Practical Experiences of Agility in the Telecom Industry

Jari Vanhanen[1], Jouni Jartti[2], and Tuomo Kähkönen[2]

[1] Helsinki University of Technology, Software Business and Engineering Institute,
P.O. Box 9600, FIN-02015 HUT, Finland
jari.vanhanen@hut.fi

[2] Nokia Research Center, P.O. Box 407, FIN-00045 NOKIA GROUP, Finland
{jouni.jartti, tuomo.kahkonen}@nokia.com

Abstract. This paper discusses the adoption level of and experiences from using agile practices in three software development projects in a large Telecom company. The use of agile practices was more emergent than planned. Project managers and developers simply used practices they considered efficient and natural. The most widely adopted agile practices were to measure progress by working code, to have developers estimate task efforts, to use coding standards, having no continuous overtime, to have the team develop its own processes, to use limited documentation, and to have the team in one physical location. The projects used conventional testing approaches. Adoption of agile testing practices, i.e., test first and automated unit tests, was low. Some agile practices can just emerge without conscious adoption, because developers find them useful. However, it seems that an emergent process aiming for agility may also neglect important agile practices.

1 Introduction

The competitiveness of IT companies is affected by how well their software development process can react to changing needs set for products [1,2]. We define agility as the ability to adapt to changing situations appropriately, quickly and effectively. In other words, agile organizations notice relevant changes early, initiate action promptly, create a feasible and effective alternative plan quickly, and reorient work and resources according to the new plan quickly and effectively.

In the Telecom industry software development has traditionally followed rather rigorous processes, typically using process frameworks such as ISO-15504 [3] or the SW-CMM [4]. As far as we know, no studies have been published discussing the use of agile methods in the Telecom industry. Thus, we believe that this paper sheds some light on the current situation.

1.1 Study Background

This study was made in a large telecom company as part of a research program, whose goal is to increase and transfer practical knowledge of achieving agility in software development. Creating a full-fledged methodology and trying to leverage it in a large organization is not feasible. However, describing a set of process patterns [5] that promote agility is more feasible. A process pattern describes a practice considering topics such as which problem it solves, when it is applicable, how to deploy it etc. McCormick's [6] ideas for creating a methodology for a project also support the process patterns approach: "What's needed is not a single software methodology, but a rich toolkit of process patterns and 'methodology components' (deliverables, techniques, process flows, and so forth) along with guidelines for how to plug them together to customize a methodology for any given project." Of course, the ideal content of an agile toolkit depends on the context, and the limits of agile practices are still unclear.

Before the study, we had collected a tentative list (Table 3) of agile practices that could be applicable in the company. Most of the practices are described in XP [7] and the rest in other literature [8,9]. This specific study aimed at increasing our understanding of the current level of use of these and potentially some other agile practices in the company. Based on the results of this study we will evaluate and improve the practicality and completeness of our tentative agile practice list.

The research questions for this study were:

1. Which agile practices were used in these projects?
2. What experiences were reported on those practices?

1.2 Research Method

Using the company's intranet and personal relationships, we identified several projects from the case company in Finland that were either consciously or non-consciously using agile software development practices. However, we could not identify any project that was using a complete agile methodology such as XP. From seven identified candidates we selected three projects, A, B, and C, which seemed to be most active in using agile practices. All selected projects were developing different kinds of products in different business units, and for different markets.

We interviewed the project manager and a developer from projects A and B, and only a developer from project C. The interviews covered all typical software project areas at a general level in order to identify other agile practices in addition to those we had already listed.

We quantified the level of use of the agile practices (Table 1) in order to better answer the first research question. Quantification was difficult because there are several dimensions that should be considered, e.g., number of people using the practice, duration of use, discipline of use, and aspects used. The second research question was answered by a qualitative analysis of the reported experiences.

Table 1. The quantification scale for the use of agile practices

Value	Description
3	Considerable use, e.g. it was almost a norm to use it.
2	Moderate use, e.g. several people used it for a long time.
1	Minor use, e.g. someone tried the practice for some time.
0	Practically no use of the practice.

1.3 Overview of the Projects

Project A developed a Unix based application. The mindset of the project's first two developers was towards light practices and because the project grew slowly over time, they had time to find the natural, minimal process. The success of the product and the small number of developers allowed them to keep the process as they wanted despite of the external pressure to change it to comply better with the organization's general process model. Project B developed low-level embedded software. The project manager decided the practices used in the project based on his previous experiences. Project C was a release project in which a small team worked in a larger 70-person sub-project that was part of the further development of a very large legacy system. The team tried to improve the current process by deploying some aspects of certain XP practices. Table 2 summarizes the characteristics of these projects.

Table 2. Project characteristics

	Project A	Project B	Project C
Project type	Development of an evolving sw product	Integration and porting of embedded sw	Development of a part of a larger system
People	1 → 12	8	4 (of 70)
Distribution	Two teams in two countries	Co-located team	Co-located team
Duration	8 years	10 months	1.5 years
Effort	~50 man years	~6 man years	~6 man years
SW Size	590 kLOC	15 kLOC	40 kLOC

2 Results

2.1 Adoption of Agile Practices

The use of agile practices was more emergent than planned. Typically, the process or the use of individual practices was not formally defined or documented, but rather the project manager and the developers used practices that they considered efficient and natural. XP was partially experimented in project C, but others did not use any documented agile methodology as the basis for their process. The motivation for using agile practices was either experimenting with something new, e.g. some XP practices, or just working in a way that felt natural and had worked earlier in other

projects. Table 3 presents the adoption level of agile practices and the following sections describe the experiences.

Table 3. Adoption level of agile practices

Practice	A	B	C
Incremental delivery	2	0	1
Continuous integration	2	0	2
Measure progress by working code	3	3	2
Interactive planning	2	0	1
Developers estimate task efforts	2	2	3
Visual modeling	1	2	2
Use cases	0	0	3
Design Patterns	2	0	0
Continuously developed architecture	2	1	1
Pair programming	0	0	1
Collective code ownership	2	1	2
Coding standard	3	1	3
Refactoring	2	2	1
Write tests first	0	0	0
Automated unit testing	0	0	0
Customer writes acceptance tests	1	0	0
Limited documentation	3	2	1
Team in one location	0	3	3
Frequent team meetings	1	1	2
Customer always available	2	2	0
Team develops its processes	3	2	1
No continuous overtime	3	2	1

3 considerable use
2 moderate use
1 minor use
0 practically no use

2.2 Use of the Practices

Incremental Delivery. Project A had a release cycle of about 6 months. Later in the cycle, pre-releases were delivered weekly to the customer. Project B planned the first release to occur 10 months after the project initiation, but the project was cancelled due to business reasons before the first release. Project C had a 6-month release cycle.

Continuous Integration. In project A, new code was checked in to the common repository as often as reasonable, typically after a few days of development. The code had to work before the check-in. In project B, developers first implemented the subsystems separately, followed by an integration phase. In project C, developers integrated their code after a few days of work.

Measure Progress by Working Code. In project A, the current version was delivered weekly to the customer. Users tried the pre-releases and gave valuable feedback replacing the need for a detailed requirement specification. Users were committed thanks to the short feedback loop. The developers also considered it rewarding to see results soon. Project B was internally split into software milestones every 1 to 6 weeks.

Interactive Planning. In project A, the project manager continuously discussed the specifications with the customer. The presence of the relevant developers in the customer meetings was considered important for gaining a common understanding and giving a feeling of appreciation for the developers. Prioritization was made together with the customer at different levels (general direction, features) also concerning the technical feasibility of the proposals.

Developers Estimate Task Efforts. In project A, the best expert in an area performed effort estimation. In project B effort estimates originated from the functional specification phase, but the development team re-estimated them before implementation. In project C, rough effort estimates were made before the project in the feasibility study, but the developers refined them later.

Visual Modeling. In project A, the technical documentation contained only a few diagrams. One reason for avoiding diagrams was the lack of a good drawing tool. In project B, developers considered a picture of the whole system showing the parts and their connections being the most important part of architectural documentation. The developers also drew UML scenario and process diagrams of their subsystems. In project C scenario and class diagrams were used.

Use Cases. In project B, use case modeling was not used, because the project was mostly technical, low-level development. In project C, requirements were documented using use cases.

Design Patterns. Project A began to use design patterns after the first major refactoring of the product. In project B, design patterns were not considered applicable due to low-level C coding.

Continuously Developed Architecture. In project A, the architecture was developed in parallel with new features. Project B made higher-level design early based on an external specification. The design remained quite stable.

Pair Programming. In project C problems in code were often solved with a pair, but during programming tasks pairing was scarce. However, even this amount of pairing when debugging spread the knowledge of the system among developers.

Collective Code Ownership. In project A, any developer was allowed to change any part of the code if needed, but in practice they focused on their own modules. In project B, developers mostly worked with their own code. Sometimes they read

others' code, but the owner made the final changes. In project C, developers were allowed to change others' code and even the platform code.

Coding Standard. In project A, a style guide was followed in naming variables, structuring the code, and organizing the code in files. In project B, instructions covered only the module interfaces. In project C, a general style guide by the company was followed.

Refactoring. In project A, architectural decay was always present as new features were added, and refactoring was exercised all the time. However, not all developers refactored consistently, causing some parts to decay quite badly. Senior people kept the code in better shape. Project B changed low-level design quite a lot during coding. Project C refactored scarcely in order not to break the existing, large code base.

Write Tests First. None of the studied projects wrote unit tests before the real code.

Automated Unit Testing. Project A tried writing unit tests without good results due to the strong GUI orientation. In project C, writing unit tests was found too difficult and time-consuming.

Customer Writes Acceptance Tests. In project A, the customer organization performed system testing. They might have had test cases defined, but the developers never saw them.

Limited Documentation. In project A, technical documentation contained a short architecture document and some technical guideline documents. These were not typically kept up-to-date. They were considered somewhat useful for new people, but apprenticeship-style hands-on training was most successful for transferring knowledge. Senior developers did not need the documents at all. Even the development at two sites did not seem to require more documents.

Project B had a short technical document of each subsystem and a general architecture description of the system. The need for design documentation was low because the size of the software was rather small and it had a modular structure. Only one person developed each subsystem and only the interfaces were of interest to others. The details and reasons behind the solutions were commented in the source code. The comments were important even for the author due to the new domain. Requirements were gathered in a short document. There was only one real user requirement and others were technical requirements.

Team in One Location. In project A a team that worked in another country was added to the project. After several months of apprenticeship in Finland, the team members started to work in their home country. The teams held frequent teleconferences, some meetings, and yearly workshops. Lead developers had own rooms and others worked in a landscaped office, which some developers did not like due to disturbing background noise. Projects B and C had adjacent two-person rooms.

Frequent Team Meetings. In project A, the team ate lunch together, and never considered it necessary to have official daily meetings. Project B had a weekly status meeting, where everyone told what he had done since the last meeting. Project C had meetings when necessary, typically 15-30 minutes once or twice a week.

Customer Always Available. In project A, there was a weekly meeting between the customer, project manager and some developers. There were also many discussions with the customer. In project B, the product manager played the customer role. His room was in another floor, but he answered questions when posed. In project C, the project manager played the role of the customer. It was difficult to identify a real customer and have proper customer involvement.

Team Develops Its Processes. In project A, the project manager created the process together with the team. The team liked the way they worked, and the lack of clear roles such as managers, designers and coders improved team spirit. In project B, the project manager created the process and discussed it with the team. The developers were most interested in developing software, so the proposed, light process did not meet with resistance. In project C, the team planned experimenting with XP practices, otherwise the process was defined outside the team.

No Continuous Overtime. In project A people worked overtime only before important releases. Project B used some overtime to speed up the first release. In project C overtime increased towards the end of the project, but was not considerable.

3 Discussion

3.1 Adoption Level

The most frequently and thoroughly adopted practices were to measure progress by working code, having developers estimate task efforts, using a coding standard, not using continuous overtime, having the team develop its own process, limited documentation, and having the team in one location.

Some practices such as interactive planning, write tests first, automated unit testing, customer writes acceptance tests, and pair programming were used almost nowhere. It may be that the familiarity with these practices was weak among the interviewees and some practices are quite exotic and difficult to adopt causing that the practices do not just emerge. It may be that the process must be created based on some documented agile methodology or a best practice list in order for these practices to be adopted. As agile testing practices are often considered as a prerequisite for many other agile practices, the low adoption of these practices suggests that more education is needed in this area.

3.2 Success Factors

We identified the following success factors in the projects. In project A, the architects stayed with the project, refactored the architecture continuously and accomplished the survival and evolution of the architecture. The first developer became the project manager, so managerial decisions were made quickly by a person who had the best technical knowledge of the software. The customer representative remained the same and a mutual trust could be built.

The personnel was one of the strengths of project B. The project manager selected developers based on their skills and traits in order to form a good team for this project. The project manager was both managerially and technically competent and designed the overall architecture.

In project C, frequent team meetings helped designing the details of the initial high-level specification. Pairing while debugging was a very useful practice and spread the knowledge of the system among the developers.

3.3 New Practices

The following additional practices were identified. *Technical authority,* e.g., a technically competent project manager improves agility because that person is able to make decisions quickly. The project manager's understanding of the technical details increases the likelihood to have the courage to decrease control elements such as managerial documents and project reviews from the process. *Team continuity*, meaning that key persons stay within the project, improves efficiency because the tacit information is preserved as happened with the architecture in project A.

3.4 Other Findings

Collective ownership improves agility by removing delays in development. If the system has a modular structure and different parts require different technical skills, the most efficient way of development may still be to let people specialize in some parts, and have one or two semi-experts as a backup for each part. Landscaped offices allow efficient and quick communication, but may disturb work requiring high concentration. Project A showed that some agile practices are viable even in a distributed project. Developers typically accepted an agile process well. This was true even in project B where they were not involved in designing the process.

3.5 Evaluation of the Research

The reported experiences of the advantages and disadvantages of specific practices were quite scarce. It may be that people have not been very conscious of using certain, admittedly vague practices that may have emerged instead of being consciously deployed. In addition, some practices, such as limited documentation and measure progress with working code actually mean not doing something, e.g., thick

documents or progress reporting. Therefore evaluating the effects of these practices and discussing about the related experiences may have been hard for the interviewees.

In the interviews, the origin of the adoption of each individual practice was not explicitly asked for. Therefore, we cannot say for sure for all practices, which were consciously adopted and which just emerged.

4 Conclusions

This paper presented experiences of the use of agile practices in three projects in the Telecom industry. The use of agile practices was more emergent than planned. Typically, processes or individual practices were not formally defined. Instead, project managers and developers used practices they considered efficient and natural.

Generally speaking both the project managers and developers were satisfied with their current software development processes compared to their earlier experiences with heavier, more formal processes. They could do what they consider important (software) and see concrete results soon through frequent customer deliveries. This might explain the positive tone of the interviewees when discussing about their projects.

The adoption level of agile testing practices, i.e., write tests first and automated unit tests, whose use is typically considered a significant prerequisite for several other agile practices, was low. More information on these practices is clearly needed among the projects.

The emergence of agile practices without conscious adoption can be considered a good sign, indicating that agile practices are considered useful by the developers themselves. However, it seems that several important practices may be neglected, if a process, whose goal is to be agile is not consciously created but instead just emerges. Agile practices presented, e.g., as process patterns could help find and deploy the most suitable practices more efficiently.

References

1. McCormack, A. et al.: Developing Products on Internet Time: The Anatomy of a Flexible Development Process. Management Science 47, 1 (2001) 133-150
2. Thomke, S. and Reinertsen, D.: Agile product development: Managing development flexibility in uncertain environments. California Management Review 41, 1 (1998) 8-30
3. ISO/IEC TR 15504: Information Technology – Software Process Assessment, Parts 1-9. Type 2 Technical Report (1998).
4. Paulk, M. et al. Capability Maturity Model for Software, Version 1.1. Technical Report CMU/SEI-93-TR24. Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute (1993).
5. Ambler, S.W.: Process Patterns. Building Large-Scale Systems Using Object Technology. Cambridge University Press, (1998)
6. McCormick, M.: Programming Extremism, Communications of the ACM 44, 6 (2001) 109-111
7. Beck, K.: Extreme Programming Explained: Embrace Change. Addison Wesley, (2000)

8. Ambler, S.W.: Agile Modeling. John Wiley & Sons, Inc., New York (2002)
9. Gamma, E. et al.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, (1995)