical studies in agile research have supported this stance. Only Auvinen et al. (2006) highlight an increased competency in a team where several agile practices are piloted. Similarly, no empirical research in the reviewed literature focused on discipline in agile processes despite the emphasis many agilists place on its importance (e.g. Beck & Boehm 2003).

Agility through CAS	**Manifested in software development**
Autonomous but sharing team	Distributed competences
	Disciplined team
	Knowledge sharing
	Context sharing
	Collective ownership of results
Stability with embraced uncertainty	Short-term certainty
	Team being satisfied, motivated and focused
	Working at a sustainable pace
	Probability to change directions
	Having a whole picture of the project
Team learning	Learning continuously
	Mutual learning
	Learning gradually

Table 4. Manifestation of agility in software development

This study suggests that a team composed of autonomous but interacting developers has a tendency to be agile. Each of them is able to solve various development issues and to interact with customers. Competences are not concentrated on few people so that there is no bottleneck in the development process. Team members are confident and courageous in the interactions with customers and with each other. They are also mature and willing to try new things. An autonomous team, however, does not mean team members can be completely amethodical and ill-disciplined. On the opposite, it is composed of disciplined, self-responsible and committed individuals. Discipline is an essential component of an autonomous team, and is drawn from the interactions among peer team members.

Sharing is a common theme investigated in several agile studies, though most are focused on knowledge sharing (Fredrick 2003, Melnik & Maurer 2004, Poole & Huisman 2001, Schatz & Abdelshafi 2005). Context sharing has also been observed, but is somewhat understated in agile literature. Melnik and Maurer (2004) believe that the so-called “background knowledge” about a project is important to achieve effective communication. It is important for all team members to have a common frame of reference - a common basis of understanding. Poole and Huisman (2001) observe that, in the organisation they studied, there was a measurable increase in the visibility of what everyone was doing on the team subsequent to the adoption of the agile practices. In fact, this improvement in visibility is considered one of the greatest successes the company has achieved. In terms of results sharing, Fredrick (2003) reports the experience of collective ownership of codes. When it is realized, even the most complex business problems can be easily figured out. In contrast, it was found that individual ownership of code made people defensive - people took it personally when someone suggested their code did not work. Schatz and Abdelshafi (2005) also document the collective ownership in their experience report where developers took ownership of the features they created and took pride in showing their work to the stakeholders during sprint reviews. Rising and Janoff (2000) notice that in a team they have studied, at every meeting, as small tasks were completed and the team could see progress toward the goal, everyone was more satisfied with their work and project progress.

The findings of this study confirm that sharing in an agile team not only means knowledge sharing. Context sharing is equally important. To effectively self-manage, a team needs to share the understanding of their working context. Context sharing is a precondition to provide effective feedback, interpret them in a sensible way, and take appropriate actions. Sharing also means results

sharing, such as collective ownership of code and solutions, which reduces the risk of knowledge loss and increases the sense of being a true team.

Another type of sharing, namely problems sharing, is reported by Rising and Janoff (2000) but does not emerge in this study. In the team they have studied, when one team member raises an obstacle in the Scrum meeting, the entire team's resources come together to bear on that problem, and the entire team immediately owns any one individual's problems.

5.2 Stability with embraced uncertainty

Several agile studies have noticed team satisfaction and motivation in agile processes (e.g. Rising & Janoff 2000, Poole & Huisman 2001, Drobka et al. 2004). For example, Drobka et al. (2004) conduct a survey of a team using XP and find that it creates a surge in morale since XP provides constant feedback to the developers and at the end of each day the team has a working product. Team members gain a sense of accomplishment from their daily work, because they immediately see the positive impact their efforts have on the project. When morale is high, people are excited about their work, leading to a more effective, efficient development team. Short-term certainty has also been noticed in agile studies, though not so extensively. Murru et al. (2003) claim that XP enhances programmers' sense of project control. They find that programmers with the experience of Rational Unified Process (RUP) felt that XP's planning game gave them a stronger feeling of control than traditional planning did. They knew where their project was going and whether it was delayed. Furthermore, programmers were more aware of keeping the project's strategic goals in focus. This knowledge improved the programmers' motivation.

The role played by uncertainty is acknowledged by agile advocates (Highsmith & Cockburn 2001, Williams and Cockburn 2003). Williams and Cockburn (2003) believe that uncertainty is inevitable in all software development. Many changes occur during the time that the team is developing the product. It is highly unlikely that any set of predefined steps will lead to a desirable, predictable outcome. It is necessitated short "inspect-and-adapt" cycles and frequent, short feedback loops. Agile software development is about change and feedback. Highsmith and Cockburn (2001) claim that agile organizations and managers understand that to demand certainty in the face of uncertainty is dysfunctional, and agile practices encourage change rather than discourage it. In turbulent business situations, the change tolerance of a development process must be geared to the change rate of a specific environment, not some internal view of how much change is acceptable. Despite these claims of agile proponents, however, few empirical studies of agile processes have focused on uncertainty and how it is embraced, with the exception of Elssamadisy and Schalliol (2002) who suggest that, when using the XP practices, especially the simple design, one should look ahead and do things incrementally, in order to have a big picture.

This study emphasizes stability as a desired property of development teams that have to deal with continuous changes due to close relationships with customers and evolving requirements. A team needs stability, needs to find a proper heartbeat for their development process so that it would not be dissolved into turbulence. Stability gives developers a sense of security and control over what they are working on. It can be drawn from a short-term certainty provided by a time-boxed development process. Stability for development also means a team is working at a sustainable pace, focused and motivated, working with ease and satisfaction. Certainty and security is only for a short term, however. Uncertainty is inevitable in software development. It comes from both the environment a team is embedded in and the development process itself. Managing uncertainty does not mean to predict what is going to happen and do future proof work today. It is to ensure the probability to change the direction a team goes towards but meantime not to get short-sighted. Team members need to have a whole picture of the project in mind.

5.3 Team learning

Learning is a common theme explored in much agile research (e.g. Dingsoyr & Hanssen 2002, Drobka et al. 2004, Hunt & Thomas 2003, Meso & Jain 2006), but the focus is mainly on individual rather than team learning. In a survey conducted by Drobka et al. (2004), it was found that XP can reduce the learning curve for new team members. Fifty-five percent of the developers believed that using XP shortened their initial project-learning curve. Hunt and Thomas (2003) emphasize that learning in an agile process is a continuous process, and it means learning about more than just the technology involved. It covers how the team works together (or how it doesn't) and team members themselves, which leads to behavior and mental model change.

Learning means doing things differently. One important consequence of learning for an individual and a team is to change either their behavior or mental model. It is a prerequisite for organizational evolution and co-evolution (Mitleton-Kelly 2003). This study suggests that team learning is different than individual learning, though closely related and dependent on it. From the CAS perspective, team learning is emergent from the interactions of team members. Team learning is a collective result, which means a team as a whole acquires new knowledge and competences, and results of individual learning are shared among team members. In an agile team, team learning happens constantly, mutually and gradually.

6 CONCLUSION

This study investigates how agility is manifested in agile software development through studying the software development processes of three teams, two using XP, one using waterfall approach. Taking the key concepts of CAS as theoretical lenses, the study explores the true meaning of being agile from several different angles, including autonomous team, stability and uncertainty, and team learning. Compared with the existing agile literature, the findings emphasize that stability and discipline are as desirable as flexibility, and context sharing is of the same value and importance as knowledge sharing in agile processes. In addition, the collective nature of learning is underlined. The main theoretical contribution of the study is the understanding of agility in software development which is both theory-informed and empirically grounded. Drawn on CAS, the study lifts the understanding beyond the advocational literature found in agile field (Baskerville & Pries-Heje 2004). The discovered agile properties enrich the understanding of agility in software development. The practical implication of the study is that the findings indicate the desired effects of using agile methods. The agile properties provide a software development team with observable agile indicators from different facets of software development.

One limitation of the study comes from using CAS as the theoretical basis of the study. CAS has originated in natural sciences. There is a deeper concern whether CAS is appropriate to the study of human organizations. A combination of CAS theory with appropriate social theories might be a promising avenue for future research. Some limitations are associated with the case study research approach. One concern is the uniqueness of corporate, team and project characteristics of each case which makes valid comparison and theoretical generalization of the case study results difficult (Kitchenham et al. 2002). Specific to this study, an affecting factor is the diversity of the team profiles (as shown in Table 2). To increase the validity of the study, the contextual information of the cases has been taken into account in the data analysis. Another limitation is that only one agile method, XP, has been involved in our case studies. Future work would be to verify if the findings apply to teams using other agile methods, such as Scrum, Lean system development, etc.

References

Anderson, P. (1999). Complexity Theory and Organization Science. Organization Science, 10(3), 216-232.

Auvinen, J., R. Back, J. Heidenberg, P. Hirkman and L. Milovanov (2006). Software process improvement with agile practices in a large telecom company. In Proceedings of Product-Focused Software Process Improvement, Springer-Verlag, Berlin, LNCS 4034, 79-93.

Baskerville, R. and J. Pries-Heje (2004). Short cycle time systems development. Information Systems Journal, 14(3), 237-264.

Beck, K. and B. Boehm (2003). Agility through discipline: a debate. Computer, 36(6), 44-46.

Beck, K. (1999). Extreme Programming Explained. Addison Wesley, Reading, MA.

Brown, S. and K. Eisenhardt (1998). Competing on the Edge: Strategy as Structured Chaos. Harvard Business School Press, Boston.

Choi, T. Y., K. J. Dooley and M. Rungtusanatham (2001). Supply Networks and Complex Adaptive Systems: Control versus Emergence. Journal of Operations Management, 19(3), 351-366.

Cockburn, A. and J. Highsmith (2001). Agile Software Development: The People Factor. Computer, 34(11), 131-133.

Conboy, K. and B. Fitzgerald (2004). Toward a Conceptual Framework of Agile Methods. In Proceedings of Extreme Programming and Agile Methods - XP/ Agile Universe 2004, Springer-Verlag, Berlin.

Dingsoyr, T. and G. K. Hanssen (2002). Extending agile methods: Postmortem reviews as extended feedback. Advances in Learning Software Organizations, Springer-Verlag, Berlin. LNCS 2640, 4-12.

Drobka, J., D. Noftz and R. Raghu (2004). Piloting XP on Four Mission-Critical Projects. IEEE Software, 21(6), 70-75.

Eisenhardt, K. M. (1989). Building Theories from Case Study Research. Academy of Management Review, 14(4), 532-550.

Elssamadisy, A. and G. Schalliol (2002). Recognizing and Responding to “Bad Smells” in Extreme Programming. In Proceedings of the 24th International Conference On Software Engineering. Association Computing Machinery, New York, 617-622.

Fredrick, C. (2003). Extreme Programming: Growing a Team Horizontally, Extreme Programming and Agile Methods - XP/Agile Universe 2003. Springer-Verlag, Berlin. LNCS 2753, 9-17.

Highsmith, J. and A. Cockburn (2001). Agile Software Development: the Business of Innovation. IEEE Computer, 34(9), 120-122.

Highsmith, J. (2002). Agile Software Development Ecosystems. Addison-Wesley, Boston.

Hunt, A. and D. Thomas (2003). Preparing the Raw Material. IEEE Software, 20(5), 97-98.

Kitchenham, B., S. L. Pfleeger, L. Pickard, P. Jones, D. C. Hoaglin, K. E. Emam and J. Rosenberg (2002). Preliminary Guidelines for Empirical Research in Software Engineering. IEEE Transaction on Software Engineering, 28(8), 721-734.

Lyytinen, K. and G. M. Rose (2006). Information System Development Agility as Organizational Learning. European Journal of Information Systems, 15(2), 183-199.

March, J. G. (1991). Exploration and Exploitation in Organizational Learning. Organization Science, 2(1), 71-87.

McMillan, E. (2004). Complexity, Organizations and Change. Routledge, Taylor & Francis Group, London.

Melnik, G. and F. Maurer (2004). Direct Verbal Communication as a Catalyst of Agile Knowledge Sharing. In Proceedings of the Agile Development Conference. IEEE Computer Soc, Los Alamitos, 21-31.

Meso, P. and R. Jain (2006). Agile Software Development: Adaptive Systems Principles and Best Practices. Information Systems Management, 23(3), 19-30.

Miles, M. B. and A. M. Huberman (1994). Qualitative Data Analysis: an Expanded Sourcebook. Sage, Thousand Oaks, California.

Mitleton-Kelly, E. (2003). Ten Principles of Complexity & Enabling Infrastructures. In E. Mitleton-Kelly (Ed.) Complex Systems & Evolutionary Perspectives of Organisations: The Application of Complexity Theory to Organisations. Elsevier, Pergamon.

Murru, O., R. Deias and G. Mugheddu (2003). Assessing XP at a European Internet Company. IEEE Software, 20(3), 37-43.

Poole, C. and J. Huisman (2001). Using Extreme Programming in a Maintenance Environment. IEEE Software, 18(6), 42-50.
Rakitin, S. (2001). Manifesto Elicits Cynicism. IEEE Computer, 34(12), p. 4.
Rising, L. and N. S. Janoff (2000). The Scrum Software Development Process for Small Teams. IEEE Software, 17(4), 26-32.
Schatz, B. and I. Abdelshafi (2005). Primavera Gets Agile: a Successful Transition to Agile Development. IEEE Software, 22(3), 36-41.
Schein, E. (1997). Organizational Culture and Leadership. Jossey-Bass, San Francisco.
Schwaber, K and A. Beedle (2002). Agile Software Development with SCRUM. Prentice-Hall, Upper Saddle River, NJ.
Senge, P. M. (1990). The Leader's New Work: Building Learning Organizations. Sloan Management Review, 32(1), 7-23.
Stacey, R. D. (2003). Strategic Management and Organisational Dynamics: The Challenge of Complexity. Fourth Edition. Financial Times, Prentice Hall.
Stephens, M. and D. Rosenberg (2003). Extreme Programming Refactored: The Case Against XP. Apress, New York.
Walsham, G. (1995). Interpretive Case Studies in IS Research: Nature and Method. European Journal of Information Systems, 4(2), 74-81.
Williams, L. and A. Cockburn (2003). Agile Software Development: it's About Feedback and Change. IEEE Computer, 36(6), 83-85.
Yin, R. K. (2003). Case Study Research: Design and Methods. Sage, Thousand Oaks, California.