

ORDER AND CHAOS IN SOFTWARE DEVELOPMENT: A COMPARISON OF TWO SOFTWARE DEVELOPMENT TEAMS IN A MAJOR IT COMPANY

Xiaofeng Wang, Lero, University of Limerick, Limerick, Ireland, xiaofeng.wang@ul.ie

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

Abstract

Agile methods have emerged and become popular over last few years as a response to shortcomings of the waterfall process model. However, agile processes are stamped by some as chaotic processes and are placed in opposition to waterfall approaches. This paper uses the edge of chaos concept from complex adaptive systems theory as a theoretical lens to analyse the roles of structure and planning in the software development process. The software development processes of two teams in a major IT company, one of whom uses agile methods and the other a waterfall approach, are presented and the project structure and planning process of each is highlighted then compared. Our research finds that structure and planning are essential to agile processes and take different forms from the waterfall model. Contrary to the belief that agile software development may be chaotic we conclude that it is possible that the waterfall method may be chaotic due to prescribed rather than effective structures.

Keywords: Agile Processes, Agile Methods, Complex Adaptive Systems, the Edge of Chaos, Structure, Planning

1 INTRODUCTION

Software development has long been considered demanding due to the inherent complexity of software products and the consequent problematics of the development process (Brooks 1987). It becomes even more challenging in an unpredictable and constantly changing business environment. Traditional software development approaches, characterized by a linear waterfall lifecycle, are considered incapable of handling this complexity (Highsmith 2002) and in the past few years, agile software development has emerged as a promising solution to the problem of software development complexity. Various agile methods have been proposed and some have gained great attention in the software engineering community, especially eXtreme Programming (XP) (Beck & Andreas 2004) and Scrum (Schwaber & Beedle 2002). However, since they came into being as a response to the failure of waterfall approaches and downplay the importance of a formal process and planning, agile methods have been pushed to the opposite of waterfall approaches (Boehm 2002) and considered generators of chaos (Rakitin 2001). But is chaos really a proper metaphor for agile processes, that is, software development processes applying agile methods?

This paper sets out to shed some lights on this question. Our objective is to provide a better understanding of the nature of agile processes, which can in turn benefit the effective application of agile methods in software development. To this end, our study draws on complex adaptive systems (CAS) theory, which can provides valuable insights to understand complex systems, adaptation and related issues (Anderson 1999). CAS theory has been applied in management and the study of organizations (Brown & Eisenhardt 1998, Haeckel 1999, Mitleton-kelly 1997, Stacey 2003). It has also been used to study agile software development (Augustine et al. 2005, Meso & Jain 2006, Vidgen & Wang 2006). In particular, this paper uses the idea of the “edge of chaos” from CAS theory to study the software development process. This paper is based on our empirical study of two software development teams in the same division of a major IT company, one of them applying an agile methodology, XP, the other using the waterfall approach. Both software development processes are analysed using the lens of the “edge of chaos”, which is part of a broader CAS-based conceptual framework (Vidgen & Wang 2006). The paper is organized as follow. Section 2 introduces an ongoing debate about agile methods and waterfall approaches. The edge of chaos concept is elaborated in Section 3. In Section 4 the two cases are presented and compared, and the findings are summarized. Then Section 5 discusses the findings and their implications for software development using the notion of the edge of chaos. The last section concludes the paper and indicates future research.

2 WATERFALL VS. AGILE: AN ONGOING DEBATE

Early software development often took place in a rather chaotic and haphazard manner, relying entirely on the skills and experience of the individual staff members performing the work. The waterfall model is the earliest method of structured system development and is widely used to counteract the ad-hoc development of software (Avison & Fitzgerald 2003a). The waterfall model is a linear process model that advocates using extensive planning, upfront and in detail, to make development a predictable activity (Boehm 2002). Although it is widely used and considered a dominant approach, it has been criticized in recent years. Avison and Fitzgerald (2003b, p.34) give a list of criticisms of the waterfall model, among which are instability and lack of control, perhaps to the surprise of those who believe that the waterfall model provides a basis for a predictable and controllable process.

An alternative model of software development process to the waterfall is the iterative model. The basic idea of the iterative model is to develop a software system incrementally, so allowing the developer to take advantage of what was being learned during the development of earlier, incremental, deliverable versions of the system (Basili & Turner 1975). Having a history arguably as long as the waterfall

approach (Larman & Basili 2003), the iterative model has gone through several cycles of popularity and has recently gained new momentum in the agile software development movement. A set of “light-weight” software development methods have emerged in the past several years as a response to the rapidly changing world and the inefficiency of conventional “rigorous” software development methodologies (Highsmith 2002). XP and Scrum are the most popular ones. The proponents of agile methods argue that software development is not a mechanistic defined process, but an organic empirical one (Schwaber 1996), as a consequence, the belief of complete pre-knowledge and control of the process embedded in the traditional methodologies is unrealistic. “In chaotic environments, success is accidental” (Highsmith 2000, p. 29). They question the values of detailed planning over the entire lifespan of a project, precise prediction and rigid control strategies characterized in the waterfall model, and advocate more subtle ways, “to bound, direct, nudge, or confine, but not to control” (Highsmith 2000, p. 40). The values and principles behind various agile methods have been summarized and explicated in the agile manifesto (<http://www.agilemanifesto.org>): individuals and interactions over processes and tools; working software over comprehensive documentation; customer collaboration over contract negotiation; responding to change over following a plan. The agile manifesto acknowledges that there is value in the terms on the right (e.g., processes and tools) but places greater value and emphasis on the terms on the left (e.g., working software).

Although the proponents of agile methods have demonstrated the successful application of their respective methods (Beck & Andres 2004, Highsmith 2000, 2002, Schwaber & Beedle 2002), scepticism and criticism have been raised around agile values and practices. Rakitin (2001) argues that process, documentation, user contract and planning are essential in software development, whereas agile values, such as people interaction and responding to change, reflect a hacker culture which allows people to irresponsibly write code and have no respect for engineering discipline. His “hacker interpretations” of the Agile Manifesto puts agile methods at the opposite polar of process and discipline, and regards agile values as chaos generators. In the same vein, in their book titled “Extreme Programming Refactored: The Case Against XP”, Stephens and Rosenberg (2003) doubt both XP practices and the philosophy behind XP and try to restore the values of documentation, upfront planning and design in software development. Boehm (2002) notices that many agile advocates also consider the agile and plan-driven software development methods to be polar opposites. Baskerville (2006, p. 113) suggests that this polar view is due to the fact that, in the attempt to clarify what agile really means, “authorities seeking to describe agile software development methods often cast about for its opposite. These opposites are sometimes called ‘disciplined’ approaches or ‘plan-driven’ approaches”. While Boehm (2002) attempts to synthesize the two opposites thus implicitly taking an either/or approach and admitting that planning is not an inherent part of agile processes, Baskerville (2006, p. 113) doubts this dichotomy and puts forward the question “But is agility really the opposite of discipline and planning”? Following Pettigrew et al. (2003) we will recast the either/or as a both/and. In the following section we draw on the edge of chaos concept in CAS theory to provide a new angle to understand agile processes.

3 SOFTWARE DEVELOPMENT AS STRUCTURED CHAOS

According to Gell-Mann (1995), when something is in complete order or complete disorder, the effective complexity is zero. Effective complexity can be high only in a region intermediate between total order and complete disorder. This critical zone is called “the edge of chaos”, where “the components of a system never quite lock into place, and yet never quite dissolve into turbulence, either” (Waldrop 1994, p. 12). Stacey (2003) terms it bounded instability, that is, stable and unstable at the same time. It is stable in the sense that the system shows patterns of behavior and the possibility space of the system’s states can be depicted using fine detail in the short term, but unstable in the sense that the path which the system will follow is uncertain and unpredictable in the long term. McKelvey (forthcoming) suggests that, in the context of organizations, it is better to think of a “region of emergent complexity” rather than an “edge of chaos”. This region lies between stasis and chaos and

is defined by two critical values. If an organization falls below the first critical value because it exhibits minimal response to addressing the adaptive tensions it faces then order will prevail. If the organization over-responds to its adaptive tensions, for example, by initiating too many change programmes too quickly, then it may exceed the second critical value and chaos will ensue.

Why would organizations poise at the edge of chaos (or the region of emergent complexity)? Because it is where all the really interesting behaviour occurs in a system, “where life has enough stability to sustain itself and enough creativity to deserve the name of life”(Waldrop 1994, p. 12). 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), and “gives them a selective advantage: systems that are driven to (but not past) the edge of chaos out-compete systems that do not” (Anderson 1999 citing Kauffman 1995). According to Stacey (2003, p. 262), the dynamic in this region should be understood as a “dynamic of paradox, that is, simultaneous stability and instability, which is a requirement for the emergence of novelty”.

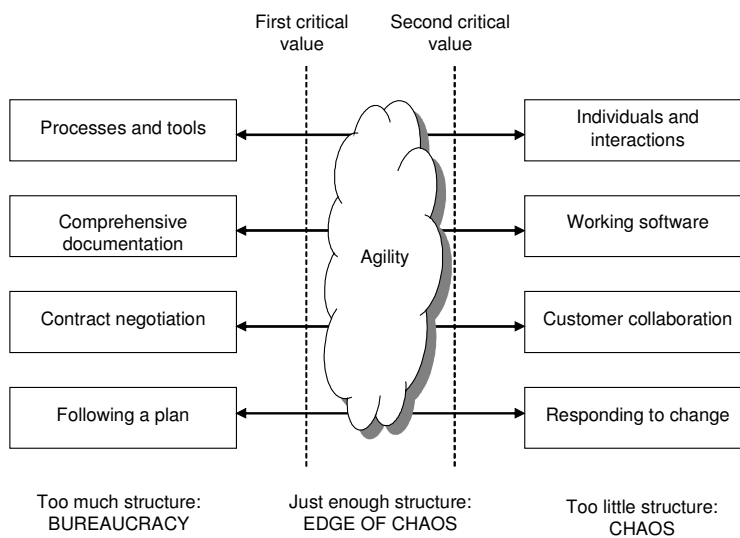


Figure 1. The agile/edge of chaos conceptual framework

Agile processes have been viewed as chaotic processes by some, as reviewed in Section 2, which put agile methods and the waterfall model on the opposites of a spectrum of increasing emphasis on structure and planning. However, according to the spectrum from order to chaos with the transit zone - the edge of chaos - somewhere in the middle (Figure 1), it can be argued that, to be responsive to change, to be adaptive and innovative, agile processes should not be chaotic, rather, they should poise at the edge of chaos. Too little structure and the software development project will exceed the second critical value and fall into chaos - Brown and Eisenhardt (1998) identify the warning signs as a rule-breaking culture, loose structure and chaotic communication. Too much structure and the project will fall below the first critical value and into bureaucracy (the warning signs being a rule-following culture, rigid processes and structures, and channelled communication). In the next section, the software development processes of two teams from the same division of a major IT company are presented and compared to illustrate our understanding of agile processes.

4 A COMPARISON OF TWO TEAMS

The case studies reported here follow an interpretive research approach which is considered appropriate and beneficial for our study. Semi-structured interviews using open-ended questions have

been conducted to collect data on the organization of software development process. Documentation review and non-participative observation have been used as complementary data collection methods. A theoretical framework based on CAS theory (Vidgen and Wang 2006) has acted as a sensitising device in the process of data collection and later analysis.

4.1 Organizational context

The two cases are two software development teams (Teams A and B for the purposes of anonymity) in the same division of a major IT company. The company is considered to be a hierarchical and bureaucratic organization by the interviewees. Agile methods have not been formally introduced to all development teams, but some teams have been using XP for several years. Team A is one of them and the earliest to adopt XP. In contrast, Team B apply the waterfall model which has been used in the company for a long time and which the senior management of the company has pushed down a directive to use. This directive has been shielded by the management of the division, especially the people manager who is directly in charge of Team A and B, so that Team A can continue to use XP. The head of the division is keen to try agile methods and has been supportive to the teams using agile methods, but expressed the feeling of being afraid of losing control over these teams, whereas the people manager who has direct knowledge of the development processes of the teams believes that teams using agile methods are actually more predictable.

The profiles of the two teams are as shown in Table 1.

| | Team A | Team B |
|-------------------------|---|--|
| Development methodology | XP (eXtreme Programming) | The waterfall approach mandated by the company |
| Development tools | Java and VB based tools | C++ based tools |
| Team composition | <ul style="list-style-type: none"> • 7 developers (among whom 1 project manager, 1 team lead and 1 technical lead) • 1 test manager • 1 onsite customer | 5 developers (among whom 1 project manager) |
| Project | <ul style="list-style-type: none"> • Redeveloping a legacy system • Web application hosted on the servers of the company • The nature of the system is very much GUI-driven (Graphic User Interface) | <ul style="list-style-type: none"> • A rolling project • Backend service residing on and assessing customer computing environment • The nature of the system is a command-line package)therefore limited interface with users) |
| Customer | Customer service organizations of the company who then deal with external customers | Field service engineers of the company who in turn deal with external customers |

Table 1. *The profiles of the two software development teams*

4.2 The software development processes of the two teams

In our study we adopt a broad definition of the software development process. It is not only a framework for the tasks and series of steps that are required to build the software (Pressman 1997), but also incorporating people that build the software and tools that support development work and the interactions among people (Schach 1998). But for the purpose of the paper we concentrate on the structure of and planning in the processes.

4.2.1 *The software development process of Team A*

The software development process of Team A is illustrated in Figure 2.

(1) The project cycle

At the beginning of the project, the onsite customer collects initial user requirements by workshopping for a piece of functionality and elaborates them with the team. When the team start to develop that piece, the customer goes on to define other requirements. So the requirement gathering starts at the beginning of the project but goes along in parallel with the development throughout the whole project. After the initial scoping phase, the project manager, team leads, onsite customer and business sponsors have a release planning meeting in which the areas that need to work on are decided and the requirements are broken down into user stories. A user story should have four attributes: it has business value, it can be estimated, it can be tested, and it acts as a token for a conversation. It is not intended to be a definitive definition of a requirement. Although there are no intermediate releases during the project as mandated by the business, several release planning meetings have been held to review the initial plan and the focuses of the next iterations. After a release planning meeting, the team begin the iterations. All code is developed in iterations (see “The iteration cycle”). When development work is completed, the system goes into testing. Since the team do not deliver to a live environment at the end of each iteration, an overall test called regression test is run on the complete system. Then the build is deployed in the live environment for customers to do field testing and eventually use the system. At the end of the project a postpartum meeting is held to address wider issues including the connection of the development team with support and how the team interface with other groups.

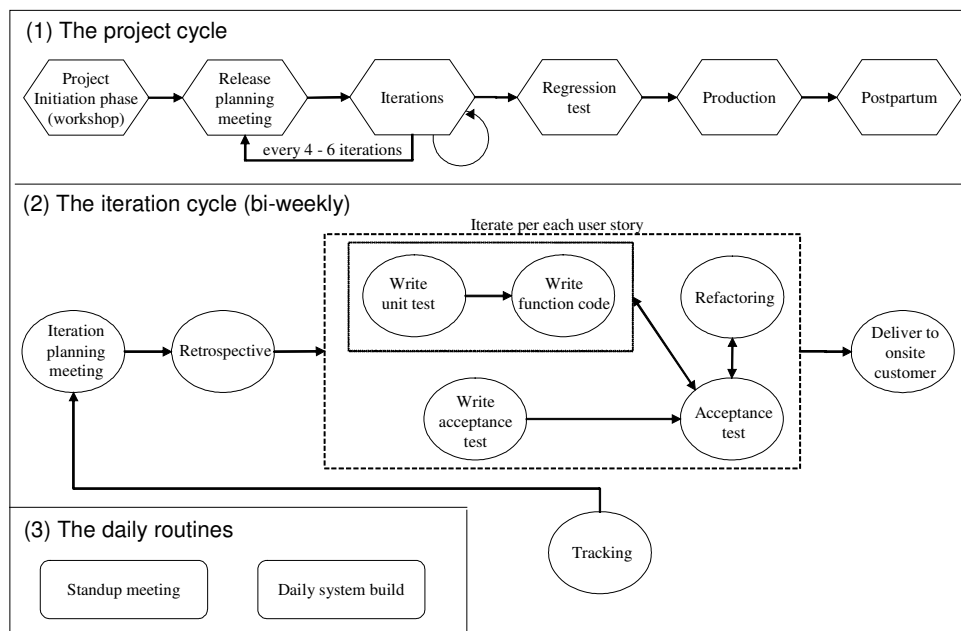


Figure 2. The software development process of Team A

(2) The iteration cycle

The team uses two-week iterations, as suggested by XP, and generally do not change the length of the iterations. An iteration begins with an iteration planning meeting in which all the team participate, including the onsite customer and the test manager, to determine what the team do in the next two week period. The onsite customer introduces each user story and the team discuss them in enough detail to provide an estimate of effort required. The estimates are expressed in “perfect engineering days”, i.e. 8 hours devoted to the problem at hand with no interruptions. A user story can be as small as 0.25 perfect engineering day, but no bigger than 4. Any user story with an estimate bigger than 4 is considered a sign of it not being at the right level of granularity. The development team inform the

customer of the velocity, i.e. the number of perfect engineering days of work that the team can complete for the current iteration. The customer then prioritises the user stories to be completed in that iteration, and ensures that the total estimates for all stories in the iteration are less than or equal to the team's velocity. The user stories that need to be implemented in the iteration are written on quarter A4-sized paper cards and stuck on the cubicle walls, which serve as story boards. In principle the user stories are not allowed to change during an iteration, but the team do adjust the user stories according to suggestions from the customer. After the planning meeting, the team generally run a retrospective to review the process used during the previous two weeks. The team provide feedback on the process under three categories: (a) keep these - things that worked well throughout the iteration; (b) problems - issues with the process identified during the iteration; and (c) try these - suggestions to alleviate the issues identified in (b) or things that would improve the process. The onsite customer participates in the retrospective of the team as well. She is involved in every step of the development process and gets the opportunity and flexibility to steer the software.

The team lead tracks the progress on the stories twice per iteration. The tracking involves asking each story owner to estimate the amount of time spent on a story to date and the amount of time remaining. The information is then fed back to the next iteration planning meeting to determine the actual velocity of the team in that iteration.

The team adopt a test first development pattern. Developers write a test for a piece of function before writing the code to implement that function. Each user story has a set of acceptance tests associated with it. While the developers implement user stories, the test manager and the customer are writing the acceptance test scripts. When a user story is implemented, it can be tested without delay and most bugs caught by testing can be fixed within the iteration.

(3) The daily routines

The team have a stand-up meeting everyday at 10 o'clock. All participate in the meeting, including the onsite customer. The team members report the progress and issues of the work from the previous day and quickly go through the work of the day. Another routine is daily system build. It is required that all code will be checked in to the code base at least daily. Before checking code in, all unit tests for that code must pass. The system is built automatically on build machines and deployed to the test server at least once a day. The deployment is available for the onsite customer to review the ongoing functionality and to assess it.

4.2.2 *The software development process of Team B*

The software development process of Team B is illustrated in Figure 3. Generally the team deliver a new major version of the product within 12 months. At the beginning of the project, in the conception and scoping stages, the project manager does a high level brainstorming with clients over what needs to be done for the next 12 months, and get a high level view of what are the major deliverables. These are the overall business requirements which do not involve input from the developers. The business requirements are broken down into product requirements with more technical details in the analysis stage. The project manager then consults the team members for assigning development tasks and works out a work schedule for the whole project. A task generally takes no less than 1 day but no more than 5 days, except for a very predictable piece of work which may have an estimate of up to 3 weeks. The work schedule has been reviewed and updated by the project manager a couple of times during the project.

Since changes happening to the project come frequently from management level, such as organizational changes and project cut or scope changes, which demand the team to terminate the project and quickly deliver something, the team have twisted the waterfall process and evolved a phased approach. Instead of having one big construction (coding) phase, they break it up into several 4 to 6-week phases. Each phase is composed of a set of mini-waterfall steps, including analysis, design, coding and testing. The most crucial blocks of work are put in early phases and packaged in a self-

contained way that, though the team do not actually deliver anything to the customer at the end of a phase, if something happens and the team have to make release, they would be able to do so.

After development phases are completed, a long test stage will follow – up to 10 weeks with accordingly high costs. After the system is deployed, a postpartum is held at the end of the project to review all the issues in the project. Everyone on the project and any other key parties participate in the meeting, including the people manager.

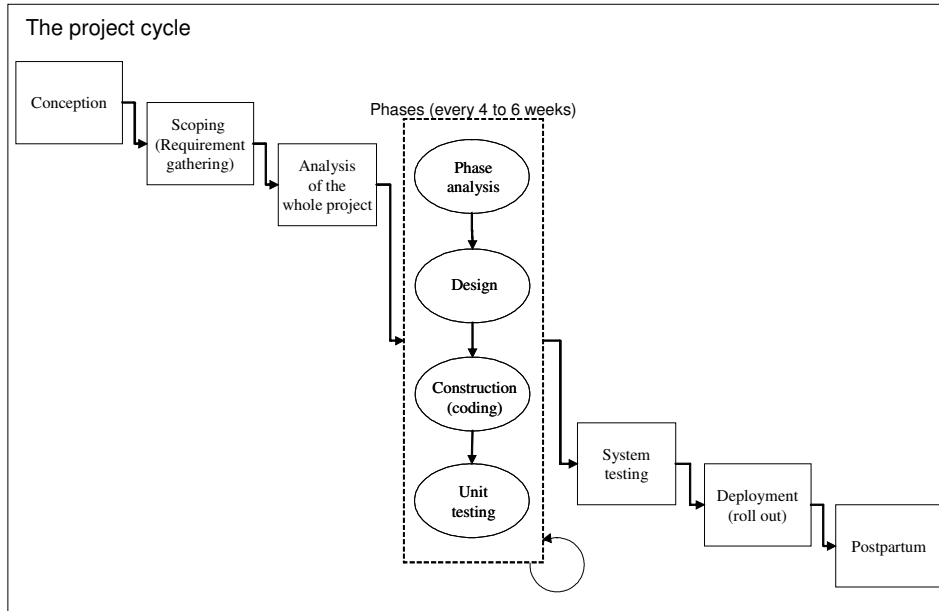


Figure 3. The software development process of Team B

Certain types and amount of documents are produced at the end of each stage and need to be signed off by the senior architects and the people manager. These documents are tokens that are used to get through the gateways from one stage to the next. A change control mechanism is used for significant changes either to the scope or to the schedule of the project. The project manager needs to fill out a change request and get it signed out by the people manager and the senior architects before the team can implement the change. Depending on where the project is, the team have meetings at different frequency. They formally meet once every 2 or 3 weeks in the middle of a design phase. During the intensive test phase, they meet formally every few days. There are no daily routines in the team’s development work.

4.3 A comparison of the two processes

4.3.1 Structure

In the case of Team A, the team members believe they have a formal process and there is a structure in the process, which brings a sense of certainty to the team. The formality and structure come from the routines in the process, such as 2-week iterations, iteration planning meetings, retrospectives and stand-up meetings. The test first coding pattern is also a routine that the team members stick to:

“There’s a structure there and I like it, and kind of you know what’s ahead of you.” (Team A)

“I suppose it’s kind of formal process, you know, that’s the difference, you have a form. [but waterfall is not formal?] it’s not, I mean, it’s do the work, test it, and leave it, really it isn’t.” (Team A)

“You can not be so flexible that things are chopping and changing every single day... We have to be rigid in order to be agile.” (Team A)

Meanwhile, Team A periodically review their process to avoid process rigidity or deterioration. It is considered essential for agile processes by the team:

“You don't have an agile process unless you have it (the retrospective) ... that constant review process, you know, from planning point of view helps us to maintain the sustainable pace.” (Team A)

The routines and the frequent retrospectives make the process internalised in the minds of the team members:

“I suppose people know the process, in this case so, you know, keep stand-up meetings or keep onsite customers, which were sort of things inside on your own ...” (Team A)

In the case of Team B, in contrast, the team members have no clear awareness of the process they are using, neither do they care about it. It is seen as imposed by the management. No retrospective regarding process has ever been conducted.

“I mean we don't concern as much the process ... maybe the project manager, I should say he knows more about the approach, me personally I just yeah I have a task, I implement it, that's it.” (Team B)

The officially signed-off documentation plays the role of gateways in the process of Team B, and is the main source of formality of the process. There are no built-in routines in the process, rather, the team members have a negative opinion on routines.

“[Do you have daily meetings?] No. I wouldn't like them anyway, daily. [You know the other team they have daily meeting?] They're losing precious time. When they could be coding they are having meetings.” (Team B)

4.3.2 Planning

In the case of Team A, they have several release planning meetings in the course of the project where they make a sketchy plan for a period of 4 to 6 iterations. Then they have a detailed plan for each 2-week iteration at the iteration planning meeting. The daily stand-up meeting functions as a quick steering of their daily work:

“We did have a lot of focus on planning, but we had a very fluid plan, fluid in the sense in that the plan changed from iteration to iteration.” (Team A)

The team do not follow plans rigidly. The release plans are reviewed regularly and the execution of iteration plans are tracked for the purpose of improving the team's ability of planning:

“What you find in your initial plan is that, as you work on stories, other stories come out that things keep added, particularly, this is actually a quite complex project ...” (Team A)

Since Team A work on short iterations and break down the user requirements to a fine level of granularity (no bigger than 4 perfect engineering days), they can plan their work for each iteration accurately and in detail.

“I find the XP estimation would be better. It's down to more detail, it's down to the actual day ... is actually taking them to a level that's typically not going to in the waterfall approach.” (The people manager)

In the case of Team B, planning is an important activity at the beginning of the project and is time consuming. The plan covers the whole lifespan of the project. The work schedule as a result is considered the most important document by the project manager and meant to be followed. It has been only reviewed twice during the 12-month project.

“The schedule is kind of like, if we were driving, for me to be the map, and if we start to go out of the map, we have to figure out how to get back on to the map ... the schedule would be what I view as a kind of directive, so where we are going.” (Team B)

Table 2 summarizes the key characteristics of the development processes of the two teams according to the dimensions of Figure 1.

| Agile manifesto | Team A | Team B |
|------------------------|---|---|
| Development process | Team members consider they have a formal process. | Team members are not clearly aware of the process they are using. |
| | The formality comes from the routines both in the iteration cycle (2-week iteration, iteration planning meeting, retrospective, test first coding pattern) and at daily level (stand-up meeting and daily build). | The formality mainly comes from the documentation sign-off, no built-in routines. Team members have negative opinion on routines. |
| | Team members periodically review the process. | No retrospective regarding the process. |
| | The process becomes internalised in the minds of the team members. | The process is imposed by the management and remains external to the team members. |
| Documentation | No heavy documentation – the user story and associated acceptance tests are in the place of requirement documents, unit tests in the place of system analysis and design documents. | Documentation is mandated and prescribed – it plays the role of a token for negotiation of process gateways. |
| Customer relationships | User story is business driven and considered as a token of conversation between the customer and the developers rather than a binding contract. | Signed-off user requirements are the contract between developers and user. |
| Planning | Planning is frequent and fluid: <ul style="list-style-type: none"> • Release planning • Iteration planning • Daily planning at stand-up meeting | Planning is upfront and covers a long period, up to 12 months. Updates to the plan are rare and infrequent. |
| | Planning at a fine level of granularity (user stories fit to a 2-week iteration, no story bigger than 4 perfect engineering days), therefore better at estimation. | Planning at a coarse level of granularity (no fixed unit of timeframe is used for estimating tasks) . |
| | Plans are reviewed regularly and their execution is tracked for the purpose of improving planning in the next round. | The plan is directive. No regular review is made of the plan. |

Table 2. *The comparison of the two software development processes*

5 DISCUSSION

The edge of chaos concept (Figure 1) implies that a truly agile process is neither chaotic nor static. It needs to be structured but not so much that order prevails and innovation is stifled. A well-structured process can provide stability to a software development team. Agile practices such as time-boxed iterations and daily stand-up meetings provide a team with regular heartbeats. Meanwhile a process needs continuous adjustment and adaptation to the specific situations a team is embedded in to avoid process rigidity and deterioration. The regular retrospectives by Team A on their process allow them to take gradual steps to change and improve it, rather than leaving it to a stage where no effective action can be taken. An agile process is a delicate balance of structuring and restructuring which enables a software development team to work at a fast yet sustainable pace.

In terms of Figure 1, Team A seems to be following an agile methodology and achieving a balance between chaos and order, thus providing the enabling conditions for poising at the edge of chaos. By

contrast, Team B is at one level over-structured and might be expected to fall into the order/bureaucratic region. But, at another level there is a disconnect between what Team B say they do and what they actually do. This gap may lead to them tipping into the chaotic region, i.e., the espoused structures adopted by Team B may become ineffectual with the result that the project has insufficient structure. Our conceptual model suggests that it is not just agile methods that are at risk of tipping into chaos - it is also possible that the waterfall approach to software development may well turn out, in practice, to be chaotic.

The edge of chaos metaphor also implies that, while it is possible to work on the fine details of the short-term patterns of a system, its long-term behaviours are unpredictable. In agile processes, short iterations provide a timeframe in which a team can work with certainty. This certainty makes short-term planning possible and meaningful. Since user requirements are typically broken down to a fine level of granularity in agile processes, it provides a good basis for accurate and detailed planning. However, there is a danger of becoming short-sighted while working on short-term iterations, that is, to ignore the potential problems posed by long-term uncertainty. Frequent planning and plan reviewing enable a team to embrace long-term uncertainty by constantly steering and adjusting the direction they head for. Frequent planning is also a natural consequence of frequent feedback loops in agile processes due to the intensive interactions between customers and development teams as well as among team members. The value of feedback would be lost if no appropriate planning action followed. Meanwhile, plan reviewing is a learning opportunity for a team to improve their ability to plan. Planning in agile processes is a “paradox of planning and not planning that unfolds as a practice required by settings in which large degrees of uncertainty and ambiguity are inevitable” (Baskerville 2006, p.115). Rather than upfront long-term planning, agile processes incorporate frequent and fluid planning which enables a team to work with short-term certainty while embracing long-term uncertainty.

In summary, agile processes are not at the opposite of structure and planning, rather, structure and planning are integral elements of agile processes. A truly agile process is able to balance order and chaos and to poise at the edge of chaos, in the region of emergent complexity. Therefore, a “chaordic” (Highsmith 2002, p. xxiii) - both order and chaos- perspective is needed for managing agile processes.

6 SUMMARY

The contribution of this paper has been to use the edge of chaos concept from Complex Adaptive Systems theory to investigate software development processes and provide a better understanding of the role played by structure and planning. To illustrate our points, the software development processes of two teams in a major IT company, one using XP, the other the waterfall process, have been presented and compared, and the issues regarding structure and planning in agile processes have been discussed using the edge of chaos and the agile manifesto as the theoretical lens. We have suggested that the waterfall method, although potentially over-structured and bureaucratic in the abstract, may, in practice, turn out to be chaotic. Conversely, it seems reasonable to assume that agile methods, when followed rigidly without reflection, could also become bureaucratic and over-ordered. Further work is needed to explore the patterns of organization and behaviour that contribute to agility, irrespective of the espoused methodology.

References

- Anderson, P. (1999). Complexity theory and Organization Science. *Organization Science: A Journal of the Institute of Management Sciences*, 10(3), 216-232.
- Augustine, S., Payne, B., Sencindiver, F. and S. Woodcock (2005). Agile Project Management: Steering from the Edges. *Communication of ACM*, 48(12), 85-89.
- Avison, D. E. and G. Fitzgerald (2003a). Where now for development methodologies? *Communications of the ACM*, 46(1), 78-82.

- Avison, D. and G. Fitzgerald (2003b). *Information Systems Development: Methodologies, Techniques and Tools*. 3rd Edition. McGraw-Hill, London.
- Basili, V. and J. Turner (1975). Iterative Enhancement: A Practical Technique for Software Development. *IEEE Trans. Software Eng.*, Dec. 1975, 390-396.
- Beck, K. and C. Andreas (2004). *Extreme Programming Explained: Embrace Change*. 2nd Edition. Addison Wesley Professional, Mass.
- Baskerville, R. L. (2006). Artful Planning. *European Journal of Information Systems*, 15(2), 113-115.
- Boehm, B. (2002). Get Ready For Agile Methods, With Care. *Computer*, 35(1), 64-69.
- Brooks, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *Proc. IFIP, IEEE CS Press*, 1987, 1069-1076; reprinted in *Computer*, Apr. 1987, 10-19.
- Brown, S. L. and K. M. Eisenhardt (1998). *Competing on the edge: strategy as structured chaos*. Harvard Business School Press, Boston.
- Gell-Mann, M. (1995). What Is Complexity? *Complexity*, 1(1), 16-19.
- Haeckel, S. (1999). *Adaptive Enterprise: Creating and Leading Sense-and-respond Organizations*. Harvard Business School Press, Boston.
- Highsmith, J. (2002). *Agile Software Development Ecosystems*. Addison-Wesley, Boston.
- Highsmith, J. A. (2000). *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House Publishing, New York.
- Kauffman, S. (1995). *At Home in the Universe: The Search for Laws of Self-Organization and Complexity*. Oxford University Press, New York.
- Larman, C. and V. R. Basili (2003). "Iterative and Incremental Development: A Brief History." *IEEE Computer*, 36(6), 47-56.
- McKelvey, B. (forthcoming). *MicroStrategy from MicroLeadership: Improving distributed intelligence via complexity science*, in A. Lewin and H. Volberda (eds.) *Mobilizing the Self-renewing Organization*. Palgrave Macmillan, New York.
- McMillan, E. (2004). *Complexity, Organizations and Change*. London, Routledge, Taylor & Francis Group.
- Meso, P. and R. Jain (2006). Agile Software Development: Adaptive Systems Principles And Best Practices. *Information Systems Management*, 23(3), 19-30.
- 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.
- Pettigrew, A., Whittington, R., Melin, L., Sanchez-Runde, C., van den Bosch, F., Ruigrok, W. and Numagami, T. (2003). *Innovative Forms of Organizing*. Sage Publications, London.
- Pressman, R. S. (1997). *Software Engineering: A Practitioner's Approach*, McGraw-Hill, Beijing.
- Rakitin, S. (2001). Manifesto Elicits Cynicism. *IEEE Computer*, 34(12), 4.
- Schach, S. R. (1998). *Software Engineering with JAVA*, McGraw-Hill, Beijing.
- Schwaber, K. (1996). *Controlled Chaos: Living on the Edge*. *American Programmer*, April 1996.
- Schwaber, K and A. Beedle (2002). *Agile Software Development with SCRUM*. Prentice-Hall, Upper Saddle River, NJ.
- Stacey, R. D. (2003). *Strategic Management and Organisational Dynamics: The Challenge of Complexity*. 4th Edition. Prentice-Hall, London.
- Stephens, M. and D. Rosenberg (2003). *Extreme Programming Refactored: The Case Against XP*. Apress, New York.
- Vidgen, R., and Wang, X., (2006). *Organizing for Agility: a Complex Adaptive Systems Perspective on Agile Software Development Process* In: *Proceedings of the 14th European Conference on Information Systems*, Göteborg, Sweden, June 12-14.
- Waldrop, M. M. (1994). *Complexity: The Emerging Science at the Edge of Chaos*. Penguin books, London.