COMPLEXITY, PROCESS AND AGILITY IN SMALL DEVELOPMENT TEAMS: AN EXPLORATORY CASE STUDY

Paul Ralph, Lancaster University, Lancaster, UK, paul@paulralph.name

Jose Eduardo Narros, Lancaster University, Lancaster, UK, jenarros@gmail.com

Abstract

The extensive prescriptive literature on software and information systems development methods routinely recommends more methodical, plan-driven approaches for more complex projects and more agile, adaptive approaches for less complex projects. This paper presents a revelatory case study in which a team with no imposed method successfully used a more methodical, plan-driven approach for a simple project and a more adaptive, amethodical approach for a more complex project. Furthermore, the team explicitly and intentionally transitioned a less methodical, more adaptive process to cope with the increasing complexity of the second project. This pattern directly contradicts the dominant narrative advocated in methods literature. The paper adopts the theory of complex adaptive systems to analyze and understand the observed pattern and deconstruct the dominant narrative.

Keywords: Software Engineering, Information Systems Development, Agility, Complex Adaptive Systems.

1 INTRODUCTION

Information System Development (ISD) and Software Engineering (SE) projects are fraught with high rates of abandonment (Ewusi-Mensah, 2003), failure (Standish Group, 2009) and effort overrun (Molokken & Jorgensen, 2003). Infamous examples include the US Federal Aviation Administration abandoning development of a new air traffic control system in 1994 after spending US$2.4 billion (Charette, 2005); the Taurus project, which was cancelled after more than US$600 million in investment (Drummond, 1996); UK supermarket chain Sainsbury's write off of a defective US$526 million supply chain management system (Charette, 2005) and the 2010 termination of the UK FiReControl system "with none of the original objectives achieved and a minimum of £469m being wasted" (BBC News, 2011).

Proponents of plan-driven methods, e.g., the Unified Process (Jacobson et al. 1999), have long argued that better planning and more mature methods increase the probability of project success (Herbsleb & Goldenson, 1996). Meanwhile, proponents of Agile methods, e.g., Scrum (Schwaber & Sutherland, 2010), contend that emphasizing working code (rather than documentation) and flexibility under uncertainty will reduce software project failure rates (Beck, 2005). However, neither side has produced compelling evidence of the superiority of their approach in part due to the serious methodological challenges of comparatively testing methods (Brooks, 2010; Dyba & Dingsøyr, 2008).

More fundamentally, however, *agility* is not well-understood and different methods seem to use inconsistent and sometimes conflicting definitions of agility (Conboy, 2009). Moreover, development practice often differs from any known method (Fitzgerald, 1998; Fitzgerald et al., 2002; Truex et al., 2000; Zheng et al., 2011), leading to a distinction between a team's method (prescribed actions) and process (actual actions) (Mathiassen & Purao, 2002). Furthermore, it is not clear how Agile methods and processes based on those methods manifest agility, how these processes interact with project complexity, or how these interactions affect risk. Therefore, this paper explores the relationship between complexity, process and agility beginning with the following research question.

> ***Research Question:*** *What is the relationship between complexity, process and agility in small, autonomous software development teams.*

Here, *complexity* refers to the extent to which the behavior of a technical or social system is emergent, i.e., cannot be accurately predicted from its components (Anderson, 1999; Gell-Mann, 1999; Holland, 1992). While a *method* is a set of prescriptions regarding how to create or modify a software system, a *process* is a set of activities actually enacted by the development team to create or modify a software system. A (development) *team* is a group of persons cooperating to create or modify a software system. A team's *agility* denotes its capacity to react quickly to change. To avoid confusion between the two meanings of Agile, we use uppercase-A *Agile* to refer to the Agile methods philosophy (Beck, 2005) and lowercase-a *agility* to refer to the property of a team. Both agility and complexity are spectra, i.e., teams and systems have varying levels of agility and complexity, respectively. A team is *autonomous* if it is in control of its process, i.e., autonomy implies that management does not impose substantive constraints on the team's methods, practices, tools or process.

We chose to focus on small teams due to the asymmetric relationship between team size and agility. Generally, the larger the team, the more difficult responsiveness becomes. However, while the best large teams cannot be as responsive as the best small teams due to their increased coordination overhead, the worst small teams can be just as unresponsive as the worst large teams by simply ignoring change. Furthermore, we chose to focus on autonomous teams not only as they are underrepresented in the literature (Truex et al., 2000) but also as we were interested in how teams alter their processes over time – a phenomenon often hindered by managerial constraints.

This paper continues with a summary of existing literature on methods as a construct, agility and Agile methods (§2). Sections three and four describe our exploratory case study methodology and three major findings, respectively. We then elaborate on the significance of and explanations for our most surprising finding (§5) and conclude the paper by summarizing its contributions and limitations (§6).

2 LITERATURE ON COMPLEXITY, PROCESS AND AGILITY

The history of methods comprises four eras (Avison & Fitzgerald, 2003). In the "pre-methodology era", developers trained in programming but not the "contexts of use" built software from a superficial understanding of users' needs and goals, leading to many failures. In the "early methodology era", development was divided into phases inspired by the Waterfall model. In the "methodology era" proper, practitioners began using the term "methodology" and many approaches emerged including structured programming, prototyping, object-oriented development and participative development. In the (current) "post-methodology era", many practitioners have rejected methodologies in general. For example, developers may omit elements of methods not out of ignorance but because they perceive them as irrelevant in their context (Fitzgerald, 1997). Possible explanations for methodology rejection include perceived ineffectiveness or poor usability of specific methods and perceived ineffectiveness of greater methodicalness in principle.

While the transition to the post-methodology era necessitates post-methodology thinking, Truex et al. (2000) argue that the concept of method occupies such a privileged status that development and methods are "completely merged in the systems development literature" (p. 56). This is evident in contemporary studies of agility (Marchenko & Abrahamsson, 2008; Moe et al., 2010; Pikkarainen & Wang, 2011). This historic entanglement of empirical research with methods research impedes uptake of a post-methodology paradigm. Therefore, we proceed by attempting to untangle process, complexity and agility from methods.

Methods prescribe coping with complexity by creating order (Baskerville et al., 1992) and structure (Truex et al., 2000), facilitating communication (Goguen, 1992) and planning, and allowing an organization in constant transition to freeze its assumptions so that it can proceed (Baskerville et al., 1992). Perhaps the most salient theme in the methods literature is the distinction between plan-driven and Agile methods (Boehm & Turner, 2003).

Plan-driven methods, including Waterfall (Royce, 1970), Spiral (Boehm, 1988) and the Unified Process (Jacobson et al. 1999), assume that the goals and desiderata are knowable and relatively stable (Berry, 2004). Some evidence (Herbsleb et al., 1997) found that higher process maturity (i.e., plan-driven methods) is associated with higher product quality, customer satisfaction, productivity and morale. However, plan-driven methods have been criticized for being too rigid, technology oriented, unfeasible and inconsistent with real-world practice (Beck, 2005; Brooks, 2010; Ralph, 2011).

In contrast, Agile methods, including Rapid Application Development (Martin, 1991), Feature Driven Development (Palmer & Felsing, 2002), Crystal Clear (Cockburn, 2004), Adaptive Software Development (Highsmith & Orr, 2000), Extreme Programming (Beck, 2005) and Scrum (Schwaber, 2004), assume that developers' ability to predict goals and desiderata up front is limited (Berry, 2004). Agile methods are associated with lower failure rates (Ambler, 2010; Standish Group, 2009), modest gains in productivity (Cardozo et al., 2010) and increased software quality (Laurie, 2003; McDowell et al., 2006). Positive attitudes toward Agile methods have spread beyond the academic and software communities – the UK government's 2011 Information and Communications Technology strategy claims that lean and agile methodologies will be used to "... reduce waste, be more responsive to changing requirements and reduce the risk of project failure" (Cabinet Office 2011). A method's suitability hence depends on the validity of the method's assumptions about the environment (Turk et al., 2005). However, Agile methods have been criticized for depending on excessive refactoring and extensive tacit knowledge (Babar, 2009; Boehm, 2002; Petersen & Wohlin, 2009).

More fundamentally, methodicalness itself also exhibits limitations. Where development occurs in an unpredictable environment, methods may obstruct project goals and reduce agility (Baskerville et al., 1992). Methods embed images of their environments (Baskerville et al., 1992) and therefore can yield different results in different settings (Baskerville et al., 1992; Naur, 1993; J. A. Turner, 1987). Individuals may even fake adherence to methods (Bansler & Bødker, 1993; Parnas & Clements, 1986). Different teams not only interpret methods differently but also manifest agility differently (O'hEocha et al., 2010). Perhaps more fundamentally, the usual way of building systems may be innately amethodical, i.e., characterized by "management and orchestration of systems development without a predefined sequence, control, rationality, or claims to universality" (Truex et al., 2000). Amethodical

development is part of a wider conflict between software development methods and theories (cf. Ralph 2010; 2011, 2012; 2013a; 2013b; 2013c; Ralph et al. 2013).

Recognizing that teams may adopt, adapt or avoid methods, researchers have introduced concepts including "method-in-action" (Fitzgerald, 1997), methods-in-use (Mathiassen & Purao, 2002) and "emergent method" (Fitzgerald et al., 2002; Madsen et al., 2006) to denote the activities in which developers engage in practice. These result from the adaptation of both the actors and the context (Offenbeek & Koopman, 1996) at multiple levels, e.g., project, organization and industry (Fitzgerald et al., 2003). Here we refer to practitioners' real-world activities as their *process*.

Furthermore, *agility* is a property which emerges in a particular setting and not inherent to any ISD practice – a practice which increases agility in one setting may not do so in another (Conboy, 2009). The practices within development teams are shaped by the interplay of the characteristics of the setting, the actors and their interactions (Madsen et al., 2006). In other words, although methods may affect agility, agility's proximate cause is the actions of the agents. Therefore, Agile methods are better understood as methods aimed at facilitating the emergence of agility in particular environments. To what extent a particular Agile method improves agility in practice is an empirical question.

Finally, *complexity* has often been associated with an increase in the risk of IS failure (Highsmith & Orr, 2000). Interestingly, both proponents and detractors of Agile methods argue that they are unsuitable for more complex projects (Beck, 2005; Turk et al., 2005), whereas plan-driven methods including the Unified Process are explicitly intended for complex projects (Jacobson et al. 1999). Although far from unanimous, there is sense in the Agile community that Agile methods do not scale to projects that require hundreds developers to write millions of lines of code over several years. Increasing complexity is thought to necessitate more coordination and more therefore more plan-driven methods (Brooks, 2010).

3 RESEARCH METHOD

Our objective was to examine the relationships between complexity, process and agility in a development project with no imposed method. Therefore, we adopted an interpretive case study approach (Eisenhardt, 1989) within a critical realist ontology (i.e., as our perception of the external world is flawed, acquiring knowledge requires critical reflection). Our approach is interpretive in that we studied the projects and processes through the meanings ascribed by participants in interviews and observations, which are socially constructed (Myers & Avison, 2002). We focused on a single case as teams without imposed methods are rare and under-studied, the following is a "revelatory case" – one of Yin's justifications for single-case designs (Yin, 2003). Moreover, "requiring case study research to involve multiple sites or multiple cases for the sake of substantiating a theory … presumes … The Fallacy of Affirming the Consequent" (Lee & Hubona, 2009). We broadly followed the recommendations of Eisenhardt (1989) and Dube and Pare (2003) in executing the study and analyzing the results.

The research took place at an Internet services company of between 40 and 50 employees in England. The company provides services including online marketing and website development. Rather than discrete project teams, the company operates as a hybrid hub-and-spokes network – each project is assigned to a manager (hub) who assign tasks to whoever has the necessary expertise (spokes), such that each developer's time is usually split between several projects. Some developers are explicitly assigned to a "large projects group" or a "small projects group", but this distinction is fuzzy in practice.

Consequently, delineating the *team* to study was challenging. Based on discussion with senior management, we identified a pair of projects (denoted Project A and Project B below) with two desirable characteristics. First, as they differed significantly in nature and scope, they were likely to produce contrasting data, which mitigates cherry-picking of observations (Yin, 2003). Second, they involved substantially overlapping teams, reducing various confounding factors related to participants. Management did not impose, formally or informally any methods or development practices on either project.

Data collection occurred over a six-week period in June and July, 2011, and included semistructured interviews (audio recorded), documents (e.g., sales proposals, technical specifications, emails) and direct observation of meetings, development activities, management activities. Interviews covered topics including the background of the company and the interviewee, the current and past activities of the team, the relationships between teams activities and other projects, critical incidents (Flanagan, 1954), perception of suitability of current processes and interviewee's knowledge of ISD methods.

Interview transcripts, observation notes, reflection notes and documents were compiled and analysed using open coding (Miles & Huberman, 1994) to identify potential categories followed by key themes and critical aspects (Eisenhardt, 1989). Process agility was evaluated qualitatively based on the team's ability to support frequent feedback and adaptation to scope changes, this being a key manifestation of agility (Conboy, 2009; Conboy & Wang, 2009). The diversity of data sources (document, interviews and observations) permitted data triangulation, thereby increasing accuracy and reliability (Yin, 2003). The main findings were discussed with the participants to verify that we had correctly understood their perspectives.

4 ANALYSIS AND RESULTS

We begin our analysis with a description of the case study context. We then discuss three major themes produced by our analysis: 1) *adaptation*, 2) *agility as a response to complexity*, and 3) *process pragmatism*.

4.1 Case Context

Projects A and B varied significantly in size, client involvement and novelty (Table 1). However, both projects were assigned to the same account manager and shared significantly overlapping teams (Table 2). Project A was a website for a manufacturing company consisting of primarily static information regarding the company and its products. Project B was a consumer e-commerce website comprising a product catalog, database, shopping cart, user accounts, customer services, administration dashboard, some static content and a nontrivial feature where shoppers could virtually try on products. Both projects were commissioned by local clients. Both the team and client for Project A felt that the project was reasonably successful. While Project B remained in development at the end of the study, both the team and the client appeared reasonably satisfied with its progress.

	Project A	Project B
Estimated length	Three months – June to August 2011.	Unknown – the project started in November 2009 and continues with no end in sight.
Client involvement	Low – two initial face-to-face meetings followed by mostly email communication initiated from the team.	High – the client participates in decision making and contacts the team up to several times a week.
Similarity to previous projects	High – the team is confident about the work required and trusts that it will be completed on schedule / budget.	Low – it is considered one of the team's most difficult projects ever; e.g., it uses effort-based costing as the team cannot team provide reliable estimates.

TABLE 1. Summary of Main Differences Between Projects

4.2 Theme 1: Adaptation

The team did not use any formalized method; rather, it adapted its process to each project. Although no critical process breakdowns occurred, participants perceived the differing approaches used across projects and lack of documentation to indicate a need for greater process structure and transparency. However, participants universally indicated that their current approach facilitated high flexibility and were concerned that adopting a formal method or tool could inhibit their ability to adapt to their environment, i.e., their agility. The organization was explicitly aware of the need to learn what works

and adapt accordingly; a feature crucial for the emergence of agility at the organizational level (Gräning & Wendler, 2011; Lyytinen & Rose, 2006; Vidgen & Xiaofeng, 2009). Their approach to progress tracking illustrates this aspect of their culture:

> *[To know the progress about a project] we would check regularly, D.D. and I would check and if there is a problem everyone knows that they should say it... I think if you have a system to do everything, you spend more time doing the system than you spend doing the work, it is just too much... There is room for improvement like when we need it we have introduced a document -D.W.*

Code	Role	Project A	Project B
R.M.	Account Manager	From start	From start
D.W.	Graphical Designer, Director	From start	From start
M.W.	Lead Developer	Not involved	Began mid-2010
D.R.	Lead Developer	From start	Not Involved
D.D.	Technical Director	From start; Supervising D.R.	From start; Supervising M.W.
M.C.	Front-end developer	From start	From start

TABLE 2. Summary of Participant Codes and Roles

Strong adaptive properties seem to underly the team's activities, facilitating self-organization. This leads to an emergent approach to ISD combining methodical, amethodical, plan-driven and adaptive elements (Theme 3), although the team would not articulate it in these terms.

4.3 Theme 2: Agility as a Response to Complexity

A common narrative in methods literature (above) is that the greater a project's complexity, the greater the need for formal methods to facilitate coordination, planning and risk mitigation. This case revealed the opposite pattern.

First, in Project A, where perceived size and complexity were low, the team employed a predominately plan-driven, document-centric, linear approach. In contrast, in Project B, where perceived size and complexity were high, the team employed a predominately adaptive, iterative approach (Fig. 1).

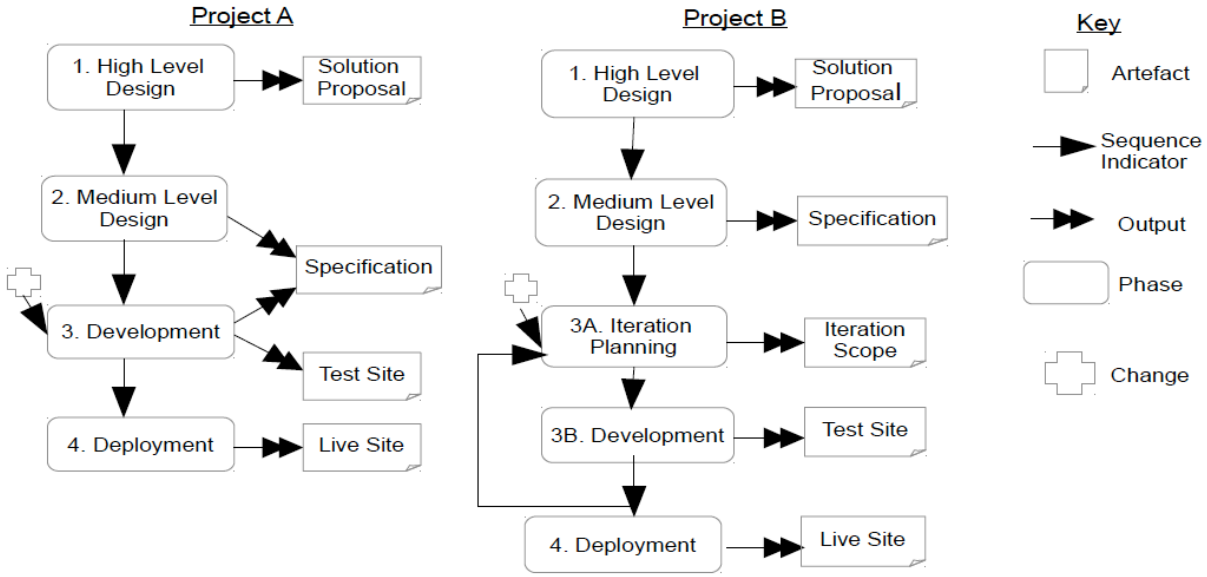

Figure 1. Process models for Projects A and B

However, Project B did not begin with an adaptive or iterative approach; rather it began with a proposal, a written requirements specification and a fixed-price/-schedule contract. However, through primarily email) communication with the client, the project's aims and scope evolved rapidly.

> *I tried to ... have it all itemized, etc., but it quickly went out of the window because of the volume of changes -R.M.*

This prompted a breakdown where, due to the volume of changes requested by the client, the team was unable to meet the client's expectations within the agreed budget.

> *We got to a point where we had to set up a meeting and we told [the client] that the emails were too much, they were sending too much information, it was too hard to get it done -R.M.*

Recognizing the project's complexity, the team transitioned to a more adaptive, amethodical approach, including replacing specifications with incremental development in 3-4 week iterations and a shorter feedback cycle with monthly meetings to review the current iteration and agree scope for the next. Furthermore, they shifted from fixed cost model to a time and resources model where the client was billed for the time it took to complete the agreed scope. The team augmented email communication with monthly face-to-face client meetings. In addition, they replaced the written specification with a collection of paragraph-length change descriptions, agreed via email and supported by phone conversations. By then, the teams' estimate of Project B's complexity was much higher.

> *This is a very complex site and ... it is a very complicated site, it has to integrate with other systems and sometimes it is not clear if it is our fault or the customer's fault ... And it is kind of a nightmare really -R.M.*

As the difficulty of up-front design became evident, the team gave the client a more participatory role and began releasing an updated site to the client for verification after each iteration.

> *Usually what happens is that I will show [the client] as I finish each of the items in the list, so will say look I have done this today so you can take a look -M.W.*

This revised process presents many elements common in Agile methods, including short incremental iterations, frequent customer feedback and prioritization of functionality based on the value provided to the customer. However, common drawbacks of Agile methods were also evident – as the technical specification became obsolete, the ability to continue progressing with the project depended heavily on the lead developer.

> *Another problem with this is that ... it is all in M.W.'s head at the moment, he knows all about this, it would take a long time for another person to get their head around this site if M.W. weren't here. -R.M.*

This supports the criticism that agility demands extensive tacit knowledge (Boehm, 2002; Turk et al., 2005). Both the client and the team heavily depended on the individuals involved in the project as there was minimal information about the site other than the code itself. The team also indicated that the need for constant refactoring – characteristic of Agile approaches (Babar, 2009; Berry, 2004; Boehm, 2002) – was problematic as including refactoring costs seemed unreasonable to the client.

In summary, as Project B unfolded, the teams' perception of its complexity increased. To manage the increased complexity, the team sought to increase their agility by reducing their reliance on documents and planning and adopted practices aimed at increasing their agility, including rapid feedback cycles and iterative development. This occurred without any external consultants or internal leaders championing Agile approaches.

4.4 Theme 3: Process Pragmatism

Design literature may be seen as bifurcated, i.e., characterized by two incommensurable paradigms (Dorst & Dijkhuis, 1995; e.g. Ralph, 2011). In this view, one can approach software development in a methodical, plan-driven way or in an amethodical, adaptive way, while combining elements of both is impractical. However, we observed the team combining practices generally associated with both ends of the methodical/amethodical and plan-driven/adaptive spectra (Table 3).

	Meaning	Examples (all from Project B)
methodical	Describes a process having an orderly and predictable sequence of activities	Development was divided into 3-4 week iterations, each of which included planning and final review meetings, functionality testing by the client, and final client approval. Each approved feature was deployed with the next planned release.
amethodical	Describes a process having an unpredictable sequence of activities	Iterations were characterized by ad hoc selection of work and unplanned conversations between the team and the client where design details were agreed.
plan-driven	Describes a process where action and progress are understood through plans, assuming a predictable environment	The team had weekly discussions to determine whether the iteration was proceeding on schedule by comparing the work done to the scope agreed during the iteration planning meeting.
adaptive	Describes a process where action is improvised a progress is understood in terms of goals, assuming an unpredictable environment	Developers often encountered situations requiring unexpected low level design decisions and responded by exploring options and improvising. Both unplanned discussions and routine monthly meetings often identified concerns over development techniques and functionality, triggering further exploration.

TABLE 3. Contemporary Examples of methodical/amethodical and plan-driven/adaptive dimensions

5 DISCUSSION

Methods literature consistently suggests that iterative, amethodical and Agile approaches are more appropriate for less complex projects while linear, methodical and plan-driven approaches are more appropriate for more complex projects. However, our analysis revealed that, left to themselves, the software development team responded to a perceived increase in project complexity by making their process more iterative, amethodical and Agile. Understanding this behavior requires a different theoretical lens. Consequently, we attempt to explicate our findings using Complex Adaptive Systems theory (CAS).

5.1 Complex Adaptive Systems

CAS posits that some systems exhibit properties or behaviors that are emergent, i.e., not evident from their components because of the components' non-linear interactions (Gell-Mann, 1999; Holland, 1992). Although CAS has no established definition, it entails constructs including interdependent autonomous agents, self-organizing networks and coevolution of problem and solution spaces (Anderson, 1999; Bak & Bak, 1996; Kauffman, 1995). Complexity is a spectrum – the greater the proportion of a system's behavior that are emergent, the more complex the system.

Philosophically, CAS is a response to reductionism. Using a reductionist approach, a system is understood by examining its components and their relationships. Using a CAS approach, in contrast, a system is understood as a whole, which has properties not evident from its components and their relationships.

Many authors have suggested that complexity is inherent to ISD and SE (Highsmith & Orr, 2000; Kennaley, 2010; Meso & Jain, 2006; Vidgen & Xiaofeng, 2009) and suggest that ISD is consistent with CAS regarding order emerging from the interactions of self-organizing individuals (Khoshroo & Rashidi, 2009; Meso & Jain, 2006; Truex et al., 1999; Vidgen & Xiaofeng, 2009). CAS has been used to examine Agile practices (Meso & Jain, 2006), study agility manifestation (Conboy, 2009), show that rigidly following either plan-driven or Agile methods can diminish adaptation (Wang & Vidgen, 2007), and propose a shift from linear plan-build-revise cycles to non-linear speculate-collaborate-learn concurrent activities (Highsmith & Orr, 2000).

5.2 Understanding Process Adaptation with CAS

ISD occurs within cultural, technical and organizational settings that are inherently complex to some extent. Human beings, conflicting goals, ambiguous problems, changing environments, changing technologies and unpredictable clients all contribute to ISD project complexity.

From a CAS perspective, the usefulness of planning, up-front design and modeling is inversely related to system complexity. As highly complex systems are predominately unpredictable, plans either suffer from an exponential explosion of contingencies or rapidly grow stale. As the designer manipulates fields she does not fully understand, the quality of a design is not evident until after implementation, converting design from logical deduction to guess and check. As the environment is both dynamic and poorly understood due to the proliferation of non-linear interactions, conceptual models and requirements are originally flawed and grow rapidly stale.

A CAS lens, therefore, provides starkly different predictions and prescriptions for ISD practice. If project participants perceive their environment as well-understood and controllable, methodical and plan-driven approaches are more efficient as they dispense with the overhead of iterations without substantially increasing risk. This plan-driven approach may contain contingencies but it assumes that the environment is largely controllable and predictable. Unexpected or unplanned elements can therefore be managed within the *a priori* planning framework.

In contrast, dealing with complexity therefore demands continuous adaptation to new, emergent conditions. If project participants perceive their environment as unstable and unpredictable, the effectiveness of upfront planning, analysis and design is greatly reduced. Meanwhile, the additional overhead of iterative development is warranted by its mediating effect on project risk. Adaptive approaches may facilitate managing uncontrollable aspects of the environment through improvisation.

In summary, examining development projects through a CAS lens produces predictions opposite to those of a methods lens. CAS suggests that as perceived complexity increases, teams should adopt more amethodical, adaptive approaches and seek greater agility; while when perceived complexity decreases, teams should adopt more methodical, plan-driven approaches and seek greater efficiency. Our observations therefore are consistent with a CAS perspective.

5.3 Implications for Research and Practice

Combining existing literature on both methods and CAS with our observations and analysis produces several implications for practice and research. For practitioners, our analysis suggests a substantially different view of development approaches than managers are likely to experience in business undergraduate and MBA programs. Contemporary textbooks (e.g. Kroenke et al., 2010; Laudon et al., 2009) present plan-driven approaches as appropriate to larger projects and Agile approaches as appropriate to smaller projects. The above analysis, however, points to a more complicated view (Table 4). For small, simple projects a linear approach is workable as risk is low, goals and requirements are clear and the domain is well-understood. Meanwhile, large, simple projects require the substantial coordination mechanisms embedded in heavyweight plan-driven approaches including the Unified Process. However, the unpredictability of complex projects undermines the efficiency and increases the risk of linear and plan-driven approaches. Agile methods including Scrum are intended to manage complexity by maximizing responsiveness. However, it remains unclear how to approach projects too large for existing Agile methods and too complex for plan-driven approaches.

	Predictable	Complex
Small	Simple Linear (e.g., Waterfall)	Agile Methods (e.g., Scrum)
Large	Plan Driven Methods (e.g., Unified Process)	?

TABLE 4. Methods by Project Size and Complexity

However, this view may still be too simple as complexity itself may be emergent, i.e., projects that appear simple at first may become or appear complex later. Our findings suggest that introducing

agility-enhancing practices may help teams cope with increasing complexity in development projects. Nevertheless, practitioners should know that not all Agile practices necessarily enhance team agility in all contexts.

For researchers, our findings suggest that studying complexity in ISD may uncover insights that are obscured by the reductionist lens of methods. Similarly, studying processes as combinations of planned, adaptive, methodical and amethodical aspects, rather than either-or approaches, may produce richer understandings of practice. Furthermore, our results highlight how studying autonomous teams may uncover concepts and patterns that are normally obscured by managerial constraints.

6 CONCLUSION

This paper reports the results of an exploratory case study of a small, autonomous web development team. It began with the research question, *What is the relationship between complexity, process and agility in small, autonomous software development teams?* The answer and primary contribution of the paper is that, in this case, developers reacted to increasing complexity by modifying their process to emphasize agility and responsiveness. This finding is contrary to major themes in methods literature, which suggest that greater complexity demands more formal methodology, planning and documentation. Consequently, we found that Complex Adaptive System theory had greater explanatory power in this context than a theoretical lens based on methods literature. In this view, complexity fundamentally undermines the value of planning and methodology. Plan-driven methods assume stability; complexity undermines stability; therefore increased complexity demands increased agility rather than better planning.

The study produced two other interesting findings. The team readily adapted its process to the conditions of different projects, rather than repeatedly applying a similar method. Furthermore, as plan-driven and Agile approaches are part of separate, predominately incompatible complexes of methods, practices, assumptions, theories and paradigms (Brooks, 2010; Ralph, 2011), we would expect teams to use one or the other. However, regardless of their predominate orientation (plan-driven or adaptive) the team blended techniques usually associated with methodical, plan-driven methods with techniques usually associated with amethodical, adaptive methods.

This study manifests several limitations. As with most case studies, the results are not statistically generalizable across any particular industry or group. However, the results are theoretically generalizable (Lee & Baskerville, 2003; Yin, 2003) in that our observations support specific theoretical conjectures. For example, they illustrate the potential usefulness of analyzing design teams using a CAS lens in addition to a methods lens. Furthermore, this is a revelatory case (Yin, 2003) in that few studies have investigated software development teams with the studied teams' freedom to alter its process at will. Second, space constraints limit us to a summary of our data collection and analysis methods, rather than a comprehensive account. Third, as with most interpretive studies, it is not possible to definitively establish causality; for example, we cannot say with certainty how the teams' responding to increased complexity by increasing its agility affected project risk or success – only that the participants believed that the project was proceeding well and that increasing agility had helped.

Further research is clearly needed to better understand the complexity-process-agility relationship, including – 1) how it varies across settings, e.g., in larger and distributed projects where the suitability of agile methods is disputed (Babar, 2009; Hossain et al., 2009; Petersen & Wohlin, 2009); 2) how methodical/amethodical and plan-driven/adaptive elements interact in different settings; and 3) how different methods and practices can be combined and adapted to particular settings to balance complexity with determinism in development projects (Kennaley, 2010). Following this research, we believe that complexity is a key variable in understanding methods, practices, productivity and success in software engineering.

References

Ambler, S. (2010). 2010 Agile Project Success Rates Survey Results. Ambysoft. Retrieved July 2, 2012, from http://www.ambysoft.com/surveys/agileSuccess2010.html

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

Avison, D., & Fitzgerald, G. (2003). Where Now for Development Methodologies. Communications of the ACM, 46(1), 79–82.

Babar, M. A. (2009). An exploratory study of architectural practices and challenges in using agile software development approaches (pp. 81–90).

Bak, P., & Bak, P. (1996). How nature works: the science of self-organized criticality. New York, NY, USA: Copernicus Press.

Bansler, J., & Bødker, K. (1993). A Reappraisal of Structured Analysis: Design in an Organizational Context. ACM Transactions on Information Systems, 11(2), 165–193.

Baskerville, R., Travis, J., & Truex, D. P. (1992). Systems Without Method: The Impact of New Technologies on Information Systems Development Projects (pp. 241–269). Presented at the IFIP WG8.2 Working Conference on The Impact of Computer Supported Technologies in Information Systems Development, Amsterdam, The Netherlands: North-Holland Publishing Co.

BBC News. (2011). Failed fire project wasted £469m, says committee of MPs. Retrieved July 2, 2012, from http://www.bbc.co.uk/news/uk-14974552

Beck, K. (2005). Extreme Programming eXplained: Embrace Change. Boston, MA, USA: Addison Wesley.

Berry, D. M. (2004). The inevitable pain of software development: Why there is no silver bullet. In Radical Innovations of Software and Systems Engineering in the Future, Springer LNCS 2941, 50–74.

Boehm, B. (1988). A spiral model of software development and enhancement. IEEE Computer, 21(5), 61–72.

Boehm, B. (2002). Get ready for agile methods, with care, IEEE Computer, 35(1), 64–69.

Boehm, B., & Turner, R. (2003). Using risk to balance agile and plan-driven methods. IEEE Computer, 36(6), 57–66.

Brooks, F. P. (2010). The Design of Design: Essays from a Computer Scientist. Addison-Wesley Professional.

Cabinet Office, UK Government (2011). Government ICT Strategy, Retrieved January 29, 2013, from http://www.cabinetoffice.gov.uk/content/government-ict-strategy

Cardozo, E., Neto, J., Barza, A., França, A., & da Silva, F. (2010). SCRUM and Productivity in Software Projects: A Systematic Literature Review. Presented at the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE).

Charette, R. N. (2005). Why Software Fails. IEEE Spectrum 42(9).

Cockburn, A. (2004). Crystal Clear: A Human-Powered Methodology for Small Teams. Addison Wesley.

Conboy, K. (2009). Agility from First Principles: Reconstructing the Concept of Agility in Information Systems Development. Information Systems Research, 20(3), 329–354.

Conboy, K., & Wang, X. (2009). Understanding agility in software development through a complex adaptive systems perspective. Presented at the European Conference on Information Systems.

Dorst, K., & Dijkhuis, J. (1995). Comparing Paradigms for Describing Design Activity. Design Studies, 16(2), 261–274.

Drummond, H. (1996). The politics of risk: trials and tribulations of the Taurus project, Journal of Information Technology 11(4), 347–357.

Dube, L., & Pare, G. (2003). Rigor in Information Systems Positivist Case Research: Current Practices, Trends and Recommendations. MIS Quarterly, 27(4), 597–635.

Dyba, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. Information and Software Technology, 50(9-10), 833–859.

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

Ewusi-Mensah, K. (2003). Software Development Failures. Cambridge, MA, USA: MIT Press.

Fitzgerald, B. (1997). The use of systems development methodologies in practice: a field study. Information Systems Journal, 7(3), 201–212.
Fitzgerald, B. (1998). An empirical investigation into the adoption of systems development methodologies. Information & Management, 34(6), 317–328.
Fitzgerald, B., Russo, N. L., & O'Kane, T. (2003). Software development method tailoring at Motorola. Communications of the ACM, 46(4), 64–70.
Fitzgerald, B., Russo, N. L., & Stolterman, E. (2002). Information systems development: methods in action. McGraw-Hill.
Flanagan, J. C. (1954). The critical incident technique. Psychological Bulletin, 51(4).
Gell-Mann, M. (1999). Complex adaptive systems. In Complexity: Metaphors, models and reality (pp. 17–45). Westview Press.
Goguen, J. A. (1992). The Dry and the Wet. North-Holland Publishing Co.
Gräning, A., & Wendler, R. (2011). How Agile Are You Thinking? – An Exploratory Case Study. Wirtschaftinformatik Proceedings 2011. Paper 33.
Herbsleb, J., & Goldenson, D. (1996). A systematic survey of CMM experience and results (pp. 323–330). Presented at the 18th International Conference on Software Engineering.
Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., & Paulk, M. (1997). Software quality and the Capability Maturity Model. Communications of the ACM, 40(6), 30–40.
Highsmith, J. A., & Orr, K. (2000). Adaptive software development: a collaborative approach to managing complex systems. New York, NY, USA: Dorset House Pub.
Holland, J. H. (1992). Complex Adaptive Systems. Daedalus, 121(1), 17–30.
Hossain, E., Babar, M. A., & Paik, H. (2009). Using Scrum in global software development: A systematic literature review, In Proceedings of the Fourth IEEE International Conference on Global Software Engineering (pp. 175–184).
Jacobson, I., Booch, G., & Rumbaugh, J. (1999a). The Unified Software Development Process. Boston, Mass: Addison-Wesley.
Kauffman, S. A. (1995). At home in the universe: The search for laws of self-organization and complexity. USA: Oxford University Press.
Kennaley, M. (2010). SDLC 3.0: Beyond a Tacit Understanding of Agile. Fourth Medium Press.
Khoshroo, B., & Rashidi, H. (2009). Towards a Framework for Agile Management Based on Chaos and Complex System Theories, In Proceedings of the 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 291–292.
Kroenke, D., Gemino, A., & Tingling, P. (2010). Experiencing MIS (Second Canadian ed.). Toronto: Pearson Prentice Hall.
Laudon, K., Laudon, J., & Brabston, M. (2009). Management Information Systems: Managing the Digital Firm (Fourth Canadian ed.). Toronto: Pearson, Prentice Hall.
Laurie, W. (2003). Test-Driven Development as a Defect-Reduction Practice. In E. M. Maximilien & V. Mladen (Eds.), Proceedings of the International Symposium on Software Reliability Engineering, Denver, CO, USA: IEEE.
Lee, A. S., & Baskerville, R. L. (2003). Generalizing Generalizability in Information Systems Research. Information Systems Research, 14(3), 221–243.
Lee, A. S., & Hubona, G. (2009). A Scientific Basis for Rigor and Relevance in Information Systems Research. MIS Quarterly, 33(2), 237–262.
Lyytinen, K., & Rose, G. M. (2006). Information system development agility as organizational learning, European Journal of Information Systems 15(2), 183–199.
Madsen, S., Kautz, K., & Vidgen, R. (2006). A framework for understanding how a unique and local IS development method emerges in practice. European Journal of Information Systems, 15(2), 225–238.
Marchenko, A., & Abrahamsson, P. (2008). Scrum in a multiproject environment: An ethnographically-inspired case study on the adoption challenges, Proceedings of Agile 2008 (pp. 15–26).
Martin, J. (1991). Rapid application development. Indianapolis, IN, USA: Macmillan Publishing.
Mathiassen, L., & Purao, S. (2002). Educating reflective systems developers. Information Systems Journal, 12(2), 81–102.
McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2006). Pair programming improves student retention, confidence, and program quality. Communications of the ACM, 49(8), 90–95.

Meso, P., & Jain, R. (2006). Agile software development: adaptive systems principles and best practices, Information Systems Management 23(3), 19–30.
Miles, M. B., & Huberman, A. M. (1994). Qualitative Data Analysis: An Expanded Sourcebook. Thousand Oaks, CA, USA: Sage Publications, Inc.
Moe, N. B., Dingsøyr, T., & Dyba, T. (2010). A teamwork model for understanding an agile team: A case study of a Scrum project, Information and Software Technology 52(5), 480–491.
Molokken, K., & Jorgensen, M. (2003). A review of software surveys on software effort estimation. In Proceedings of the International Symposium on Empirical Software Engineering, 223–230.
Myers, M. D., & Avison, D. E. (2002). Qualitative research in information systems: a reader. SAGE.
Naur, P. (1993). Understanding Turing's universal machine: personal style in program description. The Computer Journal, 36(4), 351–372.
Offenbeek, M., & Koopman, P. L. (1996). Scenarios for system development: matching context and strategy, Behaviour and Information Technology, 15(4), 250–265.
O'hEocha, C., Conboy, K., & Wang, X. (2010). So You Think You're Agile?, In Agile Processes in Software Engineering and Extreme Programming, Springer, LNBIP 48, 315–324.
Palmer, S. R., & Felsing, J. M. (2002). A Practical Guide to Feature Driven Development. Prentice Hall.
Parnas, D. L., & Clements, P. C. (1986). A rational design process: How and why to fake it. IEEE Transaction on Software Engineering, 12(2), 251–257.
Petersen, K., & Wohlin, C. (2009). A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case, Journal of Systems and Software 82(9), 1479–1490.
Pikkarainen, M., & Wang, X. (2011). An Investigation of Agility Issues in Scrum Teams Using Agility Indicators, In Information Systems Development: Asian Experiences, Springer, 449–459.
Ralph, P. (2010). Comparing Two Software Design Process Theories. In R. Winter, J. L. Zhao, & S. Aier (Eds.), (pp. 139–153). Proceedings of DESRIST, LNCS 6105, St. Gallen, Switzerland: Springer.
Ralph, P. (2011). Introducing an Empirical Model of Design. In Proceedings of The 6th Mediterranean Conference on Information Systems, Limassol, Cyprus.
Ralph, P. (2012). The Illusion of Requirements in Software Development. Requirements Engineering.
Ralph, P. (2013a). Possible Core Theories for Software Engineering. Presented at the The SEMAT Workshop on a General Theory of Software Engineering, International Conference on Software Engineering, San Francisco, CA, USA.
Ralph, P. (2013b). The Sensemaking-Coevolution-Implementation Theory of Software Design. arXiv: 1302.4061 [cs.SE].
Ralph, P. (2013c). The Two Paradigms of Software Design. arXiv:1303.5938 [cs.SE].
Ralph, P., Johnson, P., & Jordan, H. (2013). Report on the First SEMAT Workshop on a General Theory of Software Engineering (GTSE 2012). ACM SIGSOFT Software Engineering Notes, 38(2).
Royce, W. W. (1970). Managing the development of large software systems, Proceedings of WESCON, the Western Electronic Show and Convention.
Schwaber, K. (2004). Agile project management with Scrum. Microsoft Press.
Schwaber, K., & Sutherland, J. (2010). The Scrum Guide. Scrum.org. Retrieved July 2, 2012, from http://www.scrum.org/scrumguides/
Standish Group. (2009). CHAOS Summary 2009. Boston, MA, USA. Retrieved July 2, 2012, from http://www.standishgroup.com/newsroom/chaos_2009.php.
Truex, D. P., Baskerville, R., & Klein, H. (1999). Growing systems in emergent organizations. Communications of the ACM, 42(8), 117–123.
Truex, D. P., Baskerville, R., & Travis, J. (2000). Amethodical systems development: the deferred meaning of systems development methods. Accounting, Management and Information Technologies, 10(1), 53–79.
Turk, D., France, R., & Rumpe, B. (2005). Assumptions Underlying Agile Software-Development Processes. Journal of Database Management, 16(4), 62–87.
Turner, J. A. (1987). Understanding the elements of systems design. In Critical issues in information systems research (pp. 97–111). New York, NY, USA: John Wiley & Sons Inc.

Vidgen, R., & Xiaofeng, W. (2009). Coevolving Systems and the Organization of Agile Software Development. Information Systems Research, 20(3), 355–376.
Wang, X., & Vidgen, R. (2007). Order and Chaos in Software Development: A comparison of two software development teams in a major IT company.
Yin, R. (2003). Case study research: Design and methods (3rd ed.). California, USA: Sage Publications.
Zheng, Y., Venters, W., & Cornford, T. (2011). Collective agility, paradox and organizational improvisation: the development of a particle physics grid. Information Systems Journal, 21(4), 303–333.