Jussi Auvinen | Rasmus Back | Jeanette Heidenberg | Piia Hirkman | Luka Milovanov

Improving the Engineering Process Area at Ericsson with Agile Practices. A Case Study

Turku Centre *for* Computer Science

TUCS Technical Report

No 716, October 2005

Improving the Engineering Process Area at Ericsson with Agile Practices. A Case Study

Jussi Auvinen
Oy L M Ericsson Ab, Telecom R&D,
jussi.auvinen@ericsson.com
Rasmus Back
Oy L M Ericsson Ab, Telecom R&D,
rasmus.back@ericsson.com
Jeanette Heidenberg
Oy L M Ericsson Ab, Telecom R&D,
jeanette.heidenberg@ericsson.com
Piia Hirkman
Åbo Akademi University, Institute for Advanced Management Systems Research,
piia.hirkman@abo.fi
Luka Milovanov
Åbo Akademi University, Department of Computer Science,
lmilovan@abo.fi

TUCS Technical Report

No 716, October 2005

Abstract

Besides the promise of rapid and efficient software development, agile methods are well-appreciated for boosting communication and motivation of development teams. However, they are not practical ”as such” in big organizations, especially because of their well-established, rigid processes. In this paper, we present a case study where a few agile practices were injected into the software process of a large organization in order to pilot pair programming and improve the motivation and competence build-up. The selected agile practices were the planning game, pair programming, collective code ownership and on-site customer. We show how we adjust these practices in order to integrate them into the existing software process of the company in the context of a real software project.

Keywords: Motivation, Agile Methods, Software Engineering Experimentation, Software Process Improvement, Pair Programming, Competence Build-up.

TUCS Laboratory
Data Mining Laboratory, Software Construction

1 Introduction

Agile methods hold the promise of rapid and efficient software development. Reports from industry [15, 16, 26, 29], research [4, 28, 30] and educational [12, 22, 23] settings describe positive experiences of agile practices. While agile software development responds to the challenge of change, people is often stated to be one of its main focal points [13]. Also, issues related to individual agile practices, such as knowledge building [8], have been found alluring.

However, agile approaches also have their limitations and recommended application areas, as software development methods usually do. One of these issues is that many agile methods are best suited for small and medium projects [7]. For example, Extreme Programming does not easily scale for large projects [5]: all of the developers simply cannot work together in one big room.

Regardless of project size, the interest towards agile approaches rises to a great extent from the same needs, but the actual implementation is different. It requires much more tailoring in large companies than in smaller ones [21]. The challenge lies in fitting agile methods in existing processes where software development is only a small part of the product development process. The question is whether corporations with well established and rigid processes can use just a few agile methods and still see significant benefits. Beck's discussion of the 80/20 rule would suggest not, as implementing all of the principles and practices creates synergy benefits [5], but experiences in practice have tried to prove otherwise [21, 31]. This paper presents an account of how agile methods were assessed in a case study for a large organization.

The pilot project was conducted at Ericsson, the largest supplier of mobile systems in the world. Ericsson's customers include the world's 10 largest mobile operators and some 40% of all mobile calls are made through its systems. This international telecommunications company has been active worldwide since 1876 and is currently present in more than 140 countries. The pilot project was conducted at a design department at Ericsson Finland.

This paper proceeds as follows: Section 2 presents the background and drivers for the pilot. Section 3 states the goals for the pilot and describes the means to achieve these goals. In Section 4 we describe the agile practices selected for and implemented during the pilot, while in Section 5 we present the results of the pilot in the context of the goals stated in Section 3. We discuss open issues in Section 6 and present our conclusions in Section 7.

2 Background

In early spring 2004, the design department in question arranged a workshop where different improvement areas for the software design process were identified. This workshop was a part of the continuous Software Process Improvement

(SPI) [35] activity performed at the company. One of the identified areas was the motivation of the employees, and an SPI team was assigned to come up with innovative improvement proposals for this area. The results of a survey conducted within the company indicated that job satisfaction could be enhanced by changing the way the work was assigned, arranged and carried out. The organization would benefit from increasing the employees' motivation by promoting shared responsibilities among the designers, and increasing their competence in different areas of the large software systems they are working with.

An investigation of potentially suitable methods was conducted. The SPI team found that pair programming, an agile software development practice, could promote learning and shared responsibility, and hence increase motivation. The SPI team also noted that the chosen methods would need to be easy to implement within the existing process and easy to learn. Furthermore, such methods should be flexible enough in order to be changed and adapted to the standard process of the company to keep its integrity and strict deadlines. The SPI team considered an agile approach to be well-suited for the purpose.

The SPI team presented its ideas and results to the management in the summer of 2004. Based on this proposal, the management decided to pilot pair programming together with a number of other agile practices at the design department. The pilot was to be done in a real, live project so that the experiences from the pilot would be directly applicable in the organization. Since agile practices were new to the department in question, the management team decided to cooperate with the Department of Computer Science at Åbo Akademi University where agile practices had been tried out and developed further in the Gaudí Software Factory for the last four years [4].

3 Goals and Settings

The pilot started in January 2005 with planning and start-up activities and the actual implementation was done from February to mid-April. The following sections describe the goals of the pilot, and the implementation settings, such as the team composition.

3.1 Goals

The pilot set out to investigate the possibility of using pair programming and an assortment of other agile practices in the analysis, design and early testing of the project. More specifically, the question was whether it would be possible to introduce pair programming into the standard way of working of a designer, and what changes should be made to the pair programming recommendations for them to suit the surrounding environment, including premises, IT infrastructure and project process.

The second goal, competence build-up, was set during the pilot preparation. When a new designer comes to work in a design team, he or she has at least basic skills in methods and tools, but lacks the knowledge about the specific parts of the system the team is working with. The newcomer is assigned a mentor and is assumed to be up to 70% less productive than a designer familiar with the subsystem. In this situation, the mentor's productivity is also assumed to drop by up to 20%. At the beginning of the pilot it was speculated that the mentoring of a newcomer could be effectively substituted with pair programming.

One of the most important goals of the pilot was to study the impact of pair work on the motivation of the designers. But while piloting pair programming as a means to improve the motivation of the designers and build up their competence, we wanted to keep up the efficiency of the existing way of working. Even though boosting the employee's motivation was the primary goal, the deadlines and the quality could not be sacrificed for increased motivation.

3.2 Project Scope and Selected Practices

The pilot was implemented in only two subsystems of a product called the Ericsson Media Gateway for Mobile Networks (M-MGw) [2]. The whole M-MGw application system consists of eight subsystems. The application runs on the CPP (Connectivity Packet Platform), Ericsson's 3G platform. The Media Gateway itself is a part of Ericssons' Mobile Core Network, a much larger system. The Media Gateway has an interface to several of the other nodes on the Core Network. Because of the hierarchical structure of the system, the testing of the M-MGw application is performed on many levels: from unit and subsystem tests up to call cases covering the whole network and interoperability tests with other telephony networks. The multitude of required test phases puts strict constraints on the delivery process. As a consequence, it is very difficult to have a truly agile process when developing the system. Consequently, a set of agile practices was selected for the pilot instead of trying to implement a completely agile process.

Pair programming was the first selected practice. It is supposed to be a "fun" way of doing implementation [5] and could therefore be expected to improve motivation. Furthermore, it offers an efficient way of ensuring quality in an early phase, by having an extra pair of eyes checking the code as it is being written. In order for pair programming to be easily introduced in the company and to be efficient, a few supportive agile practices were considered necessary. The practices chosen to support pair programming were collective code ownership, the planning game and the on-site customer. These practices are presented and discussed in more detail in Section 4.

3.3 The Pilot Team

The design department where the pilot was conducted is one of the departments involved in creating the M-MGw product. As seen in Figure 1, a M-MGw design project consists of three main phases that involve people working at different departments. The system department does the systemization, which includes requirements handling and overall system design. The design department handles the design of the actual product, which includes implementation specific design, coding as well as unit and subsystem level testing. The test department handles the rest of the testing.

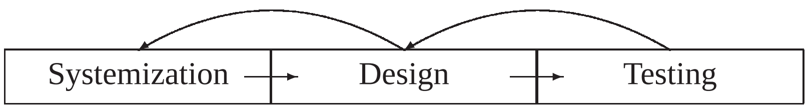

Figure 1: The main phases of a M-MGw project

There are two main roles in the design department, the designer and the subsystem tester. The designer mainly does design, implementation and unit testing, while the subsystem tester is responsible for testing on the subsystem level. The pilot was organized in such a way that only four designers involved in the actual implementation faced the pilot-induced changes in their work. In other words, the subsystem testers were only affected by the output produced by the designers. The system work was unaffected by the pilot.

Usually, each subsystem has its own team to handle all the implementation. However, the pilot team worked on two neighboring subsystems. This meant that the competence of the designers varied, allowing us to study the effect of pair programming on competence build-up. The features implemented during the pilot mostly impacted one of the subsystems, and only one of the designers had previous experience in that subsystem. Of the three other designers, two had worked with the other, less impacted subsystem. One of the designers was new to both of the subsystems. All designers were competent in the tools used as well as in the overall system principles and basic functionalities.

3.4 Room Arrangement

The room arrangements for the pilot required some attention. Since four designers were to participate in the pilot, two dedicated places for pair programming were needed. All the designers had responsibilities that required them to work alone as well, so they also needed personal work places. Furthermore, the needs for obstacle free communication and for work peace needed to be balanced.

The pilot steering team and the designers worked together to come up with the solution presented in Figure 2. In this solution, each of the designers has his own corner for private work while the pair programming work places are located in the center of the room. A divider is placed between the pair programming places to give some work peace for the pairs, without completely isolating them from each other. The designers made minor additions to the room furnishing during the pilot.

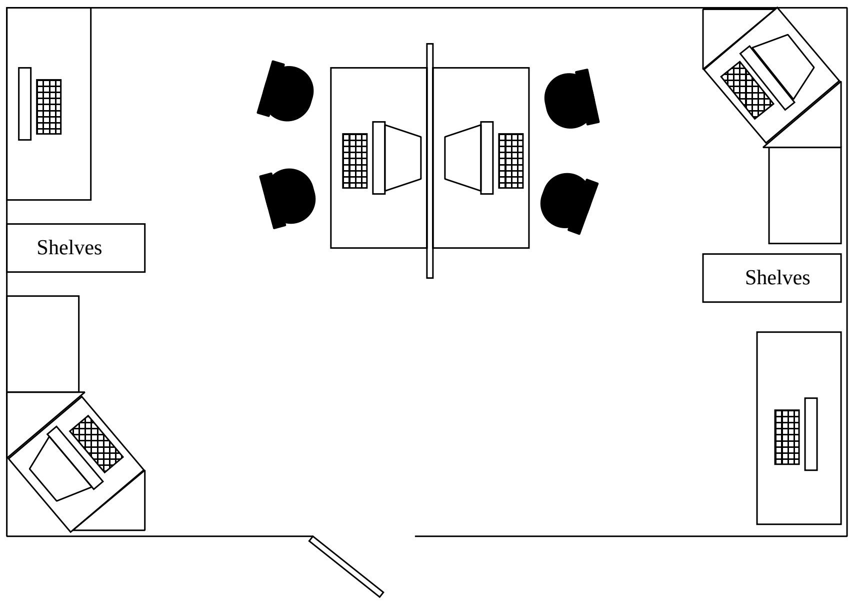

Figure 2: The pilot room

3.5 Pilot Steering

In addition to the actual pilot team, the project also had a management sponsor and a steering group. The pilot steering group was formed in order to follow the progress of the pilot and to make necessary adjustments to the pilot whenever refinement was necessary. The group included a researcher who was experienced in coaching and managing agile software projects in an academic setting and had participated in developing the process used in these projects. The task of the steering group was to monitor the adaptation of the selected agile practices into the existing process, propose changes in the practices based on this monitoring, and to collect the data needed to evaluate the achievement of the goals in the end of the pilot.

The pilot steering group and the designers held one hour long pilot steering meetings every Monday during the pilot project. The first part of each meeting was a regular team progress follow-up: what had been done, what features were currently under work and was the schedule kept. The team leader would later summarize this to the project manager. Before the pilot, this follow-up had been based on informal estimations, expressed as the designers' gut feelings, such as*"60% of the coding is done"*. This did not really provide a good sense of project progress to the managers.

During the second part of each meeting, the pilot steering group presented the results of the pilot monitoring. Typical data shown here included the distribution of the designers pair-solo work, and the accuracy of the designers' time estimation, to mention a few. The meetings proceeded with a discussion of the pilot practices both in the context of this data and the subjective opinions of the designers. These discussions aimed at finding any need for improvement and adapting the practices better into the existing process. The designers were also expected to give feedback at the end of the project. This feedback was provided in the form of questionnaires and interviews, but they also could propose ways to improve the process and the adaptation of the selected practices.

4 Agile Practices in Action

In this section, we go in detail through the agile practices selected for the pilot. We proceed with the practices one by one, first introducing the original definition of the practice and explaining the reason for selecting it. We then discuss how the practice was changed in order to be adapted to the existing process. Additionally, the sections embody some comments made by the designers during the pilot steering meetings.

4.1 Pair Programming

Pair programming is a programming technique in which two persons work together on one computer, one keyboard and one mouse [33]. One of them is called the driver and does the actual typing, while the other, the navigator, maintains a strategic viewpoint of the implementation. The roles are supposed to be switched frequently; the recommended frequency varies from an hour to a day or two.

Pair programming is broadly studied [9, 10, 18, 25, 34], and it is well appreciated for good quality of the code [9], promotion of communication and learning as well as for being a "fun" way of working. The productivity in pair programming follows Nosek's principle [27]: two programmers will implement two tasks in pair 60% slower than two programmers implementing the same tasks in parallel with solo programming.

According to the experiences with pair programming in the Gaudí Software Factory [3], pair programming should be enforced by the coach in order for it to dominate significantly over solo programming. When the choice between doing pair or solo work is left to the programmers working in the Gaudí Factory, they tend to distribute pair and solo work equally, though they all agreed that they enjoyed working in pairs more.

In this pilot, pair programming was recommended instead of enforced. While students can easily take an open-minded attitude towards pair programming, the experienced designers in the company expressed a slightly skeptical stance concerning the efficiency and benefits of the technique. As experienced designers usually do, the members of the pilot team had grown accustomed to working solo. They were concerned that adjusting to the work rhythm of the partner would be difficult.

4.2 The Planning Game

The planning game is the XP planning process [5]: the business specifies what the system needs to do, while the development specifies how much each feature costs and what budget is available per day, week or month. XP talks about two types of planning: by scope and by time. Planning by time is to choose the features to be implemented when the delivery date is fixed. Planning by scope is the opposite technique where all of the features are taken to be implemented while a release date and resources to be used are negotiated.

The planning game practice was chosen in order to facilitate pair switching. We wanted the pairs to be switched approximately weekly, but the requirements analysis documents did not as such provide suitable slots for pair switching. Moving people around when they are in the middle of working on specific parts of the system would disturb the designers, would have negative influence on productivity and quality, and would not necessarily promote competence build-up. We needed to split the features into small (one week) units supporting the desired pair dynamics. The planning game seemed to be the answer.

In the pilot, the planning game was not directly based on user stories written by the customer as in XP. The XP approach is to split the user stories to tasks. The stories describe the required functionality from a user's point of view, in about three sentences of text in the customer's terminology without using technical jargon [1, 17], whereas the tasks are written by the programmers and contain a lot of technical details [5]. The pilot planning game followed the principle of splitting requirements chunks into smaller entities. But in the pilot, the chunks corresponding to XP stories were derived from the requirements analysis document and they were called features. A task was defined as something which requires one to three days to implement, while a feature was composed of few tasks and was estimated as a week or maximum two weeks of work.

The features and tasks were identified during the planning game, which was held twice for this pilot: on the first day and in the middle of the pilot. During the first planning game, the designers selected the main pieces of the functionality from the requirements analysis document. Each of these features was then split into detailed tasks and the tasks were estimated. These estimations were in ideal programming hours and did not consider that the implementation would be done in pairs, thus no additional time for pair programming was reserved. The new estimations confirmed the original deadlines for the project. It was, however, unclear whether the pilot could follow the original deadlines since pair programming is considered to be less productive then solo programming. In a live project the deliveries have to be on schedule and if pair programming caused the schedule to slip the pilot would be cancelled.

The designers did not sign up for the tasks during the planning game. They formed pairs as needed during the development. Initially, the idea was that in order to promote competence build-up, a pair should consist of a person who is highly competent for the task, while his partner has little or no knowledge about the part of the system the task concerned. But by the time the second planning game was held, it had become clear that when defining the tasks, also the difficulty of the task should be specified and that should be used as one criterion when forming pairs and assigning tasks. Consequently, besides selecting features and splitting them to tasks, the tasks were assigned complexity (*High, Medium, Low*) and ordered during the second planning game. The division into categories was done from the most experienced designers' point of view.

After the second planning game, the distribution of tasks among pairs took both competence build-up and deadlines into consideration by using some guidelines where the level of difficulty of the task played a central role. These guidelines generally say that the simple tasks should be implemented as solo programming or in pairs where both of the designers are less familiar with the task at hand. Tasks of medium complexity should be done in pairs where the driver has less competence than the navigator. The most difficult tasks should be done in pairs where at least one of the designers has a good level of competence. These complex tasks should not be left to the end of an iteration. Additionally, both designers should be equally good when debugging in pairs.

The pilot project made no use of a special issue tracking tool, such as SourceForge or JIRA. Instead, the features and tasks were written down on the kinds of paper cards depicted in Figure 3, as originally suggested in XP.

In summary, the planning game used in the pilot was a customization of the original XP planning game. It was targeted towards the most efficient competence build-up while respecting the deadlines of the project. When implementing a task in pair, the more experienced designer taught his skills to his programming partner. This practice was highly appreciated by the designers and also by the testers. Furthermore, the designers found this type of planning game *"perhaps the most valuable practice introduced in this pilot"*. They even recommended this practice to higher management before the pilot steering group started working on the final evaluation of the pilot.

4.3 Customer on Site

The on-site customer practice was selected because it was felt that in large projects it is easy for the individual designer to lose track of the big picture and the original

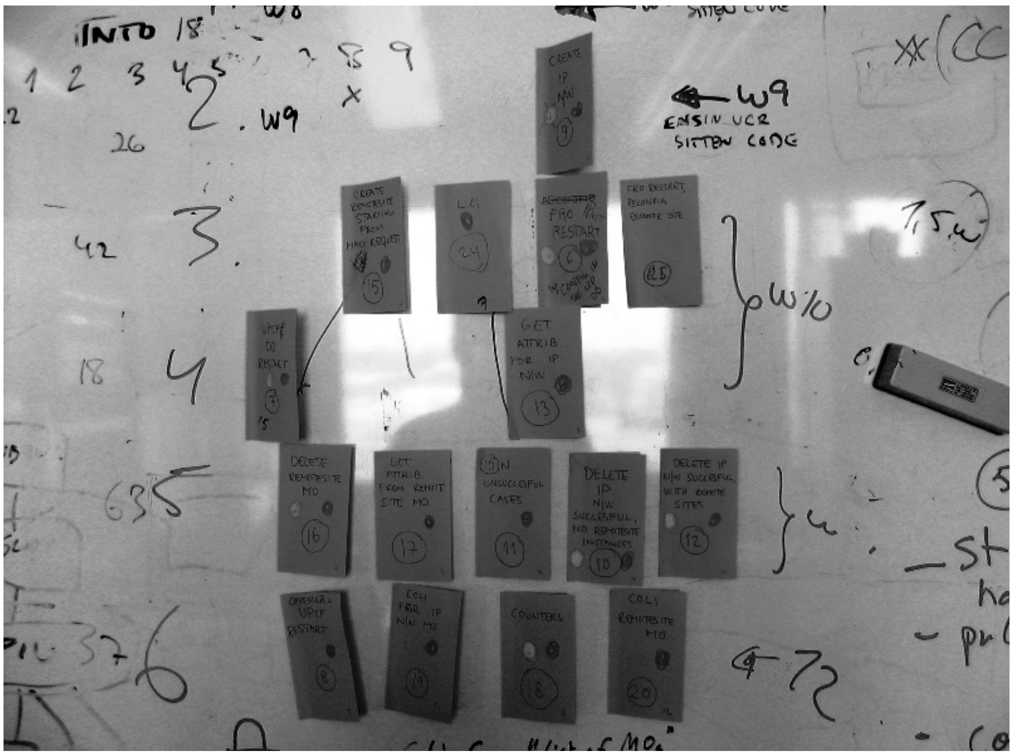

Figure 3: Features and task cards

requirements. It was thought that adding a direct channel of communication to someone in charge of the requirements would reduce the risk of misinterpretation.

The role of the customer in Extreme Programming is to write and prioritize user stories, explain them to the development team and define and run acceptance tests to verify the correct functionality of the implemented stories. One of the most distinctive features of XP is that the customer should work on site, as a member of the team, in the same room with the team and be 100% available for the team's questions.

The real on-site customer is very hard to implement in practice [11, 19, 20, 32] due to the high value or lack of commitment of the customer. Originally it was decided that the pilot will use the customer representative model as described in [14] for the customer-designers interaction, but right in the beginning of the pilot this was changed. The requirement specification for the software was complete and well-defined – this was according to the standard requirement management process used in the company. Further on, the person who prepared the latest version of the requirements was working in the same corridor as the designers involved in the pilot. Therefore it was agreed that the designers could come and ask any questions concerning the requirements personally, at any time. However, none of the designers felt the need for that during the whole pilot time – the requirements were well understood.

4.4 Collective Code Ownership

Collective code ownership in XP means that no one person owns the code and may become a bottleneck for changes. Instead, every team member is encouraged to contribute to all parts of the project. Every programmer improves any code anywhere in the system at any time if they see the opportunity [5, 6].

We chose to enforce collective code ownership in order to empower the pairs to change and update any part of the code when necessary. This concept of sharing the responsibility for the code was also necessary due to the competence build-up. By the end of the pilot, every designer should ideally have the same level of system competence. The different pairs were working with all parts of the system, designers with high competence in a particular part of the code were not the bottlenecks for changes and did not have more responsibility for these parts of the system than the rest of the design team.

Surprisingly, the design team commented this practice as *”something we already have been using, just did not know the right name for this practice”*. Nevertheless, the testers found positive changes with the introduction of shared code ownership: *”Whenever we had a question concerning some part of the code, any designer could answer us – this is something we have not seen before the pilot”*.

5 Results

In this section we present the quantitative and qualitative results of the pilot. The presented data is based on the metrics selected for the pilot and data collected during and after the pilot, minutes of the pilot steering meetings and interviews of the designers and testers. We start with the results on pair programming in Section 5.1. Actual adherence to project schedule and software quality (Section 5.2) are presented before proceeding to results concerning two softer goals stated in Section 3: competence (Section 5.3) and motivation (Section 5.4).

5.1 Pair Programming

Based on Nosek's principle [27], we expected pair programming to be less efficient than solo programming. Furthermore, other factors such as competence build-up and the overhead of introducing a new methodology led us to assume a best case scenario of a 100% increase in lead time. Nevertheless, the deliveries were made on schedule without deviations from the original man-hour estimates. This came as somewhat of a surprise, since the other teams not involved in the pilot had to work overtime to achieve their corresponding objectives on time.

The standard way of tracking time is fairly coarse, and is not suitable for the pilot. Examples of categories normally used are: implementation, testing and writing design specifications. In order to get meaningful statistics on how well the designers were adopting pair programming, a more detailed way of reporting time usage was needed. The designers were required to keep track of almost all possible work activities with the precision of half an hour. The results were collected in a database and analyzed.

The first goal of the case study was to pilot pair programming. Figure 4 shows the percentage of pair programming among the other activities of the designers. For the reader's convenience we do not distinguish between different activities such as design, programming, unit testing, etc in this chart. Instead, we generalize these tasks as *pair* or *solo* work. As seen in Figure reffig:allTasks, working in pairs took one third (33%) of all possible activities of the designers. The time when a single designer could work in pair, but had to wait for his partner due to different reasons was only 2%. This is shown as *waiting for pair* in the figure. The 9% piece of *other* activities here includes lunches, coffee breaks, department meetings and other activities not related to the project.

The largest slice in Figure 4 is *not available for pair work*. This is due to the fact that the designers had other responsibilities in other projects, in addition to their work in the pilot. This is a common situation in many organizations: a single developer is often involved in more than one project at the same time. Figure 5 shows the activities of the designers within the pilot only. Figure 5 shows that, within the pilot, more than half of the work was done in pairs: 51% pair vs. 37% solo. The pair programming practice was not enforced in the pilot, it was only

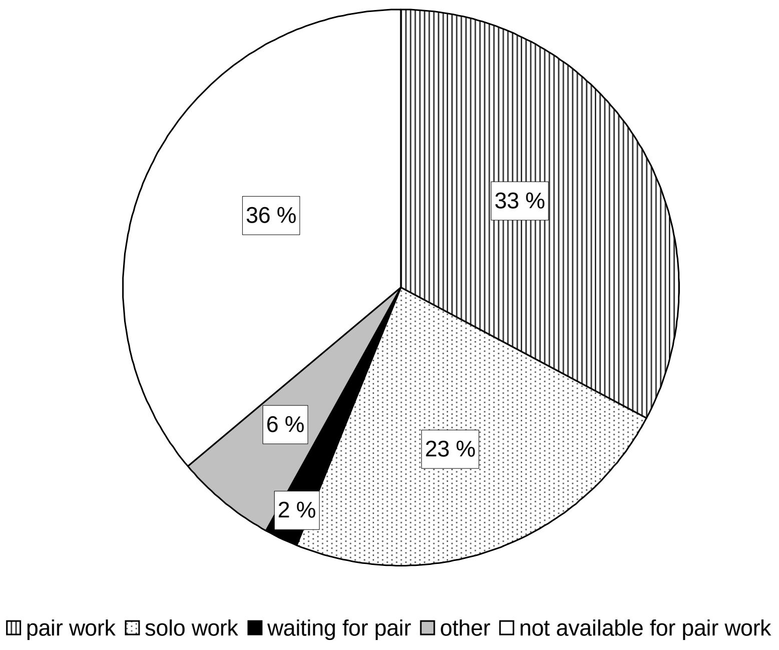

Figure 4: The distribution of the designers' activities

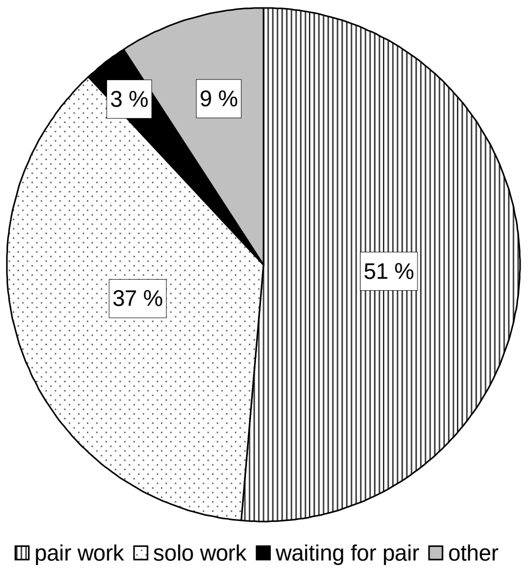

Figure 5: The distribution of the designers' activities when available for pair work

recommended. The designers decided on whether to work alone or in pair based on the estimated complexity of the task as described in Section 4.2. Based on the numbers in Figure 5 we conclude that our first goal, the piloting of the pair programming practice, was achieved.

Figure 6 gives us the share of the pair work among the tasks described in Section 4.2. There were 37 tasks defined during the planning games and implemented during the pilot. Some of the tasks required upfront design, while others were straightforward programming tasks. The numbers of the tasks (their identities) lie on the horizontal axis of the graph, while the vertical axis presents the percentage of pair work in the whole time effort to complete the tasks. Four tasks

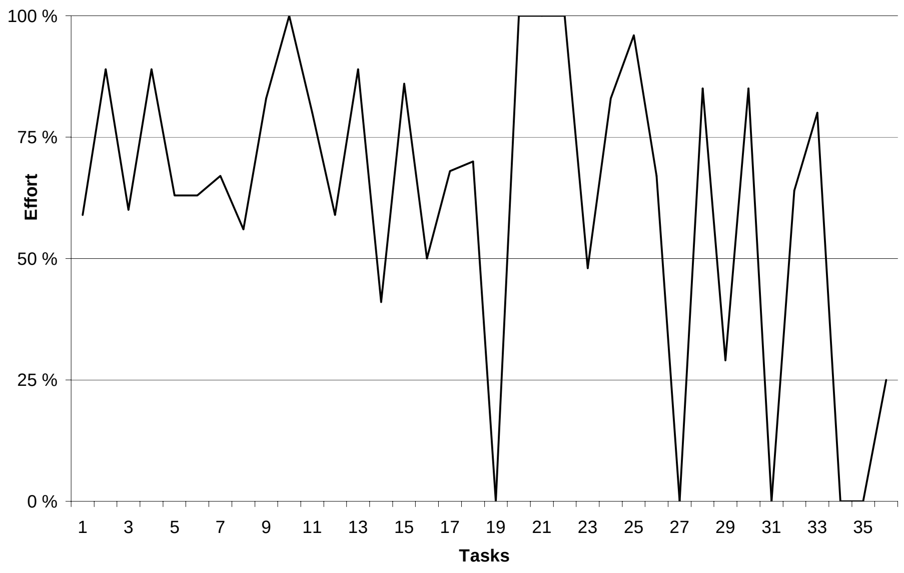

Figure 6: Percentage of pair work vs solo work per task

were implemented completely in pairs and five completely solo. For the rest of the tasks the average share of the pair work was dominating: 68% pair vs. 32% solo.

5.2 Software Quality

When the designers make the official release of the code, any faults found result in a trouble report (TR) being written. A TR describes the problem, the test which finds the fault and so forth. These TR's are the principal measure of code quality at the company.

Pair programming, as we mentioned earlier in Section 4.1, is appreciated for the higher quality code when compared to the code produced in solo work. To assess the impact of the pilot on the code quality we compared the TR count from the pilot to the TR count from a previous delivery which is similar in size and complexity. The analysis gave us a 5.5% decrease in TR's. In addition to the quantitative measure, a formal code review was conducted at the end of the pilot. The review found that the code produced during the pilot was of the same quality as code produced previously.

To assess the impact of the introduced agile practices on the subsystem testers, each of the testers was interviewed. The feedback provided by the testers was mainly positive. The first advantage of the pilot from the testers' point of view was that the planning game and the division of the system requirements into tasks. Usually, the testers start their work when the code is almost complete, but in the pilot, writing the tests could be started much earlier. This was mainly because the tasks produced by the designers in the planning game had enough information for the testers to start their work. Consequently, this evened out the testers workload.

Another positive impact of the pilot noticed by the testers was improved communication. Asking questions about the code was easier: all the designers were sitting in the same room and thanks to pair programming there were always at least two designers who were familiar with the particular piece of code a tester was asking about. As for the non-positive impact, the testers found that the number of faults found remained on the same level as before. One could argue here that this was due to the testing which was more efficient in the pilot. Another observation concerned an increase in basic (beginner's) mistakes. This could be attributed to the learning process taking place at the same time.

5.3 Competence

A very important aspect of the pilot was to measure how pair programming can improve the competence level and knowledge sharing of the designers. The designers were asked to rate their competence improvement subjectively in a questionnaire (see Section 5.5 for more details). They also answered a short quiz on both subsystems before and after the pilot. The quiz was the same on both occasions and it contained questions covering both subsystems completely. The results of the quiz before the pilot and the quiz after the pilot were compared in an effort to assess any change in competence.

The result from the first quiz was subtracted from the maximum available points. This number represented how much room for improvement the designer had. The result from the first quiz was then subtracted from the results of the second quiz. This number showed how much the score had changed after the pilot. An improvement percentage was calculated from these two numbers according to the formula shown below:

$$Improvement\% = \frac{t_2 - t_1}{max - t_1} \cdot 100\%$$

where t_1 is the score from the first quiz, t_2 the score from the second quiz and max is the maximum score of the quiz.

If a designer received 40 points out of 50 available on the first quiz, the designer's room for improvement was 10 points. If the designer then scored 45 points on the second test, the actual improvement was 5 points. Dividing 10 by 5 gives a competence improvement percentage of 50. Table 1 shows the competence improvement of each designer during the pilot. The results show a clear

	Subsystem 1	Subsystem 2	Total
Designer 1	42	31	37
Designer 2	59	67	63
Designer 3	100	33	67
Designer 4	30	7	19
Team	58	35	47

Table 1: Percentage of competence improvement

improvement in competence – the whole team gained 47%. The largest overall competence improvement was 67%, while the smallest was 19%.

We would like to note that all the members of the M-MGw project, including the pilot designers, were given a presentation covering the architecture of both subsystems before the first quiz. Consequently, the competence build-up in the pilot must have come from the actual pair work, and not the architectural presentation. If the quiz had been given before the presentation, t_1 from the improvement formula would have been smaller, hence the results for the competence improvement shown in Table 1 would be larger.

5.4 Motivation

The results concerning motivation and job satisfaction are based on semi-structured interviews conducted before and after the pilot with each designer. At the beginning of each interview, the designers also answered a questionnaire, which included statements to be rated on a 5-point scale (agree – disagree) and some open-ended questions. As the sample was small, the results of the questionnaires were not analyzed using statistical methods; they were used as an additional basis for discussion. Each interview session was recorded and took approximately an hour.

The pre-pilot interviews and questionnaires concerned general issues in job satisfaction and motivation: work content, the results of work, management, communication and social environment. Before the change in work arrangements, the questionnaires and interviews indicated that the designers liked their jobs, were well aware of the goals and expectations of their work, and had a sense of responsibility of the results. They were also committed to their work and rather content with the social environment, besides some communication aspects between teams/projects. Overall, they were rather content with their jobs. When explicitly asked about their work motivation, the designers considered their motivation to be rather good; on a school grade scale from 4 to 10, most designers gave an eight. Things that they found motivating in work included salary, challenging problems, good team, varying assignments, and learning. That is, the designers seemed to have a pragmatic view on work and job satisfaction. This down-to-earth stance reflected also on their expectations concerning the pilot. They had a somewhat positive attitude, but they did not expect any miracles. One can also note that answering motivation-related questions had become somewhat of a routine.

The nature of the post-pilot interviews was slightly less general in nature, excluding issues which were in no way affected by the pilot, such as management. On the other hand, the post-pilot questions concerned also how the designers found specific features of the pilot, such as pair programming, and how they affected their work. Concerning motivation, the results of analysis on general and pilot-specific issues differ from each other to some extent. Regarding general issues in job satisfaction, no significant changes could be seen compared to pre-pilot results. The designers did not explicitly acknowledge any change in the motivation level. The same general ingredients were still present: a pragmatic view on work, liking the job, the environment, and awareness and sense of responsibility over results. However, the pilot-specific answers reflected the actual changes in work in more detail.

The most noticeable pilot feature was *pair programming*. The designers found that it increases the sense of team work, slightly increases the sense of responsibility, smoothes out fluctuations in alertness, and facilitates learning. Pair programming also increases feedback from peers as discussions and answers immediately follow action. The most notable issue in pair programming was learning. The challenge of new things was regarded as motivating from the beginning and learning always motivating. At the same time, learning and enlarging area of experience also increases the meaningfulness of work. In addition to the positive effects, the designers also discussed the downsides of pair programming: shared pair programming schedule competes with other duties and pair work requires patience and humility. Additionally, some mentioned that pair programming was strenuous; the other side of keeping alert.

Another important pilot feature was *the planning game*. The designers liked the fact that the work was divided into smaller entities making it more systematic. But a mentioned drawback was that the pair wanted to get tasks done as fast as possible, a sense of completion frenzy. In summary, pair programming and planning game (task planning) induced a sense of learning, feedback, problem solving, responsibility, alertness, and improved structure of work.

Looking at the effects of the pilot practices from a job enrichment perspective, the detailed effects are also connected with motivation. According to Hackman and Oldham [24], five core job characteristics form the basis for job outcomes. These characteristics include skill variety, task identity, task significance, autonomy, and feedback. Comparing these with the findings in the pilot, it can be seen that all of these characteristics were affected by the changes in work during the pilot. New challenges and learning give skill variety and possibility for variation at work. Better structured work content improves task identity. Task significance is increased as the knowledge on related tasks grows. Ongoing interaction with peers increased feedback. The autonomy of work is to some extent decreased by pair programming, but the effect on felt responsibility on the other hand compensated by the increase in alertness. The core job characteristics, in turn, influence how work is experienced with regard to three aspects: meaningfulness of work (derives from skill variety, task identity, and task significance), knowledge of actual results (feedback), and responsibility for outcomes (autonomy). These three aspects, together with an individual's need for growth, lead to job outcomes, one of which is a high level of motivation and satisfaction.

6 Discussion

As we showed in the previous section, the piloted implementation process was a success. The four selected agile practices were modified to suit the larger environment and three of them were followed throughout the pilot. In spite of the new arrangements, the deliveries were made on time. However, regarding the three main goals, the outcome of the pilot was slightly different than we had expected. While the focus before the pilot was mainly on motivation and software quality, the main benefit of the pilot seems to be competence build-up. This implies that this method of working should be used specifically when there is a need for efficient competence build-up. The next two sections present possible explanations for the goal-specific results and the experiences and improvements concerning the selected agile practices. The discussion continues with directions for future work.

6.1 Goal-related issues

The fact that the designers did not explicitly acknowledge any change in motivation level after the pilot can relate to three things. First, the pilot had both positive and negative effects, which were found to balance out each other. Second, the pilot was known to last only for a certain time and many aspects of work remained the same even during the pilot. The third possible reason is related to the already stated impression of a pragmatic attitude towards work, motivation, and also measuring motivation. The pilot was possibly regarded as a refreshing interlude which broke the daily routines, but the designers were aware of the provisional nature of the pilot arrangements. In addition, a number of factors which in theory have an influence on the motivation level were not affected by the pilot. A more long-running and stable work arrangement can weigh more than temporary changes.

The question of quality, in turn, is also open for discussion. The design team was not a representative sample of a normal design team at the company, mainly due to the focus on competence build-up, which was chosen as a goal. The competence areas concerned especially the application domain and the existing base implementation of the subsystems. This fact might be the one most dominant factor to give uncertainty if the work at hand can be carried out with good enough quality and in timely fashion. Nevertheless, the outcome of the implementation was working code delivered on time.

With regard to schedule, we saw that the lead time and used man hours did not grow because of the introduction of pair programming. There is no clear reason why this is the case. This might relate to the way the process was monitored. Usually the work hours are not monitored with half-an-hour precision, and the coffee breaks and lunches are just reported to effective hours. Since it now was possible to calculate the truly effective hours, the used effective man hours might have grown, but it is impossible to tell exactly how much. Also, the increased attention to the designers' work and the more accurate monitoring of work hours was likely to increase the efficiency.

6.2 Experiences on Agile Practices

The pilot provided useful experiences considering the selected agile practices. The first improvement of the pilot was project monitoring. When originally stating the goals, transparency was not an issue. However, this was the first thing mentioned at the first steering meeting of the pilot. Previously, the data about the status of the project was vague, but during the pilot the managers and the testers could get tangible deliverables, thanks to the planning game and smaller tasks.

From the four selected practices we saw that the planning game was the most beneficial. It gave clarity to the implementation work itself, and furthermore helped in tracking the timeliness of a process. On the other hand, the customer on site was not used at all as there was no need for that. The existing organization already gives the support needed for implementation phase. Furthermore, collective code ownership was implemented so that a pair does all the needed changes for a feature (or a task) in two subsystems. Normally there would be two different teams, one making changes only to one subsystem. This new approach seemed to help in work allocation and overall function understanding, for example.

A couple of improvements were made to the piloted process already while executing the pilot. Use case realizations in the form of sequence diagrams were seen as good documents to implement in pairs. The tasks that are easy to implement should not be done in pairs. The team also noticed that some functionality is easily left out of the original tasks as the task definitions were too specific. On the other hand, it was not always easy to see what should be done for a task. Thus, the designers chose to combine some of the tasks.

6.3 Future Work

Besides the changes made during the project, other possible improvements were identified after the pilot. One suggestion for future study that was identified was task allocation and how it could be formalized to optimize different factors, such as competence build-up or lead time. The basic idea is that each task should be assigned a complexity level (*High, Medium, Low*) and an estimated completion time. Competence areas should be defined. Those areas can be based on functionality of the system or on architectural elements. Each task should belong to one competence area. Each designer should be assigned a competence level for every competence area (*High, Medium, Low*). The tasks should then be assigned based on the sum of the competence of the pair. E.g., a pair of *Low* + *Low* competence with respect to the competence area that the task belongs to, should only implement tasks of *Low* complexity, while a pair of *Low* + *High* competence should do tasks of *Medium* complexity or higher.

Using these classifications on tasks and designers, the tasks may be allocated to people in order to optimize different aspects of the development. For example, if the lead-time or more precisely the total amount of used man-hours, would be the object function, an optimization function could be of the form:

$$\sum_{i=0}^{n}[\frac{t_{ei}t_{ci}}{a}(2-\frac{c_{xi}+c_{yi}}{2})+t_{ei}]$$

where t_{ei} is the estimated amount of man-hours needed to implement the task i, t_{ci} is the complexity of a task i, $t_c \in [0..2]$, c_{xi} are competence factors for a task i for designer x, $c_{xi} \in [1..3]$, and finally a is a parameter for adjusting competence impact on implementation time

In this formula, it is assumed that the initial estimation of needed man-hours is based on having a pair, where both designers have *Medium* competence in the competence area of the task. For one task, the man hours needed for the implementation is the estimate corrected with a penalty function. If the pair's total competence is lower than average, the penalty function adds hours to the sum, otherwise it subtracts hours.

Note that there is a parameter a in the penalty function. That parameter value is not known but it can be determined from previous projects' data. It might be that the parameter is not a constant at all.

As an example, let us look at one task, t that has a time estimation of 30 hours. Its complexity is determined to be 2 (task of *High* complexity). If the pair allocated to the task have a task specific competence of 2 and 2, (i.e. both members have *Medium* competence with regard to this task), the formula reduces to

$$\frac{30*2}{a}(2-\frac{2+2}{2})+30=30$$

This means that when the pair has medium competence on a task, the estimated work hours are supposed to hold. If we choose a pair so that both members have low competence on the task, say 1, the formula reduces to

$$\frac{30*2}{a}(2-\frac{1+1}{2})+30=\frac{60}{a}+30$$

From this it can be seen that the more complex task, the more time is needed for task implementation.

We need more constraints for a complete formulation of the pair matching problem. For example, the tasks have constraints on the order they may be implemented. Also for a realistic optimization function other viewpoints should be included than used man-hours. For example, this optimization function could be a function also on parallelism, lead time and resource utilization.

This type of a task allocation can also affect job satisfaction. More time is left for learning whenever it is possible. By taking time, competence and complexity into consideration, we can decrease some of the downsides of pair programming, such as competence build-up at a time when project deadline is approaching fast. Another modification which can facilitate the use of agile practices as methods for job enrichment is having a limited, specific goal, such as job rotation, training new team members or task planning.

7 Conclusions

In this paper we presented a case study where a number of agile practices was introduced into the design department of a large company in the context of a real software project. We showed how we adjusted these practices in order to integrate them into the existing software process. The paper presented the background and goals for the pilot, the measures for the outcome of the pilot, and the actual results. The pilot concerned a small number of designers during a limited period of time.

The pilot plan originally included four agile practices, three of which were finally followed: collective code ownership, pair programming, and the planning game. While the first two provided a good experience by being helpful in overall function understanding and building competence, we found the planning game to be the most beneficial practice. The planning game with its tasks gives structure and clarity to the implementation work itself as well as increases the transparency of following the schedule. The planning game should be made an integral part of the design work as a method for work planning and progress status follow-up.

The goals of the case study were to pilot pair programming, improve the motivation and build up the competence of the designers. While two of the stated goals were reached clearly, pair programming was introduced and the increased competence was both felt and measured, the results concerning the original focus area, the motivation of employees, remained somewhat oblique and requires further study. This suggests that pair programming should be used specifically when there is a need for efficient competence build-up. The effects of learning on job satisfaction, again, can be argued for.

The value of this pilot lies ahead: the pilot gave guidelines on how to proceed with the development of the implementation process practices. Also, it seems to be beneficial to test the ideas on a wider scale, e.g. within system design or testing, and to take competence build-up and lead time into account in task allocation.

On the whole we concluded that the pilot was a success. It demonstrated that is worthwhile to use pair programming, the planning game and collective code ownership in the design and implementation. Agile methods could be refined to suite the existing settings of a large company.

Acknowledgments

We would like to thank all the participants of the pilot: managers, testers and especially the designers for their participation in the pilot and their valuable feedback. We would like to extend a special thank you to Jouni Rantanen for his involvement in the task allocation optimization function. We also would like to thank Harri Oikarinen and the M-MGw Design Department for making this study possible and Prof. Ralph-Johan Back and Dr. Ivan Porres for their help in the settings of the pilot and for fruitful discussions.

References

[1] Extreme Programming: A gentle introduction website. Online at: http://www.extremeprogramming.org/.

[2] Softswitch in Mobile Networks. Ericsson AB. 284 23-3025 UEN Rev A, April 2005. White Paper.

[3] Ralph-Johan Back, Luka Milovanov, and Ivan Porres. Software Development and Experimentation in an Academic Environment: The Gaudi Experience. Technical Report 641, TUCS, 2004.

[4] Ralph-Johan Back, Luka Milovanov, and Ivan Porres. Software Development and Experimentation in an Academic Environment: The Gaudi Experience. In *Proceedings of 6th International Conference on Product Focused Software Process Improvement – PROFES 2005*, Lecture Notes in Computer Science, Oulu, Finland, 2005. Springer.

[5] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[6] Kent Beck and Martin Fowler. *Planning Extreme Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.

[7] Barry Boehm. Get Ready for Agile Methods, with Care. *IEEE Computer*, 35(1):64–69, 2002.

[8] Gerardo Canfora, Aniello Cimitile, and Corrado Aaron Visaggio. Working in Pairs as a Means for Design Knowledge Building: An Empirical Study. In *Proceedings of the 12th International Workshop on Program Comprehension (IWPC2004)*, pages 62–69, Bari, Italy, June 2004.

[9] Alistair Cockburn and Laurie Williams. The Costs and Benefits of Pair Programming. In *Proceedings of eXtreme Programming and Flexible Processes in Software Engineering – XP2000*, Cagliari, Italy, June 2000.

[10] L. L. Constantine. *Constantine on Peopleware*. Englewood Cliffs: Prentice Hall, 1995.

[11] C. Farell, R. Narang, S. Kapitan, and H. Webber. Towards an Effective Onsite Customer Practice. In *Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering – XP2002*, Alghero, Italy, May 2002.

[12] Görel Hedin, Lars Bendix, and Boris Magnusson. Teaching Extreme Programming to Large Groups of Students. *J. Syst. Softw.*, 74(2):133–146, 2005.

[13] Jim Highsmith and Alistair Cockburn. Agile Software Development: The Business of Innovation. *IEEE Computer*, 34(9):120–122, 2001.

[14] Piia Hirkman and Luka Milovanov. Introducing a Customer Representative to High Requirement Uncertainties. A Case Study. In *Proceedings of the International Conference on Agility – ICAM 2005*, Otaniemi, Finland, July 2005.

[15] Sylvia Ilieva, Penko Ivanov, and Eliza Stefanova. Analyses of an Agile Methodology Implementation. In *Proceedings of the 30th EUROMICRO Conference*, pages 326–333. IEEE Computer Society, 2004.

[16] Andreas Jedlitschka, Dirk Hamann, Thomas Göhlert, and Astrid Schröder. Adapting PROFES for Use in an Agile Process: An Industry Experience Report. In *Proceedings of 6th International Conference on Product Focused Software Process Improvement – PROFES 2005*, Lecture Notes in Computer Science. Springer, 2005.

[17] Ron Jeffries, Ann Anderson, and Chet Hendrickson. *Extreme Programming Installed*. Addison-Wesley, 2001.

[18] David H. Johnson and James Caristi. Extreme Programming and the Software Design Course. In *Proceedings of XP Universe*, Raleigh, NC, USA, July 2001.

[19] Mikko Korkala. Extreme Programming: Introducing a Requirements Management Process for an Offsite Customer. Department of Information Processing Science research papers series A, University of Oulu, 2004.

[20] Mikko Korkala and Pekka Abrahamsson. Extreme Programming: Reassessing the Requirements Management Process for an Offsite Customer. In *Proceedings of the European Software Process Improvement Conference EUROSPI 2004*, Lecture Notes in Computer Science. Springer, 2004.

[21] Mikael Lindvall, Dirk Muthig, Aldo Dagnino, Christina Wallin, Michael Stupperich, David Kiefer, John May, and Tuomo Kähkönen. Agile Software Development in Large Organizations. *IEEE Computer*, 37(12):26–33, 2004.

[22] Grigori Melnik and Frank Maurer. Introducing Agile Methods: Three Years of Experience. In *EUROMICRO*, pages 334–341. IEEE Computer Society, 2004.

[23] Grigori Melnik and Frank Maurer. A Cross-Program Investigation of Students' Perceptions of Agile Methods. In *27th International Conference on Software Engineering*, pages 481–488, St. Louis, Missouri, USA, May 2005. ACM.

[24] Terence R. Mitchell and James R. Larson Jr. *People in Organizations: An Introduction to Organizational Behavior*. McGraw-Hill, 1987.

[25] Mathias M. Müller and Walter F. Tichy. Case Study: Extreme Programming in a University Environment. In *Proceedings of the 23rd International Conference on Software Engineering*, pages 537–544, Toronto, Ontario, Canada, May 2001. IEEE Computer Society.

[26] Orlando Murru, Roberto Deias, and Giampiero Mugheddu. Assessing XP at a European Internet Company. *IEEE Softw.*, 20(3):37–43, 2003.

[27] J.T. Nosek. The Case for Collaborative Programming. *Communications of the ACM*, 41(3):105–108, 1998.

[28] Donald J. Reifer. How Good are Agile Methods? *IEEE Software*, 19(4):16–18, 2002.

[29] Bernhard Rumpe and Astrid Schröder. Quantitative Survey on Extreme Programming Projects. In *Third International Conference on Extreme Programming and Flexible Processes in Software Engineering – XP2002*, pages 95–100, Alghero, Italy, May 2002.

[30] Outi Salo and Pekka Abrahamsson. Evaluation of Agile Software Development: The Controlled Case Study approach. In *Proceedings of the 5th International Conference on Product Focused Software Process Improvement PROFES 2004*, Lecture Notes in Computer Science. Springer, 2004.

[31] Michael K. Spayd. Evolving Agile in the Enterprise: Implementing XP on a Grand Scale. In *Agile Development Conference*, pages 60–70, Salt Lake City, UT, USA, June 2003. IEEE Computer Society.

[32] Nathan Wallace, Peter Bailey, and Neil Ashworth. Managing XP with Multiple or Remote Customers. In *Proceedings of the Third International Conference on eXtreme Programming and Agile Processes in Software Engineering – XP2002*, Alghero, Italy, May 2002.

[33] Laurie Williams and Robert Kessler. *Pair Programming Illuminated*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[34] Laurie A. Williams and Robert R. Kessler. Experimenting with Industry's Pair-Programming Model in the Computer Science Classroom. *Journal on Software Engineering Education*, 10(4), December 2000.

[35] Sami Zahran. *Software Process Improvement: Practical Guidelines for Business Success*. Addison-Wesley, 1998.

ISBN 952-12-1616-6
ISSN 1239-1891