ASSIMILATION OF AGILE PRACTICES IN USE

RUNNING HEADLINE: ASSIMILATION OF AGILE PRACTICES IN USE

Xiaofeng Wang

Lero, University of Limerick

Limerick, Ireland

Kieran Conboy

School of Information Systems, Technology and Management, UNSW

Sydney, Australia

CISC, NUI Galway, Galway, Ireland

Minna Pikkarainen

VTT - Technical Research Centre of Finland

Oulu, Finland

Number of words: 8,477 (main text)

ASSIMILATION OF AGILE PRACTICES IN USE

RUNNING HEADLINE: ASSIMILATION OF AGILE PRACTICES IN USE

Number of words: 8,477 (main text)

Abstract. *Agile method use in information systems development (ISD) has grown dramatically in recent years. The emergence of these alternative approaches was very much industry-led at the outset, and while agile method research is growing, the vast majority of these studies are descriptive and often lack a strong theoretical and conceptual base. Insights from innovation adoption research can provide a new perspective on analyzing agile method use. This paper is based on an exploratory study of the application of the innovation assimilation stages to understand the use of agile practices, focusing in particular on the later stages of assimilation, namely acceptance, routinisation and infusion. Four case studies were conducted and based on the case study findings, the concepts of acceptance, routinisation and infusion were adapted and applied to agile software development. These adapted concepts were used to glean interesting insights into agile practice use. For example, it was shown that the period of use of agile practices does not have a proportional effect on their assimilation depths. We also reflected on the sequential assumption underlying the assimilation stages, showing that adopting teams do not always move through the assimilation stages in a linear manner.*

Keywords: agile method, systems development, assimilation stages, acceptance, routinisation, infusion, agile practice assimilation stages

INTRODUCTION

Agile methods represent quite a popular initiative which complements previous critiques of formalised methods (e.g. Baskerville et al. 1992), and have been well received by practitioners and academics. According to Tan and Teo (2007), "agile techniques are fast becoming the adopted methodology commercially". The fifth annual 'State of Agile Development' survey sponsored by VersionOne, including 4,770 participants from 91 countries, shows that approximately 28% of companies adopted agile processes, with this large uptake evident across all sizes of organisation from small enterrpises to large multinationals (VersionOne, 2010). A number of methods are included in the agile[1] family, the most notable being *eXtreme Programming (XP)* (Beck, 2000), *Scrum* (Schwaber and Beedle, 2002), the *Dynamic Systems Development Method (DSDM)* (Stapleton, 1997), *Crystal* (Cockburn, 2001), *Feature Driven Design* (Coad and Palmer, 2002), and *Lean Software Development (LSD)* (Poppendieck, 2001). There are many books, journals and articles explaining these agile methods. While generally these text-book definitions are very detailed and prescriptive there is often a substantial difference between the textbook 'vanilla' version of the method and the method actually enacted in practice. Fitzgerald et al. (1997) refer to the latter as the "method-in-action". Prescribed practices are constantly tailored to suit the specific needs of teams and human nature inevitably leads to diverse interpretations and implementations of a method.

[1] There are many different interpretations of the term *agile*, so much so that it "has lost alot of its meaning" (Conboy, 2009). Therefore this paper reuses a definition from Conboy (2009) which is based on an extensive conceptual review of other studies of agility not just in IS, but also in fields such as manufacturing and management: ISD method agility is the continual readiness of an ISD method to rapidly or inherently create change, proactively or reactively embrace change, and learn from change while contributing to perceived customer value (economy, quality and simplicity), through its collective components and relationships with its environment.

There are an increasing number of empirical studies on agile methods in real-world contexts (e.g. Grenning, 2001; Lippert et al., 2003; Murru et al., 2003; Rasmusson, 2003; Sharp & Robinson, 2004; Fitzgerald et al., 2006; Mangalaraj et al., 2009). However the research on agile methods in practice is yet to yield significant systematic and insightful knowledge that can either guide future research or inform effective adoption and use of these methods in practice. Two issues in particular represent the main challenges to this branch of research. Firstly, there is a lack of strong theoretical and conceptual foundation to many studies of agile methods (Abrahamsson et al., 2009). In the absence of sound, systematic research there are few lessons learned across studies, and thus the existing body of knowledge is somewhat fragmented and inconclusive. Secondly, agile method use is often superficially judged as *used* or not *used*, whereas the actual implementation can be subtle, partial and inconsistent, and so categorising a method as *used* or *not used* may be overly simplistic (Conboy & Fitzgerald, 2011). Therefore, there is a need for studies that use a sound theoretical basis to explore the *degree* or *extent* of agile practice use.

The concepts of innovation assimilation stages developed in the Information Technology (IT) innovation research are one promising theoretical lens for this purpose. Rogers (1983) contends that a practice perceived as new by adopters can be characterised as an innovation, and so agile methods can be defined as process innovations of an ISD team. The innovation assimilation stages provide a structured mechanism to analyse agile method use, respecting the incremental nature of adoption as opposed to an overly simplistic binary perspective (i.e. either a method is used or it is not). Gallivan (2001) proposes six assimilation stages based on Cooper and Zmud (1990) and Saga and Zmud (1994), namely *initiation*, *adoption*, *adaptation*, *acceptance*, *routinisation* and *infusion*. The later three stages of assimilation, which describe various degrees of use of an innovation, have particular relevance to the study of agile methods in use.

The goal of this study is to apply the innovation assimilation concepts to develop a deeper understanding of agile method use. The agile methods investigated in this study are eXtreme Programming (XP) (Beck, 2000) and Scrum (Schwaber & Beedle, 2002)[2]. These methods were chosen for a number of reasons, the foremost being that they are the most widely used of the agile method family (as shown in VersionOne, 2009). Secondly, they are very diverse in that XP is very prescriptive and engineer-oriented while Scrum is in essence a project management method (Abrahamsson et al., 2002). By studying these two methods, this piece of research ensures that lessons learned consider both perspectives. Finally, existing research has shown that ISD teams often use a combination of XP and Scrum practices (Fitzgerald et al., 2006), and studying one set in isolation may not give a true reflection of agile adoption. We have adopted a multiple case study approach, studying four ISD teams who use either XP, Scrum or a combination of both.

The remainder of the paper is organised as follows. The next section introduces innovation assimilation theory. The research approach is then outlined, justified, and followed by a profile overview of the four cases. The results of the data analysis are then presented, and the concepts of later assimilation stages are reflected upon, resulting in the adaptation of these concepts in the context of agile software development. The usefulness

[2] XP and Scrum have provided the focus for over 20 texts, an annual conference and indeed the vast majority of agile method academic research and practice to date (Conboy & Fitzgerald, 2011). Given its popularity in both research and practice, we chose to focus on XP and Scrum in this study.

of the adapted concepts is demonstrated. The last section concludes with final remarks, limitations and suggestions for future work.

INNOVATION ASSIMILATION STAGES

Meyer and Goes (1988) define *assimilation* as an organisational process that "(1) is set in motion when individual organisation members first hear of an innovation's development, (2) can lead to the acquisition of the innovation, and (3) sometimes comes to fruition in the innovation's full acceptance, utilization, and institutionalization" (p. 897). In line with this definition, an *assimilation stage* describes how deeply an adopted innovation penetrates the adopting unit (company, group or individuals) (Gallivan, 2001). In order to understand the various stages of innovation assimilation, including the factors and events that influence them, Gallivan (2001) suggests a six-staged model based on the work of Cooper and Zmud (1990) and Saga and Zmud (1994). The six assimilation stages are as follows (Gallivan, 2001, p.59, p.63):

- *Initiation*: a match is identified between an innovation and its intended application environment;
- *Adoption*: the decision is made to adopt the innovation;
- *Adaptation*: the innovation is developed, installed and maintained, and organisational members are trained to use the innovation;
- *Acceptance*: members are committed to using the innovation;
- *Routinisation*: usage of the innovation is encouraged as a normal activity in the organisation; the innovation is no longer defined as something out of the ordinary;
- *Infusion*: the innovation is used in a comprehensive and sophisticated manner. Three different facets of infusion are described:
    - *Extensive use*: using more features of the innovation;
    - *Integrative use*: using the innovation to create new workflow linkages among tasks;
    - *Emergent use*: using the innovation to perform tasks not in the pre-conceived scope.

Cooper and Zmud (1990) imply an "unfreeze-refreeze" sequential model of these stages by associating initiation with Lewin's (1952) unfreezing stage, adoption and adaptation with change stage, and acceptance, routinisation and infusion with refreezing stage. Lewin's (1952) three-stage change model has been criticized for failing to account for feedback, or for the situated nature of actions (Orlikowski & Hofman, 1997; Palmer & Dunford, 1997). This criticism reflects a mechanistic interpretation of Lewin's model. Actually Rosch (2002) suggests that Lewin was fully aware of the situational nature of actions, but was limited at the time as the language and concepts describing the interaction of context and situation were yet to be developed (Rosch, 2002). Cooper and Zmud (1990) claim the sequential model may be more appropriate for technologies that are borrowed or adapted rather than custom made. Agile methods relate to the former category, and so the assimilation stages can be applicable to agile methods adoption and use.

Our study focuses on the later assimilation stages, i.e., acceptance, routinisation and infusion. As Mangalaraj et al. (2009) suggest, the post-adoption stage is deemed to be important in enriching our understanding of downstream phases of IT innovations, but the majority of research on software process diffusion has

concentrated on the early stages of adoption. One exception is Mangalaraj et al. (2009) who study the acceptance of agile methods to determine how factors either facilitate or impede their acceptance. They examine agile acceptance at the method level, and use acceptance in a broad sense which is not differentiated from later assimilation stages (i.e., routinisation and infusion). However, we contend there is a need to examine agile methods at the practice level due to the diversity of agile practices even within one agile method. More importantly, there is a need to examine agile practices use in a manner that goes beyond acceptance stage. In order to do so, we need clear definitions of the later assimilation stages that are relevant to agile practices. We believe that the concept of innovation assimilation stages discussed above serve this purpose.

RESEARCH APPROACH

An exploratory, case study approach was chosen for this research for a number of reasons. Firstly, this study is one of the first studies to use innovation assimilation stage concepts to investigate agile practices in use. Secondly, the assimilation theories that were used were significantly adapted, and so using a well-established previously tested instrument was not viable. Also, prior to engaging in the study, the researchers were aware that many of the modifications and sophisticated uses of the agile practices would be quite subtle and in some cases difficult to detect and verify. In such cases, exploratory research using methods such as case studies can help identify and hone in on such phenomenon (Yin, 2003) The case study approach is considered suitable for this study since it investigates a phenomenon in a real-life setting. It is also beneficial where control over behaviour is not required or possible, as research data can be collected through observation in an unmodified setting (Yin, 2003). This study uses a multiple-case design to allow a cross-case pattern search. It also allows us to see processes and outcomes across many cases, and to understand how they are affected by local conditions to develop more sophisticated descriptions and powerful explanations (Miles & Huberman, 1994). Each of the four cases is based on an ISD team in a different organisation, using a set of practices from XP, Scrum or both. The team were the unit of analysis, since agile practices are generally implemented at the team level e.g. team retrospectives and planning meetings. This choice also responds to Mangalaraj et al.'s (2009) argument that team level analysis is a much needed departure from prior studies that have used either organisation or individual as the unit of analysis.

Data Collection and Analysis

To improve the reliability and repeatability of the research, we sought to provide an 'audit trail' of the research process from data collection through to the drawing of conclusions. We followed Kirsch's (2004) model, defining a set of procedures to (i) identify and select project cases, (ii) determine who to interview and (iii) plan how interviews were to be conducted. We prepared an interview protocol based on the concepts of innovation assimilation stages discussed earlier. Data was collected primarily through personal face-to-face interviews. Personal interviews are well suited for exploratory research, allowing expansive discussions to illuminate factors of importance (Yin, 2003; Oppenheim, 1992). Also, the information gathered is likely to be more accurate than some other methods since the interviewer can avoid inaccurate or incomplete answers by explaining the questions to the

interviewee (Oppenheim, 1992). The interviews lasted between 50 and 120 minutes. The questions were largely open-ended, allowing respondents freedom to convey their experiences and views, and expression of the socially complex contexts (Yin, 2003; Oppenheim, 1992) that underpin ISD and agile method use. The interviews were conducted in a responsive (Rubin & Rubin, 2005; Wengraf, 2001), or reflexive (Trauth & O'Connor, 1991) manner, allowing the researcher to follow up on insights uncovered mid-interview, and adjust the content and schedule of the interview accordingly. In order to aid analysis of the data after the interviews, all were recorded with each interviewee's consent, and were subsequently transcribed, proof-read and annotated by the researchers. In any cases of ambiguity, clarification was sought from the corresponding interviewee via telephone or e-mail. Documentation reviews and field notes were used as complementary data collection methods.

The data was analysed in two stages. Firstly, each case was analysed as a stand-alone entity, or what Yin (2003) refers to as 'within-case analysis'. The researchers then engaged in cross-case comparison. In the within-case analysis, the agile practices used and the manner of use by each team were identified and analysed. The analysis process was guided by the innovation assimilation theory. Cross-case comparison was intended to enable a better understanding of the assimilation stages reached. The background information of the teams was also taken into account in this analysis.

Background to the Cases

Team A was based in a multinational organisation providing service-oriented architecture solutions. XP was introduced and adopted company-wide for four years prior to the study. Senior management saw the potential project management benefits of some XP practices and so mandated a company-wide adoption of the method. Team A received formal XP training at this time.

Team B resided in a medium-sized company providing IT security services to clients in over 90 countries. Before adopting agile methods, the company reached CMMI (Capability Maturity Model Integration) Level 3 and the quality and reliability of software was considered high. Initially the company used a more waterfall style process model. Subsequently the senior management made a decision to adopt agile methods across the organisation to enable a faster response to new threats in the marketplace. Large Scrum and XP training sessions were organised for all project managers and developers. Following this three agile pilot projects were launched, including Team B.

Team C was based in a small software house specialising in network security. The company's previous three projects had failed to deliver, prompting a shift to agile development. The company hoped to create conditions to allow developers to produce software better, faster and cheaper. Team C had a six-month period of intensive XP training, and subsequently applied XP to several projects with perceived success.

Team D was based in a large multinational consulting organisation, developing a web-based system for a large government body. The organisation traditionally mandated a standard development method across the entire company, and used this project on a one-off pilot basis to evaluate agile methods as an alternative. Five team members attended training courses at the project outset, and some external agile method trainers were also brought on-site to assist with the transition. The system was delivered four months early and well under budget,

and the development team and clients were apparently very satisfied with the initiative. Table 1 provides an overview of the profiles of the four case teams.

	Team A	Team B	Team C	Team D
Team size	8	6	4	20
Team composition	6 developers, 1 test manager, 1 project manager.	4 developers, 1 tester, 1 Scrum master	3 developers, 1 project manager	1 managing partner, 1 senior manager, 4 business analysts, 12 developers, 2 testers
Location	Partially distributed in two continents	Collocated in an open office space	Collocated in an open office space	Distributed across 2 sites
Development method	XP	Scrum/XP	XP	Scrum/XP
Years of agile method adoption	4.5	1.5	5	3
Type of system developed	Commercial product, middleware platform	Commercial product IT services in security	External application, management and security systems	External application, Web-based systems
Company background	Medium, service-oriented solution provider	Medium, providing security solution and services	Small software house, specialised in security, management systems	Large multinational consulting organisation

Table 1: The profiles of the four software development teams

ASSIMILATION OF AGILE PRACTICES IN THE FOUR CASES

Table 2 provides a list of the Scrum and XP practices and textbook definitions. They serve as a template against which each team's use of the agile practices is analysed. The assimilation level of each practice is then identified based on the definition of innovation assimilation stages in the previous section.

Agile practice	Textbook definition
Small releases (Sprints)	Put a simple system into production quickly, and release new versions on a very short cycle.
Planning game (Sprint planning)	Prioritisation of scope for next release based on a combination of business priorities and technical estimates.
Stand-up meetings (daily meetings)	Short daily status meeting.
Retrospectives (post game sessions)	Reflect on method strengths and weaknesses after each cycle.
40-hour week	Work time is generally limited to 40 hours per week.
On-site Customer	Include an actual user on the team, available full-time to answer questions.
Simple design	The design of the system should be as simple as possible.
Pair programming	Code is written by two programmers on the same machine.
Testing first	Continually write tests, which must run flawlessly for development to proceed. Write test code before writing function code.
Continuous integration	Integrate and build the system every time a task is completed – this may be many times per day.
Collective Ownership	Anyone can change any code anywhere in the system at anytime.
Refactoring	Programmers restructure the system, without removing functionality, to improve code, performance, simplicity and flexibility.

Table 2: The textbook descriptions of the Scrum and XP practices (from Fitzgerald et al. 2006)

Small Releases (Sprints)

This practice suggests the team put a simple system into production quickly, and release new versions on a very short cycle. In XP short iterations of 2 weeks are suggested. In Scrum it is represented as 30-day 'Sprints'.

In Team A and Team D, the iterations varied in length. Both A and D did move to more iterative development as would be expected with agile, but the iterations varied from 1 week to 9 weeks and time-boxing was applied in some cases and not in others. More specifically, Team A collaborated with other units of the company on a product line. All followed 6-week milestones between releases to maximise cross-unit coordination. At the end of each milestone, each unit was required to produce a demonstratable delivery. Within this 6-week period, they were not obliged to use 2-week iterations, and as a consequence, Team A did not use 2-week iterations. In the case of Team D, while iterations of software were used, the duration of these varied significantly, and ended whenever the team felt that they had done enough to show something substantial. In addition, deliverables were not time-boxed into iterations and so it was clear that the core components of a Scrum Sprint were not being adhered to.

In contrast, Team B and Team C used this practice consistently and routinely. Team B used regular Sprints to pace the development activities. Team C used shorter, one-week iterations. Both teams delivered fully tested working software at the end of each iteration.

Planning Game (Sprint Planning)

The 'planning game' is a practice of XP, while the corresponding practice in Scrum is referred to as Sprint planning. During a planning game or Sprint planning meeting, tasks for the next Sprint are prioritised based on a combination of business need and technical estimates.

In the case of Team A, since 2-week iterations were not used, the planning game was only conducted when needed, when dealing with large and complex features for example. As with Team A, because Team D did not use standard short iterations, and so standardised use of planning was also difficult. At the start of some iterations the work was comprehensively planned, but less so in other cases. However, customisation of the practice was evident, along with a more sophisticated use of the practice than the textbook prescription. Examples of sophistication include a voting mechanism in the planning meeting. All user stories were uploaded to an intranet and all customer employees could vote on which should be prioritised, allowing representation and consensus far beyond the traditional agile practice of involving a single customer representative. However, it must also be noted that this modification caused problems, as the prioritised user stories were often dependent on the size of some departments and the level of awareness within each. For example, on one occasion 84 votes were cast by the accounting department while the IT department cast one. The practice was modified further to combat this anomaly, with votes subsequently limited to one per department. This then became the de facto standard approach at the start of all future iterations.

Team B held regular Sprint planning meetings at the beginning of each monthly Sprint, in which requirements were tightly defined and placed in a Sprint backlog. Developers did not have much influence on how the requirements were selected and analyzed. The team was, however, responsible for defining tasks and

providing estimates for implementing them in that Sprint. The customers were partially involved in each Sprint planning meeting, but the product owner, who played the customer proxy role, was always involved in the meeting.

Team C also used planning games routinely, involving index cards (1/4 A4 size) and storyboards, two common techniques used by XP teams to facilitate planning. In addition, Team C customised the use of index cards, using different colours to distinguish between different types: green for user stories, blue for tasks and red for spikes[3]. All the cards were then appended to a storyboard and arranged in categories. Simultaneously, all information about user stories and acceptance tests were recorded in the project wiki, to which all developers and customers had access. These activities provided the rationale for regarding these teams as making sophisticated use of the practice.

Stand-up Meetings (Daily Meetings)

Stand-up meetings suggested by XP (or daily meetings by Scrum) are short daily status meetings where team members quickly plan the work of the day and identify any technical obstacles to fulfilling that work. Typical questions used in stand-up meetings are: "What did I work on yesterday?", "What do I plan to work on today?" and "What is getting in my way?"

Team A adopted stand-up meetings but did not use them on a daily basis. They typically held quick meetings on days where needed or where many days had passed without a meeting.

Team B held daily meetings regularly, which facilitated dissemination of information and analysis of project metrics which were then used for planning and project monitoring. Daily meetings had also become an effective problem-solving mechanism in Team B. Most problems, including design issues, were actually solved based on discussions initiated in these meetings.

In contrast, Team C and D used this practice in a much more comprehensive and sophisticated manner, going beyond regular use. In the case of Team C, the team referred to this practice as a "steering" mechanism. In each working day, the team held a brief steering session in the morning to plan work for that day, and identify potential technical obstacles that may exist. Steering meetings occasionally occurred more than once a day, with morning and afternoon meetings if necessary. The comprehensive use of the practice was demonstrated in Team C by this increased usage. In the case of Team D, they not only started holding meetings twice a day, but also customised this practice quite substantially after the first few months. They moved beyond the textbook format, whereby the team not only identified and discussed problems, but reprioritised and reallocated tasks to solve these problems. These are evidence that the practice was not only a routine part of the development process but also inherently infused in the team's culture and behaviour.

Retrospectives (Post Game Sessions)

Retrospectives suggested by XP (or post game sessions by Scrum) are meetings where a development team reflects on the strengths and weaknesses of the development method and process after each iteration (or Sprint).

[3] Spikes are exploratory tasks aiming at solving any issues that emerge during the course of development.

In Team A, only the team manager formally reflected on their development process, and even this, by his own admittance, was not done on a regular basis.

Retrospectives were conducted more frequently and consistently in Team B and D. At the end of each iteration they reviewed and adapted use of the practices effectively. For example, Team B's Scrum master used retrospectives to facilitate test driven development. He presented the concrete measurement of the test coverage at the retrospectives and facilitated the team's decision to improve quality of the developed product by adopting test first and continuous integration practices.

In addition to the routinised use, Team C demonstrated more intensive use and high level customisation of the practice. The team named this practice "feedback" and it was the first thing they did in a working day (rather than conducting it once per iteration) before the daily stand-up meeting. During the feedback session the team members reflected on the previous working day. The feedback not only focused on the development process, but also on achievements, feelings and anxieties amongst team members. The team recorded the feedback in the team wiki to document lessons learnt. The team had even extended the practice to self retrospectives, believing that self reflection was a precondition for effective team retrospectives.

40-hour Week

This practice advocates that work time should be limited to 40 hours per week. This practice was not implemented by Team A.

In team B the 40-hour week was endorsed and adhered to, and the working hours of developers were time boxed to ensure they did not work overtime.

In Team C and Team D the 40-hour week was not only embraced but also implemented in a sophisticated manner. A typical working day of Team C was eight hours, time-boxed into 15 working units. One unit was composed of 25-minute working time followed by a 5-minute break. During the day there were two long breaks of 15 minutes each. The team actually used a timer to help them to keep this pace. In the case of Team D, they modified the practice beyond the recommended 8 hours per day. Because the clients worked to a split-shift rota, the team alternated between 6 and 10 hour days, and in some cases stretched this to 4 and 12 hour shifts for a 2 month period to align with client working hours.

On-site Customer

This practice suggests that the development team should include an actual user on the team, who is available full/time to answer questions.

The on-site customer practice was not fully implemented or regularly used in Team A. The team used internal resources (product managers) as customer proxies in the absence of a real customer, and even the customer proxy was not always on-site.

The practice was not fully implemented in Team B and Team C either, but unlike Team A, they used the adapted practice regularly. Team B used the product owner as customer proxy, who worked closely with the development team and was always present in Sprint planning meetings. In addition, the Scrum master was in daily contact with the real customers of the project. Team C managed to involve their customers in every

planning meeting and acceptance testing, and communicated frequently with them within the weekly iteration, to achieve the same effect that a full on-site customer can bring.

The assimilation process of this practice in Team D was intriguing. The on-site customer practice was routinely implemented at the start of Team D's project, with the client representative based full-time on the development site. This continued for approximately six months followed by a more sophisticated use of the practice after this point. Firstly, the team found that a single person could not provide a sufficiently broad knowledge of the customer organisation and its needs. Therefore a second "roving role" was introduced, whereby in addition to the original representative a second would visit the site each day. This person would rotate across departments ensuring that the team had access to a legal expert on one day, a network technician on another, a system user on the third and so on. The permanent and 'roving' roles would then interchange between the two distributed locations, maximising dissemination of knowledge across the entire development effort. However, while the on-site customer practice was highly routinised in its basic form, it disintegrated once the sophisticated modifications were introduced. Since this sophistication confused the team and made it hard to maintain the availability of each type of on-site customer to each site when it was needed, the team abandoned the practice in less than two months.

Simple Design

The design of the system should be as simple as possible in order to quickly respond to change requests. This practice was not evident in Team A.

Team B's system design was initially complex but changed to follow a more simple approach when developers' knowledge of agile development methods increased. However, the practice was not adhered to consistently due to technical obstacles encountered by the team.

In contrast, Team C highly embraced and adhered to this practice. The team believed that design emerged from code and therefore simple design was embodied by simple code implemented in everyday work.

In the case of Team D, simple design was not only highly embraced and strictly adhered to but also implemented in a more sophisticated manner by involving the on-site customer. The early stages of the project suffered from high levels of complexity in terms of code, diagrams and other artefacts. This caused significant problems in terms of communication with the on-site customer and users in general. To address this, the on-site customer was deemed to be the '*acid test*' of design. When every diagram or piece of code was completed the customer would be asked to review and if she was not able to understand and explain the artefact it was deemed to be not simple enough and had to be simplified or redesigned. This practice was used closely with the on-site customer practice and created a new workflow in which the on-site customer was involved in tasks that they were not exposed to previously.

Pair Programming

This practice suggests that software code is written by two programmers on the same machine.

Neither Team A, B or D adopted the pair programming practice routinely or applied it to all programming tasks. What is quite interesting in these three teams, however, is that while pair-programming was not routinely

used, there were still examples of emergent or customised use of the practice. In Team A, use of pair programming was limited to debugging, but pairing was used for the estimation of user stories. Team B paired for code reviews. They also paired new team members with experienced ones as a way to train new staff. In Team D, pairing was limited to database development tasks and not other aspects of the project. However, they did use a software utility which projected an image of one developer's screen to another, allowing two developers in two different locations to pair program. Also, the team modified the key commands on each keyboard, so each developer's code was automatically highlighted in a different colour, and thus easy to distinguish.

In comparison, Team C used all features of pair programming routinely on all tasks. Each pair shared a desktop, with one developer using keyboard (called the driver) and the other using mouse (called the navigator). Role switching happened when the navigator had some new idea. He could take the keyboard from the driver and start to write code. Pairing was self-arranged. The developers could choose whom to pair with. Pair rotation, i.e., swapping pairing partners, happened frequently so that the team members had opportunities to collaborate and share knowledge.

Testing First

This practice suggests that tests should be continually written, preferably before writing function code, and must run without error for development to proceed.

This practice was highly embraced and adhered to by Team A, Team B and Team D. These teams generally adopted a comprehensive test-oriented development framework, as illustrated in Figure 1, which helped them to adhere to the practice.

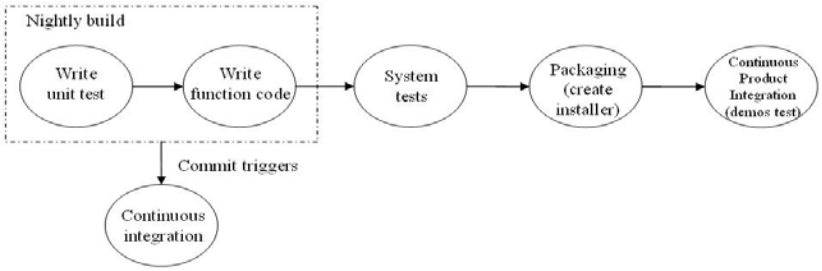

Figure 1: The test-oriented development framework implemented in Team A, B and D

In addition to adhering to the practice and embracing a similar framework, Team C also demonstrated an emergent use of the practice by using it to learn third party software. To understand how a piece of software works, rather than reading documentation, they wrote tests and ran them on it to understand its functionalities.

Continuous Integration

This practice requires the system to be integrated every time a task is completed, often happening many times per day.

Continuous integration was implemented consistently in Team A, C and D, with all completing a full integration at least once every day. The comprehensive test-oriented development framework these teams adopted (as shown in Figure 1) combined continuous integration with test-driven development to guarantee high quality code.

In addition, Team B implemented the practice in a more intensive manner from the project outset. Even though the team was challenged by technical difficulties in implementing the practice, they did not abandon the practice and all problems were continuously discussed in every iteration retrospectives meeting. When the system integration process finally worked, the team started using intensive hourly builds. Automated hourly builds were considered very beneficial and made testing much easier in each Sprint.

Collective Ownership

Collective ownership practice allows any team member to change any code anywhere in the system at anytime, as opposed to a structure whereby each developer 'owns' various parts of the system.

In Team A and Team D collective ownership was endorsed by the team members and regarded as a natural result of team collaboration. Team A and Team D had achieved collective ownership of code, as the practice is intended.

Collective ownership in Team B and C went well beyond the original intended use of the practice, whereby just code was collectively owned. In both teams, the team members shared other forms of working resources or results as well. For example, both teams had their team email addresses that they used as a shared single identity in communication with their customers. Additionally, achievements in both teams were shared among team members, and no one claimed ownership of a particular solution.

Refactoring

This practice suggests that programmers restructure the system, without removing functionality, to improve code performance, simplicity and flexibility.

All four teams were aware of the practice but none routinely used it for every piece of code. They did so only if there was a good reason. Therefore, the assimilation of the practice was at the acceptance stage in all four teams. For instance, in Team B refactoring was used only for complex parts of the system due to the perceived high cost of refactoring. In Team C the developers refactored code only when there were duplicated behaviours in the system. They attempted to deliver working code that was useful, but not perfect.

Table 3 summarises the case analysis results, presenting agile practices used in the four cases and the assimilation stages they have reached.

Agile practice	Practice in use in the four cases			
	Team A	Team B	Team C	Team D
Small releases (Sprints)	Used in an ad-hoc manner	Used routinely and consistently	Used routinely and consistently	Used in an ad-hoc manner
Planning game (Sprint planning)	Used on an ad-hoc basis	Regular Sprint planning meetings	Routinely used, and supported by different planning techniques	Customised with voting mechanism but not regularly used
Stand-up meetings	Not routinely used	Regularly conducted	Routinely used with	Routinely used with

(daily meetings)			increasing intensity	increasing intensity and adjustment
Retrospectives (post game sessions)	Only manager reflected on the process irregularly	Regularly conducted	Regularly conducted with increasing intensity and extension to self-reflection	Regularly conducted
40-hour week		Endorsed and adhered to	Endorsed and adhered to using 25-minute time-box	Endorsed and adhered to through varying working hours to suit its specific need
On-site Customer	Implemented partially using part-time customer proxy, not regularly used	Implemented partially using part-time product owner, but regularly used	Implemented partially by involving customer in planning game, but regularly used	
Simple design		Accepted as a useful approach	Highly embraced and adhered to	Highly embraced, and enabling more customer participation in the development process
Pair programming	Not routinely used, used to debugging and estimation tasks	Not routinely used, used for code review and new staff training	Routinely used on all tasks, with role switching and pair rotation	Not routinely used, limited use, but used sophisticatedly
Testing first	Highly embraced and adhered to	Highly embraced and adhered to	Highly embraced and adhered to, and used in studying new software package	Highly embraced
Continuous integration	Highly embraced and adhered to	Highly embraced and adhered to, and used in an intensive manner	Highly embraced and adhered to	Highly embraced and adhered to
Collective Ownership	Used and endorsed by whole team	Used and endorsed by whole team, not limited to code only	Used and endorsed by whole team, not limited to code only	Used and endorsed by whole team
Refactoring	Used when it makes sense	Used when it makes sense	Used when it makes sense	Used when it makes sense

Table 3: A summary of agile practice assimilation in the four cases

DISCUSSION

Our case analysis shows that the concepts of the later assimilation stages of innovation, i.e., acceptance, routinisation and infusion, are largely applicable to the context of agile methods, and reflected in the assimilation of the agile practices used in the four case teams. However, some scenarios uncovered in our case studies do not lend themselves naturally to the conceptualisation of the assimilation stages defined in the literature. In this section we reflect on the definitions of the later assimilation stages in light of the case study results.

Acceptance of Agile Practices. The acceptance of an agile practice means that the adopting team are committing to using it. Logically, the fact that an agile practice is used by the team can be taken as evidence that the adopting team have committed to using it. In line with exiting literature, our case studies show that what the team actually use is not always the text-book version of the practice, but rather tailored to the specific context of the team, such as the planning game and pair programming practices in Team D for example. It is common that an agile adopting team customize the agile practices to suit the team's organisational context before they are put into use. Furthermore, customization is a continual process and the adopting team may constantly review and

adjust the agile practices they use, reflecting the essence of agility. Based on this understanding, we adapt the definition of the acceptance of an agile practice as "*a commitment to using an agile practice, either 'by the book', or in some tailored fashion"*.

Routinisation of Agile Practices. To understand if an agile practice reaches the routinisation stage, we need to know if it becomes a routine part of the development process of the team. For some practices, we can observe the frequency of usage to see if they are used regularly or routinely, such as the planning game (supposed to be used once per iteration), stand-up meeting (daily), or pair programming (continuously). However, the level of routinisation of some other practices, such as the 40-hour week or simple design, are difficult to judge if they are routinely used based on frequency, since they are more abstract and bear closer resemblance to principles than concrete, tangible practices. Therefore for these practices, rather than looking at the frequency of usage, we need to examine if the adopting team actually endorse or embrace them and how strongly they adhere to them. Based on this understanding, the routinisation of an agile practice can be defined as "*the extent to which an agile practice is frequently used, highly embrace and adhere to, and no longer considered as something out of the ordinary*".

Infusion of Agile Practices. If an agile practice is routinised in the adopting team's development process, it then can be further analysed to see if it reaches the infusion stage by examining if the team use the practice in a comprehensive or sophisticated manner. The three facets of infusion suggested by the innovation assimilation literature, i.e., extensive use, integrative use and emergent use, are indicators of such a manner. However, our case study results show that, in the context of agile practice assimilation, the indicators of infusion are not limited to these three. There are also *intensive use* and *deeply customised use* of an agile practice that indicate a sophistication or comprehensiveness. By *intensive use* we mean that an agile practice is used with intensity beyond that suggested by the textbook. The way retrospectives and stand-up meetings were used in Team C is a good example of intensive use of these practices. By *deeply customised use* we mean that an agile practice is adapted at a deep level to suit the need of the adopting team. Generally, deep level of customisation can only be done after the adopting team use the practice for a significant period of time and become proficient in using it. The 40-hour week implemented in Team C and D are examples of deeply customised use of the practice. Based on this understanding, we define the infusion of an agile practice as "*the adopting team not only using an agile practice routinely, but also in a comprehensive or sophisticated manner*", which is indicated by any of the five facets previously discussed.

It is important to note that the indicators of infusion should not be used in isolation, but in combination with routinisation, to decide if an agile practice is infused in the adopting team. This is in line with the assumption that the assimilation stages are sequential, i.e. acceptance to routinisation to infusion. Zmud and Apple (1992) show empirical evidence to support the argument that an organisation could not achieve infusion without having reached routinisation first. An agile practice can be used in a sophisticated manner, but if it is not routinised into the adopting team's development process, it is still considered to be at the acceptance stage despite the sophistication. The planning game and pair programming practices used in Team D are such examples. Instead teams using agile practices as a routine part of their development work may be in a good position to discover

comprehensive and sophisticated ways of using them. However, the sequential assumption underlying the assimilation stages does not necessarily mean that the adopting team can always succeed in transitioning through the stages. The assimilation of the on-site customer practice in Team D was a good illustration of this point. The well-routinised on-site customer practice fell apart shortly after the deeply customised use of the practice was introduced. An interesting avenue for future research would be to examine why the adopting team does not always progress sequentially through these assimilation stages.

Table 4 summarises the adaptation and extension of the concepts of the later innovation assimilation stages in the context of agile methods.

Assimilation stages	**Agile practice assimilation**
Acceptance	a commitment to using an agile practice, either 'by the book', or in some tailored fashion;
Routinisation	the extent to which an agile practice is frequently used, highly embrace and adhere to, and no longer considered as something out of the ordinary;
Infusion	The adopting team not only using an agile practice routinely, but also in a comprehensive or sophisticated manner, which is indicated by any of the following facets: - extensive use: more features of an agile practice are used; - integrative use: an agile practice is used to create new workflow linkages among tasks - emergent use: an agile practice is used to perform tasks not in the pre-conceived scope - intensive use: an agile practice is used with intensity beyond that suggested by the textbook. - deeply customised use: an agile practice is adapted at a deep level to suit the need of the adopting team

Table 4: The assimilation stages of agile practice in use

The assimilation stages of agile practices in use, in combination with the organisational context in which an ISD team is embedded, can help develop a better understanding of the adoption and use of agile practices. Even though the primary focus of our paper is to define the assimilation stages of agile practices in use, some insights on how agile methods are used and assimilated are gleaned from our study, which illustrates the usefulness of the proposed concepts in Table 4.

One observation from the results of data analysis is that the duration of usage does not affect the level of assimilation of an agile practice. As shown in Table 3, agile practices have been used in different modes and have reached different stages of assimilation in the four teams. Many practices remained at the acceptance stage in one team but reached deeper assimilation levels in the other teams over the same or even shorter period of time. It may be argued that, based on this analysis, the duration of use does not have a proportional effect on the assimilation stage that an agile practice can reach. Fishman (2000) argues that time would be one factor affecting the ability to reach the later assimilation stages, which means that companies that adopt new innovations early would have more time to reach deeper levels of assimilation and, therefore, get benefits of the use of innovation earlier. Our study suggests that, where agile method practices are concerned, time may not be an appropriate indicator in the evaluation of the assimilation stages of agile practices. This is an important aspect to take into

consideration for studies that intend to understand how agile practices have been routinised and infused in a team or organisation. Time is an important dimension in this kind of study, but one needs to be cautious when using it as a measure to determine 'successful' adoption of agile methods.

Another practical observation is that the needs of an adopting team may drive relevant practices to a deeper level of assimilation. In Team A for example, agile practices addressing management issues remained at the acceptance stage while development practices, such as testing and continuous integration, have reached the routinisation stage within the same timeframe. One possible explanation is that Team A needed to maintain an exceptionally high level of software quality, which was a consequence of the software product line approach adopted by the company. Quality related practices such as testing first and continuous integration can address this need. Meanwhile, the company of Team A always had an open culture where developers communicated and collaborated freely. The need to emphasise these aspects through adopting the relevant agile practices was relatively low. As a result, practices such as stand-up meetings did not reach the routinisation stage even though they were in place for over 4 years. This understanding has a practical implication on how to effectively adopt agile practices. Since the agile practices that address the needs of a team have the potential to be routinised or infused, rather than taking the whole set of practices of an agile method it may be sensible to identify the areas that a team most needs to improve and then select agile practices relevant to that need.

CONCLUSION

The popularity of agile methods has been increasing over the last decade among software development practitioners and researchers alike. However a systematic and insightful understanding of agile method in use is yet to be achieved. Our study is one of the first attempts to explore agile method in use through an innovation assimilation perspective and to conceptualise the assimilation processes, focusing in particular on the later stages of assimilation, i.e., acceptance, routinisation and infusion. Based on the findings of the multiple case studies involving agile method use in four software development teams, we reflected upon, adapted and extended the concepts of innovation assimilation stages in the context of agile methods. The adapted concepts of agile practice assimilation stages can be applied as a theoretical lens for other empirical studies on agile methods in use to explore the factors behind different assimilation stages of agile practices. The adapted concepts can also be operationalised into a set of constructs for quantitatively examining the degrees of agile practice assimilation. Our study also has implications for practitioners. The agile practice assimilation stages and insights drawn from them can help teams and senior management to better understand and benchmark how agile methods are actually adopted and used in real-world contexts and to reflect on their agile method implementation.

There are limitations of the case study approach we adopted, including issues regarding generalisation of findings from a small number of cases. Because the corporate, team and project characteristics are unique to each case study, comparisons and generalisations of case study results are difficult and are subject to questions of external validity (Kitchenham et al., 2002). However, Walsham (1995) argues that, when using a case study approach, researchers are not necessarily looking for generalisation from a sample to a population, but rather plausibility and logical reasoning through developing concepts and theory, drawing specific implications, and

contributing rich insight. This is the type of generalisation we wished to achieve. To evaluate and extend the proposed agile assimilation framework built from the four cases and make it more adequate for the study of agile practices in general, one possible avenue, rather than adding more similar cases, is to examine agile practices beyond those covered in this study i.e. XP and Scrum. Methods such as Lean Software Development, Feature Driven Development, Agile Project Management, Crystal and Adaptive Software Development are all methods that could be assessed. Another specific methodological limitation of this study is that, since only snapshots of agile practices in use have been taken and analysed, it is not possible to analyse how each agile practice moves along the assimilation stages over time. To address this limitation, longitudinal studies of agile practices assimilation are desired.

Future research can extend our study and examine the effectiveness of agile practice assimilation and the barriers and facilitators affecting this assimilation. The agile practice assimilation framework illustrates the path of assimilation, but it cannot answer, at least by itself, the questions such as "how and why the assimilation of practice progress from one stage to another?" or "is such progress a good thing?" Further studies that use the assimilation framework and innovation diffusion research to identify the contextual factors that impact the assimilation stages (e.g., Kwon & Zmud, 1987) or that link agile practice assimilation to success and effectiveness may yield some very interesting insights.

ACKNOWLEDGEMENT

Special thanks to Prof. Brian Fitzgerald, Vice President for Research at the University of Limerick, for his valuable advice on this research. This work was supported, in part, by Science Foundation Ireland grant 3/CE2/I303_1 to Lero.

REFERENCES

Abrahamsson, P., Conboy, K. & Wang, X. (2009) Lots Done, More To Do': The Current State of Agile Systems Development Research. *European Journal of Information Systems*, 18(4), 281-284.

Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2002) Agile Software Development Methods: Review and Analysis. *VTT Electronics, Espoo*, 1-107.

Ambler, S. W. (2002) *Agile Modeling: Best Practices for the Unified Process and Extreme Programming*, John Wiley & Sons., New York.

Baskerville, R., Travis, J. & Truex, D. (1992) Systems without Method. In *The Impact of Computer Supported Technologies on Information Systems Development,* Kendall, K., DeGross, J. & Lyytinen, K. (Ed,), Elsevier Science Publishers, North Holland, 241-269.

Beck, K. & Andres, C. (2004) *Extreme Programming Explained: Embrace Change*, *Second Edition*, Addison-Wesley, Boston.

Beck, K. (2000) *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, Mass.

Boehm, B. & Turner, R. (2004) *Balancing Agility and Discipline: A Guide for The Perplexed*, Addison-Wesley, Boston, MA.

Boehm, B. (2002) Get Ready for Agile Methods, with Care. *IEEE Computer*, 35, 64-69.

Coad, P. & Palmer, S. (2002) *Feature-Driven Development*, Prentice Hall, Englewood Cliffs, NJ.

Cockburn, A. (2001) *Crystal Clear: A Human-Powered Software Development Methodology for Small Teams*, Addison-Wesley, Reading, MA.

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. and B. Fitzgerald (2011) Method and Developer Characteristics for Effective Agile Method Tailoring: A Study of XP Expert Opinion. *ACM Transactions on Software Engineering Methodology,* 20(1), In Press.

Cooper, R. B., & Zmud, R.W. (1990) Information Technology Implementation Research: A Technological Diffusion Approach. *Management Science*, 36(2), 123-139.

Davis, F. D. (1989) Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly,* 13 (3), 319-339.

Dingsoyr, T., Hanssen, G. K., Dyba, T., Anker, G. & Nygaard, J. O. (2006) Developing Software with Scrum in a Small Cross-organizational Project. In *Proceedings of the Thirteenth European Conference on Software Process Improvement,* Richardson, I., Runeson, P. & Messnarz, R. (Ed.), Joensuu, Finland, October 2006, 5-15.

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

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

Fichman, R. G. (1992) Information Technology Diffusion: A Review of Empirical Research. In *Proceedings of the Thirteenth International Conference on Information Systems*, Dallas, 195-206.

Fichman, R. G. (2001) The Role of Aggregation in the Measurement of IT-related Organizational Innovation. *MIS Quarterly,* 25 (4), 427-455.

Fitzgerard, B., Hartnett, G. & Conboy, K. (2006) Customising Agile Methods to Software Practices at Intel Shannon. *European Journal of Information Systems,* 15 (2), 200-213.

Gallivan, M. (2001) Organizational Adoption and Assimilation of Complex Technological Innovations: Development and Application of a New Framework. *The DATA BASE for Advances in Information Systems,* 32 (3), 51-85.

Grenning, J. (2001) Launching Extreme Programming at Process-Intensive Company. *IEEE Software,* 18(6), 27-33.

Hazzan, O. & Tomayko, J. (2003) The Reflective Practitioner Perspective in eXtreme Programming. In *Proceedings of Extreme Programming And Agile Methods - XP/Agile Universe,* Maurer, F. & Wells, D. (Ed.), New Orleans, LA, USA, August 2003, 51-61.

Highsmith, J. (2004) *Agile Project Management*, Addison-Wesley, Boston, MA.

Hovorka, D. S. & Larsen, K. R. (2006) Enabling Agile Adoption Practices through Network Organizations. *European Journal of Information Systems*, 15 (2). 159-168.

http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf (last accessed 20 January, 2010).

Karahanna, E., Straub, D.W., & Chervany, N.L. (1999) Information Technology Adoption Across Time: A Cross Sectional Comparison of Pre-Adoption and Post-Adoption Beliefs. *MIS Quarterly*, 23(2), 183-213.

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

Kwon, T. & Zmud. R. (1987) Unifying the fragmented mod- els of Information Systems implementation. R. J. Boland, R. A. Hirshcheim, eds. *Critical Issues in Information Systems Research*. John Wiley and Sons, New York, 227–251.

Layman, L., Williams, L. & Cunningham, L. (2006) Motivations and measurements in an agile case study. *Journal of Systems Architecture*, 52(11), 2006, 654-667.

Layman, L., Williams, L., Damian, D. & Bures, H. (2006) Essential Communication Practices for Extreme Programming in a Global Software Development Team. *Information & Software Technology*, 48(9), 781-794.

Lewin, K. (1952) Group Decision and Social Change. In Nwecomb and Hartley (Eds.) *Readings in Social Psychology*. Henry Holt and Company, New York, 459-473.

Lippert, M., Becker-Pachau, P., Breitling, H., Kock, J., Kornstädt, A., Roock, S., Schmolitzky, A., Wolf, H. & Züllighoven, H. (2003) Developing Complex Projects Using XP with Extensions. *IEEE Computer Society*, 36(6), 67-73.

Mangalaraj, G., Mahapatra, R., & Nerur, S. (2009) Acceptance of software process innovations - the case of extreme programming. *European Journal of Information Systems*, *18*(4), 344-354.

McCracken, G. (1988) *Qualitative Research: The Long Interview*, Sage Publications, Beverly Hills, CA.

Meyer, A. D. & Goes, J. B. (1988) Organizational Assimilation of Innovation: a Multilevel Contextual Analysis. *Academy of Management Journal*, 31(4), 897-923.

Miles, M. & Huberman, A. (1999) *Qualitative Data Analysis*, Sage, London.

Miles, M. B. & Huberman, A. M. (1994) *Qualitative Data Analysis: an Expanded Sourcebook*, Saga, London.

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

Nawrocki, J., Jasinski, M., Walter, B. & Wojciechowski, A. (2002) Combining Extreme Programming with ISO 9000. in *Proceedings of the First EurAsian Conference on Information and Communication Technology*, Tehran, Iran, October 2002, 786-794.

Oppenheim, A. (1992) *Questionnaire Design, Interviewing and Attitude Measurement*, Continuum, New York.

Orlikowski, W. J. & Hofman, J. D. (1997) An improvisational model for change management: The case of groupware technologies. *Sloan Management Review*, 38 (2). 11.Palmer, I. & Dunford, R. (1997) Conflicting

uses of metaphors: Reconceptualizing their use in the field of organizational change. *The Academy of Management Review*, 21(3). 691

Poppendieck, M. (2001) Lean Programming. *Software Development Magazine*, 9 (5), 71-75.

Rasmusson, J. (2003) Introducing XP into Greenfield Projects: Lessons Learned. *IEEE Software,* 20(3), 21-28.

Rising, L. & Janoff, N. (2000) The Scrum Software Development Process for Small Teams. *IEEE Software,* 17(4), 26-32.

Rogers, E. M. (1995) Diffusion of Innovations, 4th ed. Free Press, New York.

Rosch, E. (2002) Lewin's field theory as situated action in organizational change. *Organization Development Journal*. Summer 2002, http://findarticles.com/p/articles/mi_qa5427/is_200207/ai_n21313515/, last accessed Jan 2010.

Rubin, H. & Rubin, I. (2005) *Qualitative Interviewing: The Art of Hearing Data*, Sage, Thousand Oaks, CA.

Saga, V.K. & Zmud, R.W. (1994) The Nature and Determinants of IT Acceptance, Routinization and Infusion. In *Proceedings of the IFIP TC8 working conference on diffusion, transfer and implementation of information technology*, Levine, L. (Ed.), Amsterdam, North Holland, 1994, 67-86.

Salo, O. & Abrahamsson, P. (2007) An Iterative Improvement Approach for Agile Development: Implications from multiple case study. *Software Process: Improvement and Practice*, 12(1), 81-100.

Schatz, B. and I. Abdelshafi (2005). Primavera Gets Agile: a Successful Transition to Agile Development. *IEEE Software*, 22(3): 36-41.

Schwaber, K. & Beedle, M. (2002) *Agile Software Development with Scrum*, Prentice-Hall, Upper Saddle River, NJ.

Stapleton, J. (1997) *DSDM: Dynamic Systems Development Method*, Addison Wesley, Harlow, England, 1997.

Svensson, H. & Host, M. (2005) Introducing an Agile Process in Software Maintenance and Evolution Organization. In *Proceedings of the Ninth European Conference on Software Maintenance And Reengineering*, Manchester, UK, 256-264.

Trauth, E. & O'Connor, B. (1991) A study of the interaction between information, technology and society. In *Information Systems Research: Contemporary Approaches and Emergent Traditions,* Nissen, H., Klein, H. & Hirschheim, R. (Ed.), Elsevier, North Holland, 131-144.

Vanderburg, G. (2005) A Simple Model of Agile Software Processes - or - Extreme Programming Annealed. *ACM SIGPLAN Notices*, 40 (10), 539-545.

VersionOne (2010). State of Agile Survey, Fifth Anual Survey.

Walsham, G. (1995) Interpretive Case Studies in IS Research: Nature and Method. *European Journal of Information Systems*, 4, 74-81.

Wengraf, T. (2001) *Qualitative Research Interviewing*, Sage, London.

Williams, L., Maximilien, E. M. & Vouk, M. (2003) Test Driven Development as a Defect Reduction Practice. In *Proceedings of the Fourteenth International Symposium on Software Reliability Engineering*, Denver, Colorado, 34-45.

Yin, R. K. (2003) *Case Study Research: Design and Methods*, Saga, Thousand Oaks, California.

Zmud, R. W. & Apple, L. E. (1992) Measuring Technology Incorporation/Infusion. *Journal of Product Innovation Management*, 9(2), 148-155.

Biographies

Xiaofeng Wang is a research fellow in Lero, the Irish software engineering research centre. Her research areas include software development process, methods, agile software development, and complex adaptive systems theory. Her doctoral study investigated the application of complex adaptive systems theory in the research of agile software development. She has also worked in a major IT company in China and a research institute in Italy for several years in the area of enterprise knowledge systems. Her publications include several journal and conference papers in major IS journal and conferences. She can be reached at xiaofeng.wang@lero.ie.

Kieran Conboy is an Associate Professor at the University of New South Wales and Lero research centre in NUI Galway, Ireland. His research focuses on agile systems development approaches as well as agility across other disciplines and Kieran is currently involved in numerous national and international projects in this area. His research has been published inleading journals and conferences such as Information Systems Research, the European Journal of Information Systems, Information & Software Technology, the International Conference in Information Systems (ICIS), and the European Conference in Information Systems (ECIS). He is also associate editor of the European Journal of Information Systems, and has been a guest editor on special issues of top journals on the topics such as qualitative IS research and agility.

Minna Pikkarainen is a Principal Research Scientist in VTT Technical Research Centre of Finland. She has worked in several industrial-driven research projects and project preparations doing close industrial collaboration with large amount of organizations in Europe. Pikkarainen is a member of Sirris, The Collective Center for the Belgian technological industry. Recently, her work and numerous publications have focused on research in the areas of agile development and software innovation.