2006

Organizing for agility: A complex adaptive systems perspective on agile software development process

Richard Vidgen
University of Bath, mnsrtv@management.bath.ac.uk

Xiaofeng Wang
mnpxw@bath.ac.uk

ORGANIZING FOR AGILITY: A COMPLEX ADAPTIVE SYSTEMS PERSPECTIVE ON AGILE SOFTWARE DEVELOPMENT PROCESS

Richard Vidgen, University of Bath, Bath, BA2 7AY, mnsrtv@management.bath.ac.uk

Xiaofeng Wang, University of Bath, Bath, BA2 7AY, mnpxw@ bath.ac.uk

Abstract

Agile software development has caught the attention of both practitioners and academics in recent years. In spite of many anecdotes and papers describing lessons learnt the theoretical foundation of agile software development has not been systematically articulated. This paper proposes a conceptual framework to study agile software development based on the theory of complex adaptive systems. The framework is illustrated by a case study of an agile software development team. Several agile practices are identified and reflected on from the perspective of complex adaptive systems.

Keywords: Agile Software Development, Complex Adaptive Systems, Software Development Process.

1 INTRODUCTION

Agile software development (ASD) emerged as a response to the increasingly turbulent business environment in which modern software development projects are enacted. The movement was formalized when leading agile proponents met in November 2001 to produce a "Manifesto for Agile Software Development" (see http://agilemanifesto.org), which outlined four values and twelve principles that were believed to underlie agile methods. A set of agile methods have emerged from practice, including Scrum and eXtreme Programming (XP) (described in Highsmith 2002), and although they differ in the detail and the specific techniques, these methods have much in common, including short iterative life cycles, quick and frequent communication with customers, and constant learning. However, agile methods have been criticised for a lack of grounding in theory (Kalermo & Rissanen 2002, Conboy & Fitzgerald 2004). Further, the underlying assumptions have been questioned (Turk et al. 2002) and there has also been criticism of agile methods by practitioners (Stephens & Rosenberg 2003).

Highsmith (2000) recognizes the need for theory, "techniques without a theoretical base are reduced to a series of steps executed by rote."(p.14), and goes on to argue for complex adaptive systems (CAS) theory as the basis for agile methods (Highsmith 2002). Highsmith and CockBurn (2001) argue that one aspect of agile development often missed or glossed over is a worldview that organizations are complex adaptive systems in which decentralized, independent individuals interact in self-organizing ways, guided by a set of simple, generative rules, to create innovative, emergent results. Kent Beck, a co-inventor of XP, claims that CAS is "the only way to make sense of the world" (Highsmith 2002, p. 48). However, the claim that agile methods are grounded in CAS theory is a post-rationalization, a theory used in a loose way to justify what is done in practice. This paper uses CAS theory to build a framework for organizing the software development process to achieve agility. The resulting framework is then used to analyze a case study of agile software development (the Pongo team) and reflections are made on the fit between the CAS framework and agile practices.

2 AGILE SOFTWARE DEVELOPMENT

A large amount of anecdotal evidence has been published on the use of agile methods in practice. Generally, these studies take the form of descriptive articles, reports, and lessons learned. They argue for the effectiveness of agile process and practices in specific cases (Abrahamsson & Koskela 2004, Murru & Deias & Mugheddu 2003, Rasmusson 2003, Schuh 2001), or the extension of agile methods to cover areas where agile methods are often considered unsuitable, such as large scale projects (Crocker 2004). These descriptive studies of agile methods are valuable because they help us understand what is going on in practice. Unfortunately, the studies on the theoretical aspects of agile software development are much less in number.

Although the Agile Manifesto can be seen as the effort of agile proponents to theorize agile methods, the underlying assumptions of the Manifesto have been questioned by Turk et al. (2002) who find that these underlying assumptions hold neither for all software development environments nor for all "agile" processes in particular. Conboy and Fitzgerald (2004) argue that the Manifesto lacks grounding in theory and Kalermo and Rissanen (2002) say that the Manifesto is grounded in practice and has not been elevated to systematic thinking. Kalermo and Rissanen (2002) undertake a detailed analysis of the Manifesto and re-organize the agile values and principles along two dimensions: internal vs. external, and technical vs. social. Based on the mappings, they construct an agile conceptual framework, intending to get a more organized insight of the Manifesto and its focal areas, and guide agile software development by giving a higher-level and more conceptual view. As preliminary as it might be, their framework could be seen as a valuable theoretical effort because, as the authors contend, although the ideas behind agile software development are not new and innovative,

the way they are interconnected and brought forward is innovative and can provide new schemes for system development.

Wendorff (2002) makes an explicit theoretical linkage between agile software development and systems thinking. From the viewpoint of general systems theory, the author argues that systems thinking allows a clear and meaningful characterization of essential aspects of agile methods. Wendorff's work is focused on a specific facet of software development, namely project delay. Two sources of project delay are identified in the paper and how agile methods address them is convincingly explained in terms of general systems theory. Further work is needed to explore the application of general systems theory to the broader organization of agile software development process.

Jain and Meso (2004) explicitly use CAS theory to explain several practices suggested by various agile methods, such as iterations, minimal planning, and frequent feedback. Their work sheds some light on how CAS theory can provide useful insights for understanding how agile software development organizations function and organize themselves to accomplish the development task. Although empirical evidence has yet to be collected to demonstrate the linkages they assume between CAS theory and agile practices, their work show that applying the theory of complex adaptive systems in the study of agile software development could yield fruitful insights of agile organizations and practices.

3 COMPLEX ADAPTIVE SYSTEMS (CAS)

CAS theory is a branch of complexity science that studies how a complex system can be adaptive to its environment and how innovative properties of a system emerge from the interactions of its lower level components. Having originated in the natural sciences, CAS has been extensively applied in management and the study of organizations (Mitleton-kelly 1997, Brown & Eisenhardt 1998, Anderson 1999, Haeckel 1999, Stacey 2003). Anderson (1999) suggests that CAS should no longer be considered a new theory in organizational studies and that several concepts of CAS can be regarded as well-established scientifically, including interconnected autonomous agents, poise at the edge of chaos, self-organization, co-evolution and emergence. There are many accounts of CAS theory and although all are broadly in agreement there is no single, unifying theory of CAS. Volberda and Lewin (2003) draw out three overarching principles from various literature on complexity and organizations, which they believe are the "basic higher-order principles that must underlie any theory of self-renewal and its associated enabling managerial routines and capabilities involving strategy, structures, processes and leadership" (p. 2126).

Principle 1: managing internal rates of change to match or exceed the relevant external change rates. Organizations need to match or exceed the co-evolution rate of the systems (institutional configuration, industries, social movement, etc) within which the organization is embedded. Organizations must develop routines, capabilities and measures which monitor and track rates of change in all aspects of their environment and adjust the applicable internal processes to match or exceed these rates.

Principle 2: optimizing self-organizing. Self-organization is the process by which organizations find order no matter how complex or convoluted the structure of the organization. Self-organization requires a fundamental departure from command and control philosophy of traditional hierarchical bureaucratic organizations. Self-organization requires a belief in local rationality of individuals and units (e.g. those closest to the customer know the customer best). It is consistent with the often espoused idea of delegating decision making to the lowest possible level and it implies maximizing capabilities of scope at every level of organization. However, self-organization does not mean that individuals or units can behave at will and break all the rules; rules and structures are an essential part of self-organizing systems.

Principle 3: synchronizing concurrent exploration and exploitation. This principle is concerned with balancing concurrent innovation and knowledge creation (exploration) with improvements in productivity, process improvements, efficiency and product extensions and enhancements (exploitation). The long-term survival of an organization depends on its ability to engage in enough exploitation to ensure the organization's current viability and engage in enough exploration to ensure its future viability. Self-renewing organizations synchronize and balance concurrent exploration for new opportunities and exploitation of existing capabilities. Both attributes are accepted and present and operate simultaneously.

There are many accounts of CAS but a review of the literature (space does not permit a full account here, but see Anderson (1999) and [refs removed for anonymity] for more details) highlights a number of core concepts that relate to the three principles (Table 1).

Principles	Concepts
Principle 1: Managing internal rates of change to match or exceed the relevant external change rates	**Time-pacing** Change is triggered by the passage of time rather than by the occurrence of events. Organizations create an internal rhythm that drives the momentum for change (Brown & Eisenhardt 1998).
	Coevolution Parts of a complex adaptive system tend to alter their structures or behaviours as responses to interactions with other parts and the environment. In order to survive, all parts are striving for fitness and seeking to avoid extinction. Since they have to continue to respond to the change caused by coupled fitness landscapes they are not just evolving but coevolving. The highest average fitness is at the transition from order to chaos (the edge of chaos) (Kauffman 1993, 1996).
	Poise at the edge of chaos The edge of chaos is a zone between total order and complete disorder. At the edge of chaos, a system shows bounded instability, that is, it is stable and unstable at the same time. It is stable because the system's behaviour shows patterns, and the possibility space of the system's states can be depicted using fine detail in the short term. It is unstable in the longer term in that the path which the system will follow is uncertain and unpredictable - the system is in a state far from equilibrium (Anderson 1999).
Principle 2: optimizing self-organizing	**Interconnected autonomous agents** Interconnected agents – human and non-human. They are independent but loosely coupled and interconnected. The interconnectivity is in such a way that keeps them responsive to the change around them but not overwhelmed by the information flowing to them through their interconnectivity (Anderson 1999).
	Self-organization Self-organization is the ability of a complex adaptive system to evolve into an organized form without external force, resulting from the interactions among interconnected autonomous agents (Anderson 1999).
Principle 3: synchronizing concurrent exploration and exploitation	**Poise at the edge of time** Focus on today while keeping sight of the future (exploration) and the past (exploitation) (Brown & Eisenhardt 1998).

Table 1: CAS Organizing principles and related concepts

Brown and Eisenhardt (1998) define **time-pacing** as an internal metronome that drives organizations according to the calendar, e.g., "creating a new product every nine months, generating 20% of annual sales from new services" (p. 167). Time-pacing requires organizations to change frequently but can also stop them from changing too often or too quickly. Rhythm is used by organizations to synchronize their clock with the marketplace and with the internals of their business. Time pacing is therefore not arbitrary, although Brown and Eisenhardt give no indication as to how an organization might identify and set the pace of the internal metronome. **Coevolution** draws on the work of

Kauffman (1993, 1996) and has been used by many organizational theorists, including Milteton-kelly (2000) who uses it to study the relationship between the business and information system (IS) domains to gain insight into the problems of legacy systems. Coevolution can be applied to humans, e.g., users and developers, to technologies and artifacts, e.g., software and business processes, and to human-technology interactions, e.g., users and software. The **edge of chaos** lies between order and chaos and is a compromise between structure and surprise (Kauffman 1995). Anderson (1999), drawing on Kauffman's coevolutionary theory claims "Systems that are driven to (but not past) the edge of chaos out-compete systems that do not" (p. 223-224), a view that is reinforced by Brown and Eisenhart who argue that organizations that achieve the edge of chaos will compete more effectively than those that don't; at the edge of chaos "organizations never quite settle into a stable equilibrium but never quite fall apart, either" (p. 12). **Autonomous agents** and **self-organization** are central themes of CAS and reflect the view of organizyation as bottom-up and emergent rather than top-down and planned. There is no blueprint for the system – order emerges from the interactions of the agents. The **edge of time** is conceptualized by Brown and Eisenhardt (1998) as "rooted in the present, yet aware of the past and future" (p. 12). Managers must avoid being "mired in the past" but not so over-enamoured with the future that they waste time over-planning it. Organizations that focus on the past and exploitation become trapped but those that forget the past are always starting from new and repeating mistakes.

The organizing principles and concepts help to establish a context in which the emergence of system level properties is encouraged (Holland 1998). For software development the property of interest is agility. Various definitions of agility in organizations have been proposed (e.g., March 1991, Goldman & Nagel & Preiss 1995, Dove 1996, Samburthy & Bharadwaj & Grover 2003), but the wikipedia.org definition captures the essential aspects: "the ability of a firm to sense and respond to business opportunities in order to stay innovative and competitive in a turbulent and quickly changing business environment". Agility is an emergent property that is manifested at the system level. It is an organization's *qualitative* pattern of response to change (Dove1996, Conboy & Fitzgerald 2004) and is unlikely to be captured by lower level process metrics (Lappo & Andrew 2004). An essential foundation of agility is innovation, which is "the implementation of a new or significantly improved idea, good, service, process or practice which is intended to be useful" and can be categorized as product, process, marketing, and organizational innovation (wikipedia.org). The source of innovation is creativity. Thus, creativity and innovation are building blocks of agility.

4 RESEARCH METHOD

This research is an exploration of agile practices as seen through the lens of CAS and explicated in Table 1. An interpretive research approach is considered appropriate and beneficial for this study since a social constructivist perspective on the software development process emphasizes processes as made and enacted by people with different values, expectations and strategies, as a result of different frames of interpretation, which act as filters enabling people to perceive some things but ignore others (Melao and Pidd 2000). This perspective allows us to gain a rich insight into how agile projects are organized.

This study focuses on the software development process. Generally, the software development process is considered a framework for the tasks and series of steps that are required to build the software (Pressman 1997). A broader view of it also incorporates the tools used and the individuals building the software (Schach 1998), which is the scope adopted in this study. The case presented in this paper is an exploratory case study that forms part of a multiple case study programme of research.

Semi-structured interviews using open-ended questions have been conducted to collect data on the organization of the software development process of the Pongo Team. All four members of the team and one external collaborating developer were interviewed. Besides interviews, documentation review and field notes are the complementary methods used to collect data. The theoretical framework developed in Table 1 acted as a sensitizing device in the process of data collection and analysis.

5 THE PONGO TEAM

The Pongo team is part of an independent software house in Italy that works for a number of external clients on projects such as an inter-library loans system. The team has described its rigid software development process and the problems it had in meeting customer expectations prior to the adoption of agile methods (Dani et al. 2003). The Pongo team took its name from the Italian for 'play-doh', which they explained "in our search for a team name we liked the idea of linking it to the quality that we felt was most necessary to support change: malleability" (Dani et al. 2003). In January 2002 the Pongo team began a six-month immersive programme at XPLabs (www.xplabs.com) in Rome. At the time of our interviews, summer 2005, the Pongo team had three years intensive experience of working with XP. The Pongo Team is composed of four members: a manager (who is also at the management level of the company); a coach, who is also a developer; and two developers (The team also works with external collaborating developers).

The Pongo team organizes around three cycles: project, iteration, and work (Figure 1). A project is initiated by the configuration activity (Figure 1.A) and an initial user requirements capture, followed by a number of development iterations. The duration of each iteration is generally one week (Figure 1.B). Each iteration realizes the user stories the users and developers write and choose together during the planning game. At the end of the iteration, the development team delivers a piece of working software, and the customers test it according to the acceptance test they specified beforehand. The iterations are implemented on a daily basis through the work cycle described in Figure 1.C. A typical working day of the Pongo Team is eight hours with the day being measured using the "pomodoro" as a unit of time.

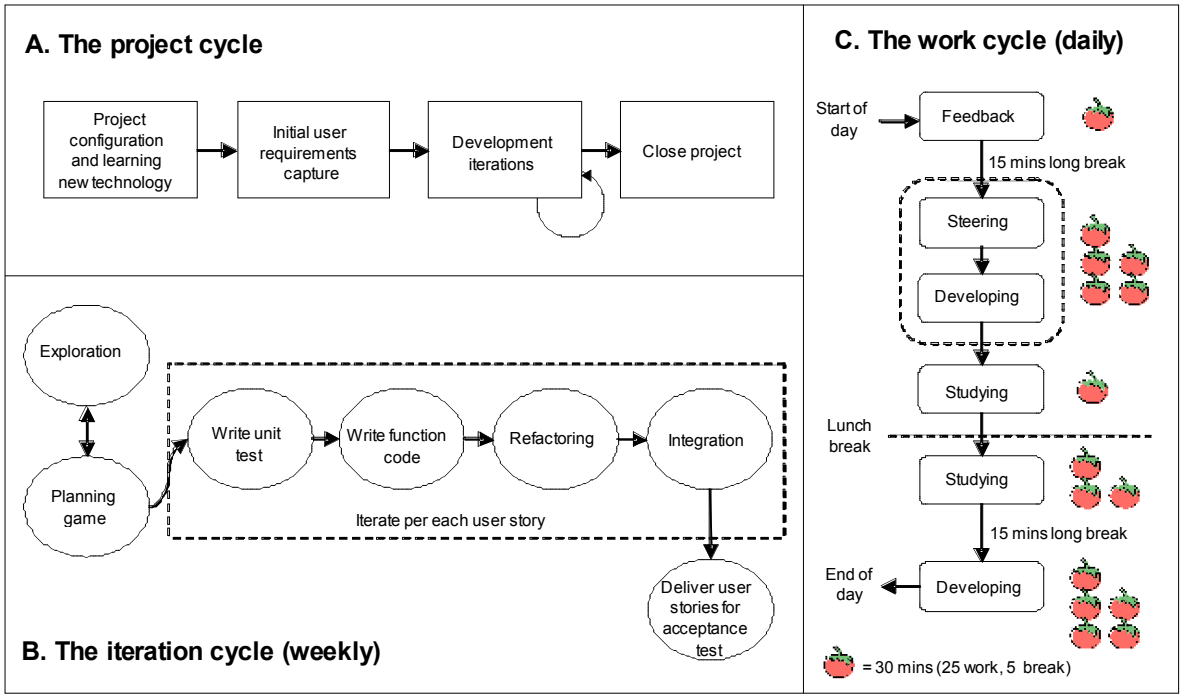

Figure 1: Time cycle orientation of the Pongo team

Pomodoro (tomato) originally was a tomato-shaped kitchen timer the Pongo team used at the XP training laboratory. It is set for 25 minutes of working followed by a 5 minute break. After the short break, the team gets back to work. Every four pomodoros the team takes a longer break of 15 minutes. The pomodoro has become a unit of time for the team and is used extensively to timebox working and

studying time. It is also used as a unit of measure to calculate the effort needed to deliver software; user requirements are captured on user story cards onto which a number of pomodoros are drawn to represent the work effort needed to complete the user story.

Having analyzed the software development process of the Pongo Team, several agile practices have been identified and linked to the relevant CAS principles and concepts of the conceptual framework presented in Table 1.

5.1 Principle 1: managing internal rates of change to match or exceed the relevant external change rates

Time-pacing is reflected by fixed and short iterations, typically one week. During the iteration, the user requirements are not allowed to change, although customers can check up on progress anytime they want. This approach gives the development team stability:

> *"The rate of the change is the iteration duration. If I have an iteration of 3 weeks, the time of not changeable is three weeks. If I have an iteration of one week, I have the rigidity for one week's development." (Manager)*

The team found that iterations of a week generally provide an appropriate time-pacing for development – iterations of less than a week led to hurry and anxiety. Iteration duration can be varied, for example the team might start with one week iterations in the early stage of the project where frequent customer feedback is needed but then move to two, or even three, week iterations as the requirements become better-defined: "We change duration of iterations according to the client's requirements" (Developer 1). However, the team prefers the consistency of weekly iteration cycles and are wary of changing the iteration time for fear of losing the rhythm of the project:

> *"When you can maintain a rhythm you have no anxiety. You have no worry about something particularly, so you are not stressed. And playing instrument is the same. I play the guitar so I know what I am talking about. When you reach the right rhythm, you can feel it. It is a special condition where work is ideal, but it is a special condition you can not reach every day … Rhythm is something that is very close to life. You are working without any anxiety, something that life does. And sometimes pomodoro help us to reach, to keep, to maintain the rhythm."(Coach)*

Within the working day there is a rhythm set by the pomodoro and this helps guard against over-working:

> *"I think it is a good rhythm, a good way to have rhythm because after 20, 25 minutes, you are a little tired. A little break is good to have the chance to be more focused on the next pomodoro. You can unplug your mind from the problem, but your mind still works, new ideas can come out and you can try on the next pomodoro. If you have to stay 8 hours [working] it's too difficult." (Developer 2)*

Time-pacing and weekly iterations help to drive the coevolution of developers/software and customers/business processes. The short iteration cycle means there is constant and rapid feedback from customers that allows developers to adjust priorities and customers to explore the fit of the software with their business environment. The key to this is the planning game that starts each iteration:

> *"The process is in this way: the customers write the user stories, the team reads the user stories, has to estimate the user stories by pomodori (tomatoes), which are the measure of efforts, not only pomodori, but pomodori per pair …. Now the customers and the team decide the iteration. The customer picks up a number of user stories and the team decides the effort to put in."(Manager)*

Without structure a fast-paced software development team might well pass the edge and tip into chaos. The key source of structure is the planning mechanism. Planning is fundamental to the Pongo team's work. It is not a question of whether planning is needed, it is about how to plan. It is about always planning. The planning game with the customers at the beginning of each short iteration and the daily feedback and steering give the team the ability to respond to change quickly by constantly adjusting the direction of the development process:

> *"We are planning everyday, every moment. But we don't plan the whole project. It is useless."(External Developer)*
>
> *"Always plan, this is the core."(Manager)*

5.2 Principle 2: optimizing self-organization

User stories are written on to cards and work estimates are made in pomodoro units. Developers self-recommend the user story cards they feel confident in completing and sign the card to show they are taking responsibility for delivery of that user story. Neither the manager nor the coach assigns tasks; the role of the coach is to observe and to facilitate in order to keep the development process smooth and swift: according to the Coach, "I try not to make decisions. I help something emerge."

A key requirement for self-organization is the ability for the system to be self-referencing, a property exhibited by the feedback and steering session that starts each day (Figure 1.C). The feedback is not only on the work process of the previous day, but also the feelings team members have, anxieties felt, or whether something is wrong. It is done quickly and briefly, generally one person per minute, and can be as simple as a single sentence. The steering session is when team members express problems about development activities and ask others for help.

The Pongo Team adopts pair programming practice of XP. The developers work in pairs based on self-pairing. They physically sit together and share one desktop while working. The pairing is not fixed, instead, pair rotations are frequent, to the point of one rotation every pomodoro.

> *"Doing so made sharing our perception of the progress of the project, or adopting new tools or practices, or finding collaborative solutions to complex problems, a natural part of the process. At first glance, frequent rotations might seem to slow down progress, but even today our experience is that overall we get an increase in speed." (Dani et al. 2003)*

The self-pairing is optimized by pair rotation. The pairs are working on different tasks. Between the two developers of a pair, the one who is responsible for the task they are working on (by signing his name on the task card) stays, the other goes to pair with a different developer. Sometimes developers from outside the team will come in and pair with team members as well. When a pair is stuck into a problem and can't find a suitable solution, the coach can decide to rotate pairs immediately instead of waiting for the end of a pomodoro.

> *"Sometimes it happens that one pair working for a long time on a task or a user story and they want to continue to work on it, want to see the end. OK, the responsible person should be frozen until the end but the other person sometimes works for a long time on the story and wants to see the end. It is not a good thing because the information about this particular part of the project could not be shared with others. Rotation helps to share. As a coach I decide to rotate the pair in this case." (Coach)*

5.3 Principle 3: synchronizing concurrent exploration and exploitation

The Pongo Team put aside four pomodoros for studying (Figure 1.C). The content of the study may or may not be directly related to the project. Any team member can suggest the subject or content of the study time. Study time is important in part because it introduces variety and provides the space for novelty to emerge:

"When sometimes we skip study time, we have to develop all the time so we are doing the same activity all day, our efficiency is lower. It's very important to switch between activities of different kinds, very important, because when we study, to pay attention to other issues, when we begin, start again the development, we can start with more imagination. [Exploring issues not directly related to the project] works better than always studying project related issues, because you could find some result that seems not useful for the project. But sometimes it works, magically. When planning everything, maybe (the magic won't happen)…" (Coach)

Study time is also important for learning and reflecting on behaviour:

"What is the effect of learning? That you change something … If I learn something, I modify, I change my behaviour. If I see the behaviour is the same there is no more learning in the cycle."(Manager)

5.4 Other practices

Not all of the practices found in the Pongo team fitted neatly into the CAS framework. Of particular importance to the Pongo team was the test-driven approach to development adopted for both the delivery of user stories to customers and the development of software within the Pongo team.

User acceptance tests are written with, and agreed by, the customer during the planning game. The only delivery to the customer is working software, as summarized by the External Developer: "We deliver software, never documents. There is no need of them, because there is acceptance test. We developed what the customers described in acceptance test, they do the test, so they know how to use it." Similarly, the developers write unit test code before writing function code, arguing that they get less bugs, more confidence in the code, new ideas for the application, and better design (Dani et al. 2003). When the team needs to understand a new software package they write tests to see in unambiguous terms how the software behaves and what it is capable of doing (ibid.). All of these practices point to an emphasis on external behaviour rather than internal workings.

6 DISCUSSION

Table 2 provides a summary of the lessons from the Pongo case. Time-pacing is a fundamental building block of agile software development that drives the coevolutionary engine of developers and users as well as providing a structure for short-term planning. Self-organization is promoted through autonomous agents that are connected but not overwhelmed by those connections. The case demonstrated the practice of encapsulating behaviour as a way of dealing with complexity of connections between agents (human and non-human) as illustrated through the test-driven approach to user acceptance and programming. This suggests that the object-oriented (OO) paradigm is a useful concept in organizing the connections between agents.

CAS theory would suggest that achieving agility is not a pick and mix approach; all of the concepts are heavily intertwined and mutually reinforcing. For example, it might be tempting to focus on time pacing and self-organization, but to remove the space and resource for exploration, but this approach would give an inappropriate weighting to exploitation with the result that innovation and agility would likely suffer. It is also essential to note that a CAS/ASD approach does not guarantee success or survival and that the outcome is inherently unpredictable. The role of managers is to set the context in which self-organization and emergence can be encouraged, to find more subtle ways "to bound, direct, nudge, or confine, but not to control" (Highsmith 2000, p.40). Streatfield (2001) presents the traditional view of managers as 'in control' as selecting, designing, planning a course of action, correcting deviation, working in a stable environment with regular patterns, conformity, and consensus forming. Streatfield (ibid.) continues with the 'not in control' view, which sees action as evoked, provoked, emerging, amplifying deviation, and an unstable and unpredictable environment with

diversity and conflict (figure 8.2, p. 134). Rather than accept one or other of these poles managers must work with the paradox of control – they are simultaneously in control and not in control.

Concepts	Learning from case and implications for software development
Time-pacing	The regular delivery of user stories, e.g., weekly, provides a rhythm for the software development process that gives stability to the project team, a relief against anxiety, and a guard against over-working; The rhythm is not the same for all teams and all projects and all times – the team must find its own rhythm in each specific context; Rather than being a recipe for chaos, fixed iterations help to stop a project from over-responding to change.
Coevolution	Through coevolution user organization and software development organization mutually adapt each to the other through coupled deforming fitness landscapes; The coevolutionary process requires continuous knowledge sharing of users and developers (developers need to understand the constantly changing business setting and users the capability of technology). The principal exchange mechanism is the XP planning game.
Poise at the edge of chaos	There needs to be sufficient freedom to cope with change but enough structure to stop the software development process from falling apart. The planning cycle allows for software evolution (and the possibility of the emergence of innovative solutions) but supplies a barely sufficient structure in the form of iterations and daily work cycle; Planning is of central and fundamental importance, but it is bottom-up based on user stories rather than top-down at the project level.
Interconnected autonomous agents	Developers in a software development team require a degree of autonomy, e.g., they decide which user stories they are best fitted to implementing; Pair programming builds connections between agents and promotes collaboration, problem-solving, and knowledge sharing; The OO paradigm of encapsulation and communication by messages (black-boxing) is a valuable principle for coping with information overload. Users can see what stories will be delivered from the plan but not the inner details of how the plan is executed. Test driven development is also a form of black-boxing in which software behaviour takes priority over code.
Self-organization	Managers do not plan and control in a rigid way, e.g., pair programming is managed primarily by self-pairing. Managers must foster a context that allows the agile project to evolve and coevolve to the edge of chaos where innovation and creativity are encouraged and emergence of new structures and forms is possible. The ability to self-reference through reflection on what is working, and what is not working, supports self-organization.
Poise at the edge of time	Slack resource is needed for studying to promote exploration and to support emergence and innovation. In the case this is achieved by making a set period available each day for study and exploration; The 'magic' of exploration might be forthcoming, but it is not guaranteed.

Table 2: CAS concepts and lessons for agile software development

6.1 Limitations and future directions

The research reported here presents a single case study and the framework needs to be refined and tested with further cases. There is also a deeper concern whether CAS is appropriate to the study of human organizations. Capra (2002) argues that we must adapt CAS for new domains to take account of social aspects, such as power relationships, by drawing on social theory, philosophy, cognitive science and other disciplines (p. 71).

The Pongo team consider that they are now agile, that they no longer fear change but embrace it as "a positive tension that moves us to act" (Dani et al. 2003). The study time creates the time and resource for the team to be creative and to innovate through the introduction of new technologies and practices as well as allowing the space for emergence to unfold, for example in the form of new and better software designs. The research reported here relies to a large extent on observation and self-reporting of agility and further work is therefore needed in the assessment of agility to allow informed judgements to be made concerning the effectiveness, or otherwise, of ASD practices.

For the organization as a whole to be agile, i.e., not just the software development team, then the customer must also have the space to evolve their business practices such that the business process and the software development process coevolve. The coevolution of business and IT suggests that CAS may help us understand the business/IT alignment issue (Luftman 2005) in a new light. However, to study coevolution in this way will require research designs that take greater account of the relationships between developers and business users as well as other stakeholders.

7 SUMMARY

The contribution of this paper has been to use complex adaptive systems (CAS) theory to develop a theoretical grounding for the practice of agile software development. Drawing on Volberda and Lewin's (2003) three principles - matching change rates, optimizing self-organization, and synchronizing exploration and exploitation – six concepts were identified: time-pacing, coevolution, the edge of chaos, interconnected autonomous agents, self-organization, and the edge of time. Jointly, these concepts have the potential to promote system level emergence of agility (defined by Highsmith and Cockburn (2001) as the ability to create and respond to change), a property that cannot be reduced to any of the individual parts or to the lower level practices in the system. The CAS framework was applied to a case study of the Pongo team, who organize their software development using eXtreme Programming. A strong correspondence between CAS theory and the practice of ASD practice was found and summarized in Table 2. In addition, the Object-Oriented concept of encapsulation of behaviour was found to be a valuable concept that is not readily apparent from the CAS literature. Future work will involve conducting more case studies with software development teams and developing substantive theories of organizing to achieve agility in the software development process based on CAS theory.

References

Abrahamsson, P. and J. Koskela (2004). Extreme Programming: Empirical Results from a Controlled Case Study. IEEE International Symposium on Empirical Software Engineering, Redondo Beach CA, USA, ACM.

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

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

Capra, F. (2002). The Hidden Connections. Harper-Collins, London.

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

Crocker, R. (2004). Large Scale Agile Software Development. Extreme Programming And Agile Methods - XP/ Agile Universe 2004, Proceedings. Berlin, Springer-Verlag Berlin. 3134, 231-231.

Dani et al. (2003). "How We Became the Pongo Team", XP2003 Conference, Genova, Italy, May 2003.

Dove, R. (1996). Tools for Analyzing and Constructing Agile Capabilities. Agility Forum, PA96-01, Jan 1996.

Goldman, S. L., R. N. Nagel and K. Preiss (1995). Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer. Van Nostrand Reinhold, New York.
Haeckel, S. (1999). Adaptive Enterprise: Creating and Leading Sense-and-respond Organizations. Harvard Business School Press, Boston.
Highsmith, J. (2000). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. Dorset House Publishing, New York.
Highsmith, J. (2002). Agile Software Development Ecosystems. Addison-Wesley, Boston.
Highsmith, J. and A. Cockburn (2001). Agile Software Development: the Business of Innovation. Computer, 34(9), 120-122.
Holland, J. H. (1998). Emergence: From Chaos to Order. Oxford University Press, Oxford.
Jain, R. and P. Meso (2004). Theory of Complex Adaptive Systems and Agile Software Development. Proceedings of the Tenth Americas Conference on Information Systems, New York, New York, August 2004, 1661-1668
Kalermo, J. and J. Rissanen (2002). Agile Software Development in Theory and Practice, M.Sc. Thesis on Information Systems Science. University of Jyväskylä, Jyväskylä, 2002.
Kauffman, S. (1993). The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press, New York.
Kauffman, S. (1996). At Home in the Universe: The Search for Laws of Self-Organization and Complexity. Oxford University Press, New York.
Lappo, P. and H. C. T. Andrew (2004). Assessing agility. Extreme Programming And Agile Processes In Software Engineering, Proceedings. Berlin, Springer-Verlag Berlin. 3092, 331-338.
Luftman, J. (2005). Key Issues For IT Executives 2004. MIS Quarterly Executive, 4(2), 269-285.
March, J. G. (1991). Exploration and Exploitation in Organizational Learning. Organization Science, 2(1), 71-87.
McKelvey, B. (1999). Avoiding Complexity Catastrophe in Coevolutionary Pockets: Strategies for Rugged Landscapes. Organization Science, 10(3), 294-321.
Melao, N. and M. Pidd (2000). A Conceptual Framework for Understanding Business Processes and Business Process Modeling. Information Systems Journal, 10(2), 105-129.
Mitleton-kelly, E. (1997). Organisations as Co-evolving Complex Adaptive Systems. British Academy of Management Conference, BPRC (Business Processes Resource Center) Paper Series, No 5.
Murru, O., R. Deias and G. Mugheddu (2003). Assessing XP at a European Internet Company. IEEE Software, 2003, 37-43.
Pressman, R. S. (1997). Software Engineering: A Practitioner's Approach, McGraw-Hill.
Rasmusson, J. (2003). Introducing XP into Greenfield Projects: Lessons Learned. IEEE Software, 2003, 21-28.
Sambamurthy, V., Bharadwaj, A., and V. Grover (2003). Shaping Agility through Digital Options: Reconceptualizing the Role of Information Technology in Contemporary Firms. MIS Quarterly, 27(2), 237-263.
Schach, S. R. (1998). Software Engineering with JAVA, McGraw-Hill.
Schuh, P. (2001). Recovery, Redemption, and Extreme Programming. IEEE Software, vol. 18, 34-41.
Schwaber, K. (1996) Controlled Chaos: Living on the Edge. American Programmer, April 1996.
Stacey, R. D. (2003). Strategic Management and Organisational Dynamics: The Challenge of Complexity. Fourth Edition. Financial Times Prentice Hall.
Streatfield, P. (2001). The Paradox of Control in Organizations. Routledge, London.
Turk, D., R. France, et al. (2002). Limitations of Agile Software Processes. Third International Conference on eXtreme Programming and Agile Processes in Software Engineering. Alghero, Sardinia, Italy.
Volberda, H. W. and A. Y. Lewin (2003). Guest Editors' Introduction Co-evolutionary Dynamics Within and Between Firms: From Evolution to Co-evolution. Journal of Management Studies, 40, 2111-2136.
Wendorff, P. (2002). An Essential Distinction of Agile Software Development Processes Based on Systems Thinking in Software Engineering Management. Third International Conference on eXtreme Programming and Agile Processes in Software Engineering, Alghero, Sardinia, Italy.