12 A STUDY OF THE USE OF AGILE METHODS WITHIN INTEL

Brian Fitzgerald
University of Limerick
Limerick, IRELAND

Gerard Hartnett
Intel Communications Europe
Shannon, IRELAND

Abstract *This study investigated the use of the agile methods, eXtreme programming (XP) and Scrum, at the Intel Network Processor Division engineering team based in Shannon, Ireland, over a three-year period. The study is noteworthy as it is based on real industrial software projects involving experienced software engineers, with continuous reflection and monitoring of the application of these approaches. It provides evidence that agile methods are far from anti method; rather, they require disciplined application and careful customization to the particular needs of the development context. The study also shows how XP and Scrum can complement each other to provide a comprehensive agile development method, with XP providing support for technical aspects and Scrum providing support for project planning and tracking. The manner in which XP and Scrum have been customized to suit the needs of the development environment at Intel Shannon is described, as are the lessons learned. The XP practices that were applied did lead to significant benefits, with pair-programming leading to reductions in code defect density of a factor of seven, and one project actually achieving zero defect density. However, some observed limitations of pair-programming are described. Intel Shannon also found that not all XP practices were applicable in their context. Thus, the study suggests that, contrary to suggestions that XP is not divisible or individually selectable, a la carte selection and tailoring of XP practices can work very well. In the case of Scrum, some local customization has led to a very committed adoption by developers themselves, in contrast to many development methods whose use is decreed mandatory by management. The success of Scrum is significant. Projects of six-month and one-year duration have been delivered ahead of schedule, which bodes well for future ability to accurately plan development projects, a black art in software development up to now.*

1 INTRODUCTION

Despite 50 years of software development experience, the vast majority of software projects continue to exceed budget and development schedule, and are often of poor quality when completed. In recent times, agile approaches have emerged as an apparently revolutionary new practice-led paradigm that can address these central problems. The agile approaches comprise a broad range—eXtreme Programming (XP) (Beck 2000), dynamic systems development method (DSDM) (Stapleton 1998), Scrum (Schwaber and Beedle 2002); Crystal (Cockburn 2001); agile modeling (Ambler 2002); feature driven design (Coad et al. 1999); lean programming (Poppendieck 2001), and perhaps even the rational unified process (RUP) (Kruchten 2000), although there is considerable disagreement on whether or not RUP is an agile method. These approaches differ significantly from traditional approaches to software development, emphasizing development productivity rather than process rigor, and seeking to deliver business value quickly, while also accommodating changing user requirements.

It is important to emphasize that agile approaches are not anti method; rather, they operate on the lean principle of "barely sufficient methodology" (Highsmith 2002). The change in emphasis from the traditional approaches is summarized in the following value-tradeoffs:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Advocates of the agile approaches recognize that both sides of these value statements are relevant to software development. However, they choose to emphasize the first part of each statement as more important than the second part. The overall principles underpinning the agile approaches are summarized in the agile manifesto (www.agilemanifesto.com).

The use of agile approaches is growing rapidly, estimated to be in use in two-thirds of all IT development companies in 2002 (Sliwa 2002). Practice is ahead of research in this area, but much of the evidence offered thus far has been anecdotal in nature. Thus, the study reported on here in Intel Shannon is particularly useful as the findings are based on intensive investigation of the agile initiatives that have been implemented. Two of the most popular and widely used agile methods are XP and Scrum, and both of these are in active use in Intel Shannon. Hence, a brief background summary of each of these approaches is provided here.

1.1 eXtreme Programming (XP)

The eXtreme Programming (XP) approach explicitly acknowledges that it is not a magic "silver bullet" of revolutionary new techniques; rather, it is a set of tried and trusted principles that are well-established as part of the conventional wisdom of software engineering, but which are taken to an extreme level—hence the name eXtreme

Programming. XP has been pioneered by Kent Beck, and has its origins in a project to develop an internal payroll system at Chrysler in 1996-97. It is comprehensively described in Beck (2000, p. xv), where he describes it as "a light-weight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly-changing requirements." XP comprises five key values, *communication, feedback, simplicity, courage*, and *respect*. These are underpinned by 12 key practices, summarized in Table 1.

A marked feature of XP is that several of the practices overlap to some extent and thus serve to complement and reinforce each other—refactoring, simple design, collective ownership, and coding standards, for example. However, while XP is acknowledged as not being a "one size fits all" approach suited to every development context, there is by no means unanimous agreement on where the limits of its applicability lie. Thus, its application in Intel Shannon is especially pertinent as it represents an industrial product development setting with experienced software engineers. Many of the reported benefits of XP to date have been in academic university environments (e.g., Hedin et al. 2003; Muller and Tichy 2001) and, therefore, lessons learned from its application in a real software development context are invaluable, as quite few such studies have been published (Heim and Hemphill 2003). Also, McBreen (2003, p. 88) identifies the importance of "continuous reflection" on the application of XP practices, and this was very much a feature of the Intel Shannon context.

1.2 Scrum

Scrum (Schwaber and Beedle 2002) is a simple, low-overhead process for managing and tracking software development. While it is very much influenced by Boehm's (1988) spiral model, it has its origins in a project by Jeff Sutherland at the Easel Corporation in 1993 where it was used in the development of an object-oriented analysis and design tool. While XP is used in Intel Shannon for the technical engineering aspects of development, Scrum is used for the project management aspects, for which it is better suited. Scrum differs from traditional approaches in that it assumes that analysis, design, and development processes are largely unpredictable. At its heart, Scrum comprises a number of stages which, building on its underpinning metaphor of a rugby scrum, also follow a sporting theme.

- First, the **pre-game** phases:
  - **Planning**: This phase involves the definition of a new release of the system based on the currently known backlog of required modifications, along with an estimate of its schedule and cost. If a new system is being developed, this phase consists of both conceptualization and analysis. If an existing system is being enhanced, this phase consists of limited analysis.
  - **Architecture**: This phase includes system architecture modification and high-level design as to how the backlog items will be implemented.

Table 1. Key Practices of XP (Adapted from K. Beck, *Extreme Programming Explained*, Addison-Wesley, 2000)

The Planning Game
A quick determination of the scope of the next software release, based on a combination of business priorities and technical estimates. It is accepted that this plan will probably change.

Small Releases
Put a simple system into production quickly, then release new versions on a very short cycle.

Metaphor
Guide all development with a simple shared story of how the whole system works.

Simple Design
The system should be designed as simply as possible at any given moment in time.

Testing
Programmers continually write tests which must be run flawlessly for development to proceed. Customers write function tests to demonstrate the features implemented.

Refactoring
Programmers restructure the system, without removing functionality, to improve nonfunctional aspects (e.g., duplication of code, simplicity, flexibility).

Pair-Programming
All production code is written by two programmers at one machine.

Collective Ownership
Anyone can change any code anywhere in the system at any time.

Continuous Integration
Integrate and build the system every time a task is completed—this may be many times per day.

40-Hour Week
Work no more than 40 hours per week as a rule.

On-Site Customers
Include an actual user on the team, available full-time to answer questions.

Coding Standards
Adherence to coding rules which emphasize communication via program code.

- Following this is the main **game** phase:
  - **Sprints**: This involves development of new release functionality, with constant respect to the variables of time, requirements, quality, cost, and competition. Interaction with these variables defines the end of this phase. There are multiple, iterative development sprints, or cycles, that are used to evolve the system.

- Finally, there is the **post-game** phase:
  - **Closure:** Here the focus is on preparation for release, including final documentation, pre-release staged testing, and release.

The first and last Scrum phases (planning and closure) consist of defined processes, where all processes, inputs, and outputs are well defined. The knowledge of how to do these processes is explicit. The flow is linear, with some iteration in the planning phase.

Sprints are nonlinear and flexible. Where available, explicit process knowledge is used; otherwise tacit knowledge and trial and error is used to build process knowledge. Sprints are used to evolve the final product. The project is open to the environment until the closure phase. The deliverable can be changed at any time during the planning and sprint phases of the project. The project remains open to environmental complexity, including competitive, time, quality, and financial pressures, throughout these phases.

One of the most interesting aspects of Scrum is the daily meeting of the project team. The daily meeting is kept short, typically 15 minutes. Everyone answers three questions.

- What did you do in the last 24 hours?
- What roadblocks did you encounter that you need someone to remove?
- What is your plan for the next 24 hours?

Within Intel Shannon, quite a lot of experimentation has been done using Scrum on projects of different sizes and complexity. Despite the claim by its proponents that Scrum has been used on "thousands of Scrum projects" (Schwaber and Beedle 2002), there have been few accounts of the use of Scrum in real-world projects (Abrahamsson et al. 2003), a notable exception being the study by Rising and Janoff (2000).

The remainder of the paper is structured as follows: In the next section, contextual background information is provided in relation to Intel Shannon. Following this, the case study research method and the personal interview process employed in this study is discussed. In the next section, the actual implementation of XP and Scrum and the lessons learned are discussed. Finally, the conclusions from the study are presented.

2 BACKGROUND: INTEL SHANNON DEVELOPMENT CONTEXT

Intel Shannon is based in the west of Ireland and is part of the Intel's Infrastructure Processor Division. The main Intel plant in Ireland near Dublin employs 4,200 people.

The Intel Shannon organization employs close to 100 people, and about 70 are involved in engineering, software development, and silicon design. The products under development are network processors for networking equipment typically for SMEs, the small office/home (SOHO), and 3G wireless markets. For these products, requirements analysis is typically done in the United States, the software and silicon design is done in Shannon. Intel Shannon has seen significant growth in their workforce over the past few years. They are now striving to institute a repeatable engineering process whereby they will have multiple products under development in parallel in different phases. In the past, their portfolio has been characterized by a startup/single-product focus.

In terms of software development, Intel Shannon has been formally assessed at Level 2 on the capability maturity model (CMM). While this has led to some discipline in the development process, the rapid time-to-market pressures have led Intel Shannon to consider agile methods. Further, they are a company that embraces innovation and seeks to rigorously assess new techniques and methods that could meet their market needs. Intel Shannon has been deploying a range of agile methods over the past three years, principally two flavors of agile methods: XP for the technical engineering aspects of software development and SCRUM for the project planning and tracking.

While the move to CMM certification was driven more as a top-down mandate within the organization, in contrast, Scrum and XP were introduced at a grassroots engineering level as optional techniques. As such, their adoption has grown organically over time. They were not mandated or compulsory as the techniques were being introduced in parallel with CMM implementation. While many tend to view CMM and agile methods as axiomatically incommensurable, this has been cogently shown to be an oversimplification (Paulk 2001).

Agile methods are also finding use in the wider Intel software engineering community. The company now has an internal wiki Web site and diverse teams meet on a regular basis to share experiences with different agile methodologies. Again, this community is driven by grassroots engineering.

The lessons learned have been significant and are discussed in section 4, but first the research method employed in this study is described.

3 RESEARCH METHOD

Given that the agile methods area is a relatively new research area, research of an exploratory and descriptive nature is needed, and any research method chosen should reflect this. Marshall and Rossman (1989) propose a framework for matching research purpose with research methods and data capture techniques. In the case of research which has a descriptive and exploratory focus, a combination of case study and in-depth interviewing is deemed appropriate according to their framework.

3.1 The Case Study Method

The case study is not viewed in a similar fashion by all researchers (see Smith 1990). However, according to one of the more common interpretations, it describes a

single situation, and usually involves the collection of a large amount of qualitative information (see Benbasat et al. 1987; Lee 1989; Yin 1994). Case studies can be very valuable in generating an understanding of the reality of a particular situation, and can provide a good basis for discussion. There is neither an attempt at experimental design nor any control of variables. However, since the information collected is often specific to the particular situation at a particular point in time, results may not be generalizable.

Notwithstanding this limitation, the case study was chosen as the research method for this study, as its advantage in providing thick description was seen as outweighing its limitations. Also, the project manager responsible for the deployment of agile methods subsequently became a coauthor of the paper. Thus, the findings are further strengthened through the direct validation of those responsible for the process being studied.

3.2 In-Depth Personal Interviews

The purpose of the personal interview is to encourage the interviewee to relate experiences and attitudes relevant to the research problem (Walker 1988). It is a very flexible technique in that the interviewer can probe any interesting details that emerge during the interview, and concentrate in detail on particular aspects.

It should be noted that a reflexive approach was deliberately allowed in the interview phase adopted in this study. This has been identified as important in exploratory research (Trauth and O'Connor 1991) as it allows for refocusing as the research progresses, in that responses to certain questions can stimulate new awareness and interest in particular issues which may then require additional probing. Eisenhardt (1989) also recommends such a strategy, labeling it *controlled opportunism*.

In this study, a series of formal and informal interviews were conducted over a one-year period with the project manager and key staff responsible for agile deployment at Intel Shannon. Interviews were generally of one- to two-hour duration. Informal interviews were used to clarify and refine issues as they emerged. Also, as one of the primary sources of information became a coauthor of the paper, the correctness of the researchers' interpretation was less of an issue than in the traditional model whereby exclusively external authors interpret the research findings.

4 USE OF XP AND SCRUM AT INTEL SHANNON

4.1 XP

Intel Shannon has been using XP for five years. However, even though they have been committed users of XP, they have been quite pragmatic in choosing only those aspects of XP which they perceived as relevant to the needs of their development context. The XP practices that have been deployed, however, have been carefully monitored and the implications measured. These practices were pair-programming, testing, refactoring, simple design, coding standards, and collective ownership. Their experiences with each are discussed in turn below.

Scrum has also been used for five years. Again, the documented technique has been tailored locally.

Scrum has seen more enthusiastic adoption at the individual team level than eXtreme Programming. The reasons for this are discussed in more detail below.

4.1.1 Pair-Programming

Pair-programming is perhaps the best known of the XP practices, with generally positive reports on its usage, although Muller and Tichy (2001) suggest that it decreases overall productivity. While most of the other XP practices have been applied across all of the individual software teams at Intel Shannon, pair-programming has been selectively applied. Most teams consist of between two and six software engineers with a wide range of experience. Pair-programming was applied initially by two teams on two components of the software for the IXP2XX network processor. On the later IXP4XX network processor, it was again employed by two teams.

Pair-programming was perceived as having a number of significant advantages at Intel Shannon. First, it was estimated that the required code quality level was achieved earlier. On the IXP2XX project, the pair-programmed components had the lowest defect density in the whole product. The defect densities were a factor of seven below the component with the highest density. On the IXP4XX project, two of the three Intel Shannon based teams used pair-programming. One of the teams achieved zero defect quality. The team with the highest defect density was the team that did not. The three teams all had similar experience profiles. With pair-programming, developers did not get stuck wondering what to do next. If one person was unsure, the other probably did know. Developers also believed that they learned quite a lot from each other and that they remained more focused on the job at hand, and less likely to go off on a tangent.

The essential nature of pair-programming, where one person is effectively looking over the other's shoulder, meant that minor errors were caught early, saving considerable debugging time. Also, it was useful for testing and debugging, as a fresh viewpoint could spot the obvious flaw which was not obvious to the pair partner. The overall process also ensured that more than one developer gained a deep understanding of the design and code, thus facilitating collective ownership (discussed below). Developers suggested that they had more fun, and found the work more interesting. They also seemed more enthusiastic about their work.

However, there were a number of problematic aspects associated with the use of pair-programming also. For example, it was found to be unsuitable for simple or well-understood problems, which could be fixed as quickly as a single developer could type. In a similar vein, when doing lots of small changes (e.g., eliminating To-Do's), it tended to get frustrating.

Some developers found pair-programming could break their flow of concentration as they needed to pause to communicate nonobvious ideas to the pair partner. Indeed, some developers expressed the view that it was difficult to reflect and concentrate with someone by their side.

Overall, Intel Shannon has documented a number of lessons which will guide its future use of pair-programming.

- Some basic rules of pair working etiquette are required, e.g., no keyboard wrestling.
- Consideration needs to be given to neighbors to keep background noise to a minimum.
- Use large fonts.
- Set clear objectives at the start of a programming session.
- Planning and coordination may be necessary to prioritize programming over other activities (e.g., helping other engineers, phone calls, meetings), otherwise both people may not be free simultaneously.
- Pair-programming was not seen as valuable during sustaining activities on the project when the amount of coding is not as significant.

4.1.2 Testing

Intel Shannon also implemented a test-code development strategy (i.e., writing the unit-test code while writing production code). They found this had a number of advantages. It set a direction for the immediate development, namely to get the test case working. It also helped developers get a better understanding of the functionality required of the software from a client point of view. The unit-tests are also implemented as part of a regression test suite and all component unit tests are run on the code repository nightly. Integration tests are also developed to test the individual components in concert and "smoke tests" are run daily with external test equipment in the weeks leading up to a release.

4.1.3 Refactoring

Refactoring was another XP technique that was quite widely used at Intel Shannon. They found it worked best when it was done early, as it eliminated a lot of bugs that would have taken up a lot of debugging time otherwise. Refactoring also became akin to a continuous design activity, which is discussed next.

4.1.4 Simple Design

In this case, design was done on a whiteboard before each block of code was written. As a result, the design document emerged on an ongoing basis in parallel with the code implementation. Quite significantly, however, they have not subscribed to the XP concept of the code being the design as documentation is an integral part of the product deliverable at Intel Shannon. Simplicity increasingly became the guiding principle and, over time, developers stopped trying to second-guess the client code and just implemented the requirements. As already mentioned, this practice was very closely linked to refactoring.

4.1.5 Collective Ownership

This practice led to a number of benefits. First, it ensured that several members of the project team knew the code well enough to make changes, so if one person was busy, another person could make the requested change. Also, in the Intel Shannon context, changes in team composition were quite common. In the past, this meant that developers had to choose between bringing any code they wrote with them and continuing to maintain it, or spending time teaching the code to someone else and handing over responsibility. Collective ownership allowed management more flexibility as it resulted in teams being able to maintain the code base as several of the original members would know it well enough to maintain it.

However, Intel Shannon found that collective ownership was only appropriate on a single team basis. Code ownership across multiple teams was not applied. The software engineering team on the whole product could be as many as 30 engineers and the team felt collective ownership could not scale to this wide a population.

4.1.6 Coding Standards

Intel Shannon defined a C-coding standard early in the project and referred to it extensively during coding and code inspections. Coding standards were already a very strong feature of their development environment prior to the application of XP.

4.1.7 Unused XP Practices

XP pioneers have suggested that it cannot be applied with piecemeal cherry-picking of individual practices. As Schwaber (2001, p. 8) puts it, "[XP] values and their underlying practices and techniques are not divisible and individually selectable; they form a coherent, whole process." However, a number of XP practices were not applied at Intel Shannon as they felt they were not applicable to their development context. The unused practices include the planning game, small releases, continuous integration, 40-hour week, metaphor, and on-site customers. The reasons for lack of adoption of these practices were as follows:

The planning game was not used as many aspects of planning are covered by the Scrum technique, discussed later. From a business priority perspective, a product-marketing team has the responsibility for deciding feature priorities. They are in a separate organization, most of whom are not physically colocated. In future, however, they intend to use some prioritization aspects of the planning game.

The XP practice of small releases is not feasible early in the product schedule as in this business the software releases are tied to silicon availability. Once silicon is available, the team typically delivers minor releases every four to six weeks and major releases every two quarters.

While continuous integration is practiced for each component, given the complexity of the overall software and the need for external test equipment, full system integration is done only in the fortnight leading up to a release.

The 40-hour week was seen as a great aspiration but it was not consistently achievable in the Intel Shannon development context, where the discrepancy in time zones between Europe and the United States serves to extend working hours.

On-site customers are not available. These projects are tied to the design of silicon and in many cases do not have specific customers during the early conceptual stages. The product marketing group acts as a customer proxy, prioritizing features based on potential revenue.

Metaphor was not explicitly used, but at a high level the software components do correspond to the interfaces on the silicon and have common patterns of functions on the APIs.

4.1.8 Overall Lessons on XP Practices

Overall, Intel Shannon is quite happy with the XP experience. Some of the practices, such as simple design and testing, are now used across the board on all development teams. Testing is also integrated into the development environment.

Despite its success, pair-programming has not grown to the same extent as Scrum, for example. This dichotomy will be discussed below.

In general, where pair-programming was adopted, it tended to lead to a smaller code base and, as defect rate is directly correlated with code length, this has led to more efficient use of resources.

As a thought experiment, the developers tried to imagine how the software would have turned out if a more traditional development process had been followed. They believed it would have taken in or around the same time—any discrepancies would be lost in the noise of overhead. However, they felt the traditional code would probably have been quite a bit more complex and long to cater for situations that would probably never occur. As mentioned above, since the defect rate is a constant, this would equate to more bugs.

4.2 Scrum

Scrum has been used for three years at Intel Shannon although some of the engineers had used it for almost five years in their previous organizations. Scrum has really only been documented in book form since 2002 (Schwaber and Beedle 2002). Up to then the technique was documented on a number of Web sites (e.g., http://www.jeffsutherland.org/scrum/index.html and http://www.controlchaos.com/scrum.pdf). The Intel team also employed a number of techniques from EPISODES (Cunningham 1995), the precursor to eXtreme planning.

Scrum was initially piloted by one team and its use has grown organically to the extent that it now is used by most of the teams in Intel Shannon. They believe the key reason for this enthusiastic embrace of the technique is due to one of the customizations this initial team made. The daily Scrum meeting took place around a board covered with yellow post-it notes. The team recorded tasks for the 24-hour period on post-its. This made Scrum very visible in the organization, and curiosity from other teams helped the

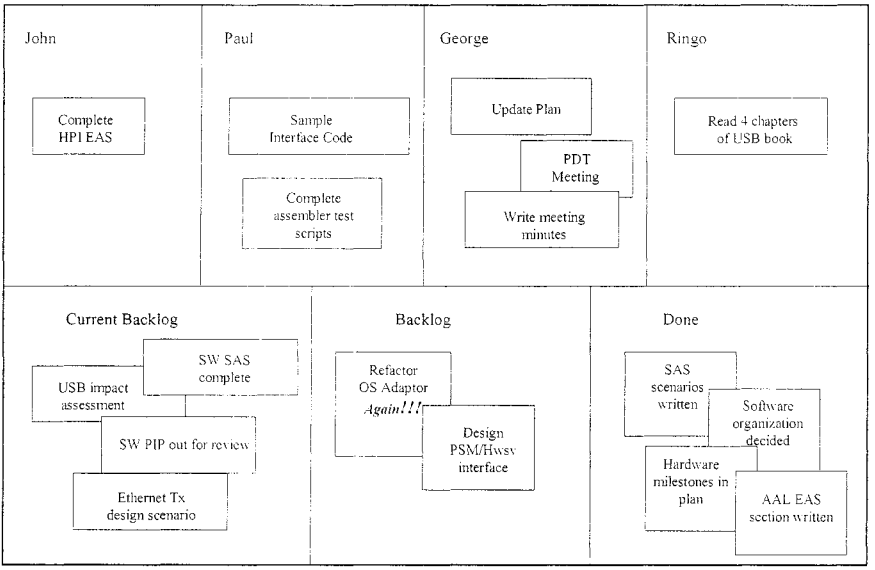

Figure 1. Sample Scrum Daily Meeting Post-It Record

initial spread of the technique. Figure 1 illustrates a sample meeting record with post-its attached.

Team members arrive at the daily meeting with their new post-its for the next 24 hours. The post-its in their named area are the tasks that were committed to at the last meeting. If a task is too big for the next 24 hours, they write a subset of it on a new post-it. During the Scrum meeting, the team members move completed tasks into the "done" area. Moving the post-its around helps achieve a shared group visualization of the tasks and project progress.

They have also experimented with other innovative practices. For example, one team member took notes and then published the tasks on a Web page. However, they found this was a significant overhead for that team. They also tried running the meeting with each individual taking notes in a personal notebook, but this reduced the shared group visualization of the project. Overall they found the shared post-it board the most useful.

The post-its encourage people to prepare more thoroughly in advance for the daily meeting. Continuous preparation happens as developers stick new post-its to their PC screens during their work in the interim between daily meetings.

Until recently, all teams were geographically colocated so the simple low-tech post-it technique has worked very well. Interestingly, they now have one distributed team, which has commenced using the technique by employing a shared spreadsheet and networked meeting software. It is too early to report on the results of this project, but early indications are promising, thus indicating that some agile methods may be more applicable to distributed development than has been suggested up to now (McBreen 2003).

effective number of engineers		2.6	2.6	2.7	2.8	2.6	2.8	
Sprint Tasks	Baseline	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Unassigned
Atmd.HLD.workshops	12.0							12.0
Aal5/Aal0/Aal2.workshops	2.0							2.0
Fpath	64.5							64.5
QMgr	43.5							43.5
Atmd	111.5							111.5
Aal5Acc.FuncSpec.Draft/Review	10.0							10.0
Aal5Acc.DesignSpec	7.0							7.0
Aal5Acc.Code	6.0							6.0
Aal5Codelet.Spec	7.5							7.5
Aal5Codelet.Code	7.0							7.0
Total	271	0	0	0	0	0	0	271.0
Fixed Overhead								
Holidays.Public	4.0							4.0
Holidays.Vacation	45.0							45.0
Training	4.0							4.0
Total	53	0	0	0	0	0	0	53.0
		0	0	0	0	0	0	

Figure 2. Scrum Planning

4.2.1 Scrum Planning

Intel Shannon has made some modifications to the planning process as well. They use two planning stages, one at the start of each sprint and one at the start of the project.

Planning is kept simple. There is no complex Gantt chart with complex inter-dependencies between tasks. The overall plan is a series of sprints (see Figure 2). Internal or external milestones can be lined up with sprint completions, but the dependencies between the tasks within the sprint are not worked out in advance.

Each team lead does a plan outlining all of the sprints to the end of the project. Initial meetings are conducted by the engineers to get high-level estimates that can be allocated and distributed across a number of sprints. In one of the projects, the wide-band Delphi technique was used to generate the estimates (Linstone and Turoff 1975). Dependencies between teams are made between end-of-sprint milestones.

In terms of deliverables, the team lead provides a list of sprint milestones and the contents of each sprint to the overall project lead.

Intel Shannon does not use sprint time boxing which is part of some implementations of Scrum. The high-level tasks are split to distribute them across sprints. They then continue to distribute and split tasks until the duration of each sprint is at most 20 working days. Contingency is built into the plan and effort estimates are done based on ideal engineering effort. The contingency factor is tuned as the project progresses.

At the start of each sprint, the team decides which tasks are going to be done in the next sprint. They look at the start of project sprint plan and look at any new backlog items that may have come up during the last sprint. Tasks are allocated to individuals to spread the load. The sprint protects the team from the environment surrounding it for a meaningful amount of time.

At the end of the sprint, the team lead writes a wrap-up report, listing the tasks completed including extra tasks that were not part of the original sprint plan. The report will also contain lessons learned and a measurement of the actual effort expended in the sprint versus the estimate at the start-of-project. Other end-of-sprint deliverables could include a demo, a project review, or a release.

4.2.2 Overall Lessons on Scrum

Project teams have had excellent success delivering projects on time and within budget. An early project of 5.5 months duration with four team members delivered their final release within three days of the original plan. The IXP4XX release 1.0 software was delivered one week ahead of schedule on a project with an original planned duration of over a year. The team consisted of 5 teams and over 30 engineers. All teams used Scrum.

The key advantages of Scrum that the team observed were

- Planning and tracking become a collaboration involving the whole team
- Excellent communication builds up within the team, thus building morale and helping the team to gel
- The team lead has more bandwidth for technical work
- Enables the team to deliver on-time

The early adoption of Scrum has led to the formulation of internal training courses and in short time the use of Scrum has reached critical mass. In the case of XP, pair-programming was not as visible and did not reach the same critical mass. In general, most of the engineers acknowledge the utility and advantages of pair-programming but are still slow to apply it. They are not making a conscious decision not to use it and maybe the technique needs some renewed internal promotion.

Another possible factor limiting the spontaneous adoption of pair-programming at the individual engineer level may be the perception that individual ownership of code components is of more value when performance reviews are being evaluated.

5 CONCLUSIONS

Overall, there are many lessons from this research at Intel Shannon. The study is useful in being solidly based on the rigorous and disciplined implementation of agile approaches in a real development context involving experienced software engineers, with a careful reflection on subsequent results. The study confirms that both XP and Scrum have merit and are very complementary in that XP provides good support for the more technical aspects of development while Scrum provides a very good framework for project planning and tracking. Also, it is clear that these approaches are not anti method but require a disciplined approach and indeed need to be tailored to the needs of the development context. Notwithstanding this, developers themselves have embraced these techniques and use has grown over time, in stark contrast to many organizations where the use of development methods is mandated by management, which leads to far less actual usage of these methods (Fitzgerald 1998).

Intel Shannon did not find that all of the XP practices were applicable in their context. Pair-programming, testing, refactoring, simple design, coding standards, and collective ownership were all applied to good effect. However, while they found pair-programming to have significant benefits, in terms of code quality for example, its use is not increasing, but this may be explained by the need for other management support mechanisms to support its use. Several XP practices were not considered applicable, such as the planning game, small releases, continuous integration, metaphor, on-site

customer, and 40-hour week. While XP advocates reasonably point to the fact that the practices form a coherent whole, this does not mean that selective relevant practices cannot be applied to good effect. Intel Shannon certainly derived value from a subset of the practices. Also of interest is the fact that the XP principle that the code is the documentation did not feature at Intel Shannon since documentation is an integral part of the product deliverable.

Intel Shannon has also achieved significant benefits through the use of Scrum. Again, they have adapted it very much to their needs with the highly visible daily meeting report. Also, the use of Scrum has led to consistent meeting of development schedules on very complex projects with long project durations, but with no degradation in product quality. Scrum has been more robust than XP over time, when sustained on just grassroots engineering sponsorship. Finally, the deployment of Scrum on a distributed development project suggests that some agile approaches may be more amenable to distributed development than has been assumed up to now. This will be the focus of further study.

REFERENCES

Abrahamsson, P., Warsta, J., Siponen, M., and Ronkainen, J. "New Directions on Agile Methods: A Comparative Analysis," in *Proceedings of 25th International Conference on Software Engineering*, New York: ACM Press, 2003, pp. 244-254.

Ambler, S. *Agile Modeling: Effective Processes for Extreme Programming and the Unified Process*, New York: Wiley & Sons, 2002.

Beck, K. *Extreme Programming Explained*, Upper Saddle River, NJ: Addison-Wesley, 2000.

Benbasat, I., Goldstein, D., and Mead, M. "The Case Research Strategy in Studies of Information Systems," *MIS Quarterly* (11:3), September 1987, pp. 369-386.

Boehm, B. "A Spiral Model of Software Development and Maintenance," *IEEE Computer* (21:5), 1988, pp. 61-72.

Coad, P., Lefebvre, E., and DeLuca, J. *Java Modeling in Color with UML*, Upper Saddle River, NJ: Prentice Hall, 1999.

Cockburn, A. "Crystal Light Methods," *Cutter IT Journal*, 2001 (available online at http://alistair.cockburn.us/crystal/articles/clm/crystallightmethods.htm).

Cunningham, W. "EPISODES: A Pattern Language of Competitive Development Part I," 1995 (available online at http://c2.com/ppr/episodes.html).

Eisenhardt, K. "Building Theory from Case Study Research," *Academy of Management Review* (14:4), 1989, pp. 532-550.

Fitzgerald, B. "An Empirical Investigation into the Adoption of Systems Development Methodologies," *Information and Management* (34), 1998, pp. pp. 317-328.

Hedin, G., Bendix, L., and Magnusson, B. "Introducing Software Engineering by Means of Extreme Programming," in *Proceedings of 25th International Conference on Software Engineering*, New York: ACM Press, 2003, pp. 586-593.

Heim, C., and Hemhill, D. "Extreme Programming and Scrum: A Local Experience Report from Gearworks," Object Technology Users Group, St. Paul, MN, October 2003 (available online at http://www.otug.org/meeting/200310.html).

Highsmith, J. "Does Agility Work?," *Software Development*, June 2002, pp. 28-36.

Kruchten, P. *Rational Unified Process: An Introduction*, Reading, MA: Addison-Wesley, 2000.

Lee, A. "A Scientific Methodology for MIS Case Studies," *MIS Quarterly* (13:1), 1989, pp. 33-50.

Linstone, H., and Turoff, M. *The Delphi Method: Techniques and Applications*, Reading, MA: Addison-Wesley, 1975.

Marshall, C., and Rossman, G. *Designing Qualitative Research*, Thousand Oaks, CA: Sage Publications, 1989.

McBreen, P. *Questioning Extreme Programming*, Boston: Addison-Wesley, 2003.

Muller, M., and Tichy, W. "Extreme Programming in a University Environment," in *Proceedings of 23rd International Conference on Software Engineering*, New York: ACM Press, 2001, pp. 537-544.

Paulk, M. "Extreme Programming from a CMM Perspective," *IEEE Software* (18:6), November-December 2001, pp. 19-26.

Poppendieck, M. "Lean Programming," *Software Development*, June 2001, pp. 71-75.

Rising, L., and Janoff, N. "The Scrum Software Development Process for Small Teams," *IEEE Software* (17:4), July-August 2000, pp. 26-32.

Schwaber, K. "Will the Real Agile Processes Please Stand Up", *Cutter Consortium Executive Report* (2:8), 2001, pp. 1-22.

Schwaber, K., and Beedle, J. *Agile Software Development with Scrum*, Upper Saddle River, NJ: Prentice Hall, 2002.

Sliwa, C. "Agile Programming Techniques Spark Interest," *ComputerWorld*, March 14, 2002 (available online at http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,69079,00.html).

Smith, N. "The Case Study: A Useful Research Method for Information Management," *Journal of Information Technology* (5), 1990, pp. 123-133.

Stapleton, J. *Dynamic Systems Development Method: The Method in Practice*, Reading, MA: Addison-Wesley, 1998.

Trauth, E., and O'Connor, B. "A Study of the Interaction Between Information, Technology and Society," in *Information Systems Research: Contemporary Approaches and Emergent Traditions*, H-E. Nissen, H. K. Klein, and R. A. Hirschheim (Eds.), Amsterdam: Elsevier Publishers, 1991, pp. 131-144.

Walker, R. *Applied Qualitative Research*, Hampshire, England: Gower, 1994.

Yin, R. *Case Study Research: Design and Methods* (2nd ed.), Thousand Oaks, CA: Sage Publications, CA, 1994.

ABOUT THE AUTHORS

Brian Fitzgerald holds the Frederick A Krehbiel II Chair in Innovation in Global Business and Technology at the University of Limerick, Ireland, where he also is a Research Fellow of the University, in addition to being a Science Foundation Ireland Investigator. He has a Ph.D. from the University of London and has held positions at University College Cork, Ireland, Northern Illinois University, United States, the University of Gothenburg, Sweden, and Northumbria University, United Kingdom. His publications include seven books and more than 70 papers, published in leading international conferences and journals. Having worked in industry prior to taking up an academic position, he has more than 20 years experience in the IS field. Brian can be reached at bf@ul.ie.

Gerard Hartnett is a software architect in Intel's Infrastructure Processor Division focusing primarily on new products. He has been with Intel for five years, in which time he has focused on architecture and engineering management on Intel's entry-level and mid-range network processors. He is coauthor of the forthcoming book, *Designing Embedded Networking Applications*, from Intel Press. Prior to Intel, he worked for many years in the telecommunications industry on GSM, ATM, voice, and network management products. He has an M.Sc. from University College Cork and a B.Eng. from the University of Limerick. Gerard can be reached at gerard.hartnett@intel.com.