

An Iterative Improvement Process for Agile Software Development

Outi Salo^{*,†} and Pekka Abrahamsson

VTT Technical Research Centre of Finland, P.O. Box 1100, FIN-90571
 Oulu, Finland



Research Section

Agile software development of short iterative cycles offers an opportunity for rapid, visible and motivating software process improvement (SPI). The agile principles suggest the regular reflections of agile project teams for improving the efficiency and adaptation of the process. However, current literature provides little support or empirical evidence for conducting such improvement efficiently, systematically and in a validated manner. Thus, this article proposes an Iterative Improvement Process for conducting SPI within individual agile project teams, which aims at increasing the ability of software developers to improve the development process based on their experiences and context knowledge. The approach has been trialed within a multiple case study of five consecutive case projects where both qualitative and quantitative research data has been systematically collected. The empirical data confirms the positive effects of the Iterative Improvement Process on the software development projects and reveals the willingness of software developers to participate in the Iterative Improvement activities due to the rapid and visible changes in their working practices. Copyright © 2006 John Wiley & Sons, Ltd.

KEY WORDS: software process improvement; SPI; agile software development

1. INTRODUCTION

Software process improvement (SPI) aims at providing the means for improving the capabilities of software development teams and organizations. SPI methods have been intensively studied and developed since the early 1980's. The existing SPI methods designed for traditional software development approaches, such as the waterfall model, usually adopt a top-down approach for improving the software development process. In, for example, the Goal/Question/Metric method (GQM) (Basili

1994) and in assessment reference models, such as CMMI (Capability Maturity Model[®] Integration) (SEI 2002), the organizational business goals lay the foundation for all SPI initiatives. Furthermore, the underlying SPI approaches, such as Quality Improvement Paradigm (QIP) (Basili 1989) and IDEAL (McFeeley 1996) support the traditional idea of evolving universal and repeatable software development processes and address organizational process control (Boehm and Turner 2003, Lycett *et al.* 2003).

Lately, the traditional methodologies have been challenged by agile software development approaches such as Extreme Programming (XP) (Beck 1999). These agile approaches seek to provide adaptable processes to support context-specific development, increased customer satisfaction, lower defect

* Correspondence to: Outi Salo, VTT Technical Research Centre of Finland, P.O. Box 1100, FIN-90571 Oulu, Finland
 †E-mail: Outi.Salo@vtt.fi



rates, faster development, and responsiveness to rapidly changing requirements (Boehm and Turner 2003).

Evidently, the underlying differences of the traditional and agile software development approaches require new SPI mechanisms to fit the context of agile software development. The short development cycles of agile software development provide continuous and rapid loops to iteratively learn, to enhance the process and to pilot the improvements. The underlying principles of agile processes (Agile Alliance 2001) propose that 'at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly'. In other words, the agile methodologies seek to move the process control from management to developers (Lycett *et al.* 2003) while suggesting that 'each situation calls for a different methodology' (Cockburn 2002, p. 184), a dilemma which has been widely acknowledged far prior to the agile movement (e.g. Malouin and Landry 1983). However, there is still neither enough guidance nor empirical evidence available on how to execute these improvements rapidly and effectively in the agile-specific manner while still taking into consideration the aspects found fundamental in traditional SPI, i.e. strong methodological basis and an empirical validation. Yet, XP, for example, calls for assessing the effects of the changes if the project team is adapting its working practices (Beck 1999).

This article proposes an Iterative Improvement Process for conducting SPI within agile software development teams. It focuses on utilizing the context knowledge and learning of the software developers on improving their daily working practices. The process is founded on the principles of agile software development (Agile Alliance 2001), and also on postmortem review (Dingsøyr and Hanssen 2002) and reflection workshop (Cockburn 2002) techniques. The method is trialed within five consecutive agile software development projects where both quantitative and qualitative research data is collected on the project teams applying iterative improvement mechanisms in order to improve the organizational base process (i.e. adapted version of XP). The results indicate that agile teams are both capable and willing to effectively improve their development processes with concrete and visible results. Furthermore, it is also revealed that the importance of management involvement in

the activities of the Iterative Improvement Process should not be overlooked.

The article is organized as follows. Section 2 presents the background for SPI in the agile development context. An Iterative Improvement Process for team focused improvement of agile software development is presented in Section 3 and the research design is defined in Section 4. In section 5, the quantitative results from the case projects are laid out while the perceptions of software developers are discussed in Section 6. The discussion and conclusions of the multiple case study is presented in Section 7.

2. BACKGROUND

In this section, the underlying differences of traditional and agile software development are defined from the viewpoint of conducting SPI, and the existing methods for agile process adaptation are discussed.

2.1. Underlying Differences of Traditional and Agile Software Development and SPI

There are certain fundamental differences between traditional and agile software development (Table 1) that advocate the need to define new SPI mechanisms for agile software development context.

The traditional approaches of SPI, such as the QIP (Basili 1989) and the IDEAL model (McFeeley 1996), enhance continuous improvement of software development processes in the software engineering context. Usually, the traditional SPI approaches support the ideology of creating and improving a universal and repeatable software development process for an organization (Lycett *et al.* 2003). In addition, SPI initiatives are, traditionally, largely controlled by the organizational level¹ and management (Lycett *et al.* 2003). In contrast, the principles of agile software development request that 'at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly' (Agile Alliance

¹In this article, organizational level SPI refers to the SPI stakeholders (such as Software Engineering Process Group (SEPG) (SEI 2002) and activities that aim for the organizational improvement of the software development process



Table 1. Underlying differences of traditional and agile software development and SPI

| | Traditional software development and SPI | Agile software development and SPI |
|--|---|--|
| Software development process | Universal approach and repeatable solution to provide predictability and high assurance | Flexible approach adapted with collective understanding of contextual needs to provide faster development times, responsiveness to rapid changes, increased customer satisfaction, and lower defect rates. |
| Process control | Control on organizational level | Self-organizing teams |
| Primary means of knowledge transfer | Document based knowledge transfer | Face-to-face communication |
| Immediate focus of process improvement | Improvement of organizational software development processes/(future projects) | Improvement of daily working practices of ongoing project |

2001). Thus, the ideologies of agile software development emphasize the need for process adaptation within ongoing projects and seek to move process control from the organizational level to practitioners (Lycett *et al.* 2003).

Traditionally, the mechanisms of organizational SPI to learn from projects has been largely based on external knowledge capture (Lycett *et al.* 2003), i.e. by collecting and analyzing, for example, metrics data and relevant documentation from pilot projects according to specified organizational SPI goals. Then, the organizational learning can be disseminated back to the project level as validated improvements in the software process. In the principles of agile software development, however, the emphasis is on face-to-face communication as a primary means of conveying information to and within development teams.

In the context of traditional and agile software development, it has been realized that the core process knowledge for creating organizational learning takes place at the group level (Nonaka and Hiro-taka 1995, Vandeville 2000). Traditionally, project postmortems have been regarded as ‘an excellent step into continuous knowledge management and

improvement activities’ (Birk *et al.* 2002). They have served the purpose of harvesting experiences of success and failure of previous projects and have been claimed to be a valuable tool for organizational learning (Stålhane *et al.* 2001) and improving methods and practices (Collier *et al.* 1996) for future projects in an organization. Thus, the post-mortem techniques (e.g. Collier *et al.* 1996, Kerth 2001) adopt a traditional approach to SPI and aim at delivering project experiences for the purpose of organizational level analysis and for improving future projects. Consequently, the implications of such project postmortems need to be validated in pilot projects at a later stage and, if found useful, disseminated through various organizational practices in time. However, according to the fundamentals of agile software development, the primary focus of process adaptation within agile project teams is on the immediate use of the experiences of developers in improving the ongoing project.

2.2. Existing Methods for Agile Process Adaptation

Complying with the agile principles, many agile methodologies such as Scrum (Schwaber 1995) and Crystal Clear (Cockburn 2005) propose conducting iterative improvement within the teams. For example, Crystal (Cockburn 2002), p. 184 contains a reflection workshop technique and Scrum the activity of Sprint Retrospective (Schwaber 2004). While many of the agile methodologies suggest an activity of iterative process adaptation, they often seem to lack specific and clear procedures for conducting such activities. Consequently, case studies have been reporting lack of focus and long duration of such retrospectives (e.g. Mann and Maurer 2005).

Methodology-independent techniques have also been suggested for agile process adaptation, such as the postmortem review technique by Dingsøyr and Hanssen (2002). The ultimate goal of the postmortem reviews, as well as that of the reflection workshops of Crystal Clear, is to transfer the experiences of project teams into immediate and concrete software process improvements. They suggest how to conduct a workshop for collecting the experiences of past iterations from project teams and for generating software process improvements for the following iterations. While these techniques will most likely accomplish their goals, they appear to fall short in some central SPI aspects. Firstly, they



do not consider nor provide means for validating the implemented software process improvements in the ongoing project. In addition, the means for actual implementation and systematic follow-up of improvement activities seem to be missing. However, follow-up procedures are considered important also in the agile context; XP, for instance, requires that the project teams should assess the effects of the implemented changes (Beck 1999).

3. AN ITERATIVE IMPROVEMENT PROCESS

In this section, a process for conducting SPI within agile software development projects is presented. It is founded on the fundamentals of agile software development, where the development teams are empowered and encouraged to adapt and improve their daily working practices iteratively and in a face-to-face manner (Table 1). The steps of the Iterative Improvement Process are conducted rapidly and iteratively, spinning several times during the project. Its immediate goal is to provide means for project teams to improve the daily working practices of the ongoing project effectively and systematically. On the other hand, it emphasizes conducting SPI in a validated manner, i.e. providing project teams with mechanisms to assess the effects of the changes in the process, as suggested by Beck (1999).

The Iterative Improvement Process (Figure 1) consists of six steps: (a) preparation, (b) experience collection, (c) planning of improvement actions, (d) piloting, (e) follow-up and validation, and

(f) storing. Figure 1 illustrates the timing and sequence of the steps within the iterations of the agile software development process. The activities requiring the software developer's participation and effort are illustrated in grey while the white color indicates the activities that involve other relevant SPI stakeholders (e.g. facilitator).

In the Post-Iteration Workshop (hereafter referred to as PIWs), after every completed iteration, the project team gathers together to discuss the problems and obstacles of the previous iteration (step 2.) and seeks solutions for improving their daily working practices (step 3). Apart from the steps needed to conduct PIWs, the project team also implements the agreed SPI activities during the next software development iteration (step 4). The PIW session is led by a facilitator who is also in charge of the preparatory activities (step 1) and also performs the step of storing the results (step 6). Thus, the facilitator is responsible for the activities that do not, as such, provide immediate added value for the project team itself. On the other hand, the facilitator may provide the project team with valuable SPI expertise on the adopted SPI techniques as well as on the existing organizational practices and guidelines to support the project team in effectively and systematically turning its experiences and knowledge to concrete improvements. The facilitator should, preferably, be someone outside of the project team with adequate technical competence and neutral perspective (Kerth 2001). In addition, a scribe is needed for maintaining the important outputs during the execution of PIWs. This responsibility should be taken by one

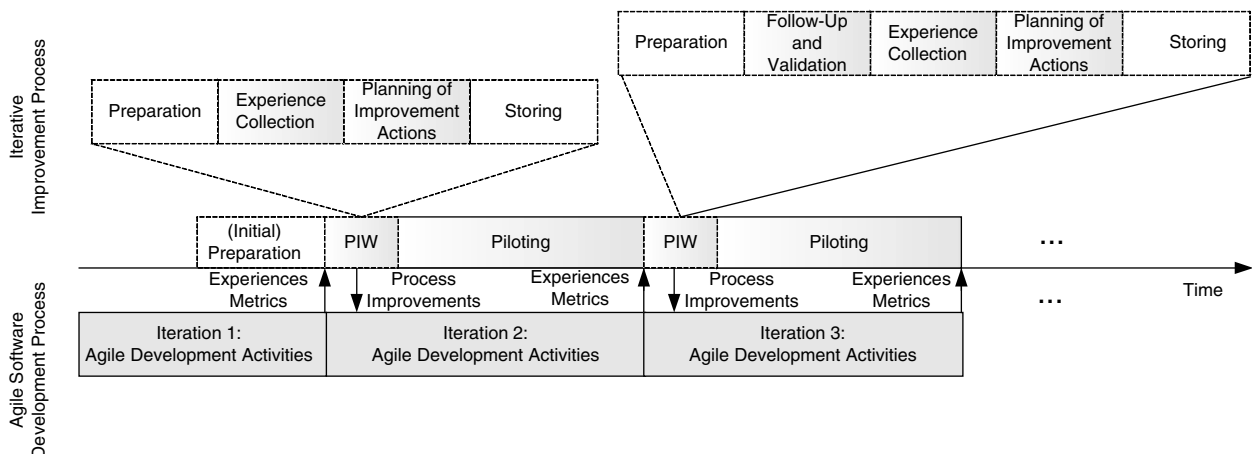


Figure 1. Steps and information flows of iterative improvement within agile software development process



of the project team members. In the following subsections, the six steps of the Iterative Improvement Process are defined in more detail.

3.1. Step 1. Preparation

Firstly, preparation for conducting Iterative Improvement is needed prior to or early in the project, involving, for example, planning the frequency and schedule of the PIWs. The short development cycles of agile software development provide a suitable rhythm for conducting Iterative Improvement. The preparation should also include careful planning of the various techniques that are to be used in the different steps of PIWs depending on factors such as the team composition and size (Kerth 2001). During the project, the preparation task includes setting up equipment for the PIW sessions and preparing workshop material, such as the analysis and visualizations of the metrics data to be used in the 'follow-up and validation' step. This step may also involve altering the selected techniques during the project.

3.2. Step 2. Experience Collection

The experience collection step is the first activity of the actual PIW session that takes place from the first completed iteration on. The central idea of the PIW is to base the process improvements on the obstacles and problems that are identified by the software development team. Few techniques

have been suggested for deriving experiences from the project team and generating them into process improvements in the agile software development context. For example, Dingsøyr and Hanssen (2002) have suggested the so-called *KJ method* (Scupin 1997) for collecting positive and negative experiences on a flap sheet and structuring them separately into labelled groups for further analysis (Figure 2).

The groups generated from the negative findings provide a means of visualizing different problem areas and can, later on, be used as a tool for finding solutions to the problems.

3.3. Step 3. Planning of Improvement Actions

The improvement actions for the next iteration are planned in a PIW session after the obstacles and problems have been identified. Thus, the starting point for planning the improvement actions is the set of negative experiences of the project team from the previous iteration, which are grouped into problem areas. The context knowledge of the software developers is then used to formulate useful and functional process enhancements for the next iteration.

Different techniques, such as Root Cause Analysis (Bergman *et al.* 2002) or the focus group technique (Kerlinger and Lee 2002) can be used for finding the causes and solutions for problems that have emerged. The focus group technique is based on a free and open discussion led by a facilitator. Its advantages are, according to Kerlinger and



Figure 2. Grouping of findings



Lee (2002), low cost and fast generation of ideas along with flexibility and effectiveness. The facilitator – regardless of the selected problem solving technique – plays an important role in guiding the software development team to effectively use their knowledge in formulating feasible and collectively agreed solutions to be tried out in the next iteration. When selecting a technique, it should be considered whether the goal is to focus on a few central problems or to cover the problems that have emerged more widely. However, it should be noted that as new knowledge and team learning accumulate during the problem solving session, new problems may also emerge.

In order to pilot the agreed process improvements during the software development activities of the incipient iteration, the improvements need to be explicitly defined, regarding, e.g. what the problem is exactly, what the concrete actions are that need to be taken to improve the situation, who is responsible for carrying out the improvement actions and when. Preferably, the improvement actions should be small enough to be implemented in the incipient iteration. In addition, the software development team should collectively agree on the planned improvement actions and the facilitator should confirm that these are within the possible organizational limitations for process tailoring.

Furthermore, each improvement action should include a plan of how its implementation is followed-up and validated after piloting. The enhancements can often be validated using merely qualitative data, while relying on the experience and knowledge of the project team. However, quantitative validation may also be used. When considering quantitative validation, various issues should be taken into consideration, such as the availability of metrics data, and the effort needed for collecting additional metrics weighed against the added value it would provide. Different techniques, such as GQM (Basili 1994), may be used for defining the metrics needed for quantitative validation. The responsibilities and schedules for validation tasks (e.g. data collection and analysis) should be defined at this point as well.

Apart from the process improvement actions that are conducted within a software development project, it is most likely that the development project will also require process improvement actions for which external support is needed. Related requests

should be addressed to the relevant external stakeholders. The facilitator's important role as a change agent between the two organizational layers should be noted.

3.4. Step 4. Piloting of Process Enhancements

The piloting step is conducted in the incipient iteration of software development according to the plans agreed upon in the previous PIW. In addition to implementing the defined plans for process improvement, the measurement data for validation purposes is collected and the necessary data analysis is prepared as feedback for the next PIW. In the Iterative Improvement Process, the planned improvement actions should be put into action as soon as possible, preferably in the incipient iteration. This makes the software developers' effort put on the SPI activities visible and increases the motivation and satisfaction of the software developers regarding both their daily work and the improvement itself (Salo 2004). However, without appropriate organizational engagement to support the adaptation of agile project teams, the externalized requests of improvement will fail.

3.5. Step 5. Follow-Up and Validation

The follow-up and validation of piloted process improvements are carried out at the beginning of the next PIW session. This needs to be done to determine if the previously defined improvement actions have taken place as planned or if they need to be postponed to the next iteration. The effectiveness of each of the piloted process improvements should also be systematically validated. For example, the project team should be able to monitor the development and assess if the process is, in fact, being improved and then take further action accordingly. The validation should be conducted using the experiences of the project team (qualitative validation), and analyzed metrics (quantitative validation) as planned for each improvement action. The qualitative validation involves a brief discussion and resolution of the team members on whether the adopted improvement is considered worthwhile or if the original way of doing things should be readopted. Although it may take longer for the project team to conduct qualitative validation, i.e. to interpret the metrics



data, this may provide in-depth and meaningful realizations among the project team and result in valuable future improvements. To make this activity as effective as possible, the facilitator's role is vital.

3.6. Step 6. Storing of SPI Results

The storing of PIW results should be done systematically. In this context, storing simply denotes filling of an electronic or document template to provide project teams with a tool for recalling the agreed improvement actions during the next iteration and for conducting their follow-up and validation as agreed in the following PIW. The storing step may also provide feedback for the organizational level SPI stakeholders and other project teams if so required. This activity should not take longer than half an hour after the workshop nor exceed two pages in length. The storing of data and knowledge from a PIW session should be conducted immediately after the PIW session, when the important details are still fresh in the facilitator's mind. The validation data for the planned improvement actions should also be stored after its interpretation in the following PIW.

4. RESEARCH DESIGN

In this section, the context of this research is described, including the setting of the research, and the research methods applied during the research process, along with the target of this research.

4.1. Research Setting

All the five case projects were conducted as controlled case studies (Salo and Abrahamsson 2004).

This research approach has been designed to suit the study of agile methodologies in particular, and is aimed at providing valuable results for both scientific and practical software engineering communities. The adopted research approach enables software development in close-to-industry settings where business pressure is present as well. Therefore, it contains some of the features typical of laboratory experiments, such as a high degree of control over independent variables, execution and measurement, and environmental conditions.

The controlled case study approach aims for a dual outcome: (a) fully functional software system or a software product for a customer, and (b) research data on selected aspects. Thus, one central difference to laboratory experiments is that the project team, in fact, produces a real software product for a real customer organization, which participates in the project in the role of a customer and also provides software developers for the project team. In this study, all the case projects were conducted in the same open office settings at VTT, the Technical Research Centre of Finland, with different customers and different project teams. Some of the central characteristics of the case projects are presented in Table 2.

The project teams adopted all the central XP practices (Beck 2000) except for that of the on-site customer, which was available only for the first project. The project teams were free to iteratively adapt the software development practices based on their experiences within certain organizational limitations. The aim was to facilitate the daily software development work and to adapt the base process, i.e. XP (Beck 2000), iteratively to better suit the given mobile software development context. The project level process enhancements were to be made solely according to the Iterative Improvement Process and

Table 2. Central characteristics of the five agile case projects

Table with 6 columns: Characteristic, Case 1, Case 2, Case 3, Case 4, Case 5. Rows include Size of project, End product, Duration, Iteration length, Team size, and Schedule.

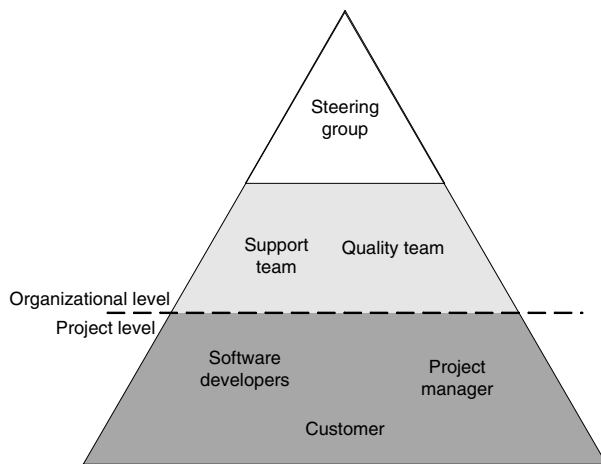


Figure 3. Organization structure of case projects

within certain organizational limitations and guidelines provided by the facilitator of the workshop. The improved process was then, to as great an extent as possible, transferred to the next project.

The organization of the case study context (Figure 3) included a steering group of members from both the customer and the research organizations. The task of the steering group was to monitor the progress of the software development and to make high-level decisions on issues such as large acquisitions or overall scheduling of the project. The support and quality teams consisted of researchers of the organization involved in the software development projects. Both teams were working in close collaboration with the software development teams. The role of the support team was to train and coach the project members on the software development process and its practices, while the quality team would provide SPI related support needed by the project team, e.g. regarding the implementation of the agreed SPI activities. In addition, the quality team systematically utilized the learning of the agile development team in defining an agile software development process named *Mobile-D*[™] (Abrahamsson *et al.* 2004, Ihme and Abrahamsson 2005). The customer, though working off-site, was a fixed part of the software development teams, participating iteratively in activities such as planning and testing.

4.2. Research Goals

The goal of this research was to study how SPI could be conducted systematically and effectively

within agile software development projects, taking into consideration the fundamentals of agile software development (Table 1). The effectiveness of the SPI activities vis-à-vis the effort required from the software development itself, and the acceptability of SPI activities for the software developers were considered important. In addition, the mechanisms of following up and validating software process improvements were considered as vital SPI elements needed for a systematic improvement effort within agile project teams.

Elements from two existing agile learning techniques, namely the reflection workshop technique (Cockburn 2002) and postmortem reviews (Dingsøy and Hanssen 2002), were selected as a starting point for the Iterative Improvement Process, which was incrementally evolved throughout the case studies as presented in Table 3. During the multiple case study, the mechanisms for systemizing the planning, follow-up and validation of the process improvements and for storing the resulting process knowledge were established (e.g. action point template for structured storing of the agreed SPI actions (Table 3)).

4.3. Research Methods and Data Collection

This study can be characterized as constructive research, in which a total of five consecutive case studies formed the basis for building and evaluating (March and Smith 1995) the Iterative Improvement Process. An action research approach (e.g. Cunningham (1997)) was applied, the researcher playing the role of a facilitator. This enabled an effective way to 'integrate theory with practice through an iterative process of problem diagnosis, action intervention, and reflective learning' (Lau 1999).

The empirical research data was collected from a total of 19 PIWs. Neither this number nor the analysis presented in this article includes the last workshops of the projects involved as these were conducted as traditional project postmortem sessions and, thus, were not comparable with the actual PIWs. In addition, the postmortems solely served the purposes of organizational SPI, which is out of the direct scope of this article.

Both quantitative and qualitative research data was collected from the five case projects. The project level data of Iterative Improvement was collected on (a) the effort used on PIWs, and (b) the quantity of positive and negative experiences and (c) their



Table 3. Evolution of the iterative improvement mechanisms in the case projects

| Step of iterative improvement | Case 1 | Case 2 | Case 3 | Case 4 | Case 5 |
|----------------------------------|--|---|--|---|---|
| Experience collection | <ul style="list-style-type: none"> • KJ method | <ul style="list-style-type: none"> • KJ method | <ul style="list-style-type: none"> • KJ method | <ul style="list-style-type: none"> • KJ method • Bi-weekly stand-up PIW | <ul style="list-style-type: none"> • KJ method • Bi-weekly stand-up PIW |
| Planning the improvement actions | <ul style="list-style-type: none"> • Focus group technique • Free form action point list | <ul style="list-style-type: none"> • Focus group technique • Action point list for systematic follow-up | <ul style="list-style-type: none"> • Focus group technique • Action point template | <ul style="list-style-type: none"> • Focus group technique • Action point template | <ul style="list-style-type: none"> • Focus group technique • Action point template |
| Piloting | – | – | <ul style="list-style-type: none"> • Action point template | <ul style="list-style-type: none"> • Action point template | <ul style="list-style-type: none"> • Action point template |
| Follow-up and validation | – | – | <ul style="list-style-type: none"> • Qualitative validation | <ul style="list-style-type: none"> • Qualitative validation • Quantitative validation | <ul style="list-style-type: none"> • Qualitative validation • Quantitative validation |
| Storing | | | <ul style="list-style-type: none"> • Action point template | <ul style="list-style-type: none"> • Action point template | <ul style="list-style-type: none"> • Action point template |

content, (d) the quantity of the implemented process enhancements at project level and (e) their content. The data also includes (f) the quality of the validation of each SPI initiative in the different projects, and (g) the results of the validation (i.e. if the enhancements were actually found to improve the process). Finally, the project teams participated in a semi-structured focus group interview (Morgan 1984) at the end of each project. Each interview covered a range of aspects of agile software development also concerning the Iterative Improvement activities of project teams. The interview aimed to reveal the attitudes and perceptions of developers with regard to their participation in the Iterative Improvement activities. The resulting transcriptions of the taped interviews were analyzed in a hermeneutic mode (Klein and Myers 1999) by interpreting the textual data in order to either confirm or diminish the findings and suggestions made on the basis of the quantitative research data.

5. RESULTS OF THE MULTIPLE CASE STUDY

In this section, the steps of the Iterative Improvement Process are described as conducted in the case studies. The lessons learned from each phase are presented along with the empirical data.

5.1. Preparation

The activities of Iterative Improvement in the case studies were conducted in teams of less than ten developers. One central guideline in their planning was to make them as effective as possible concerning both effort and results. Thus, the SPI mechanisms were tailored throughout the case studies (Table 3), as shortages and improvement opportunities were detected regarding the used methods and techniques.

The PIW sessions took place after the first four completed iterations in every project (only three in Case 3) (Table 2) as the first activity of development iteration. At that point, the experiences of the previous development cycle were still fresh in the minds of the developers.

Figure 4 illustrates the effort spent on PIWs (duration). On average, one PIW session lasted for <1.5 h (84 min) ranging from an average of 1 h in Case 3 (62 min) to an average of nearly 2 h in Case 2 (109 min). The data excludes the effort spent on preparation, piloting and storing. In all the projects, the duration of PIWs seems to decline toward the end of the project, except for the second PIWs in the three middle case projects (i.e. Case 2, Case 3 and Case 4).

The average percentage of developers' effort spent in participating in PIWs was 1.9% (all case projects included). The percentage per project varied between 1.4% (in Case 4) and 2.4% (in Case 2). This figure does not include the effort of the final

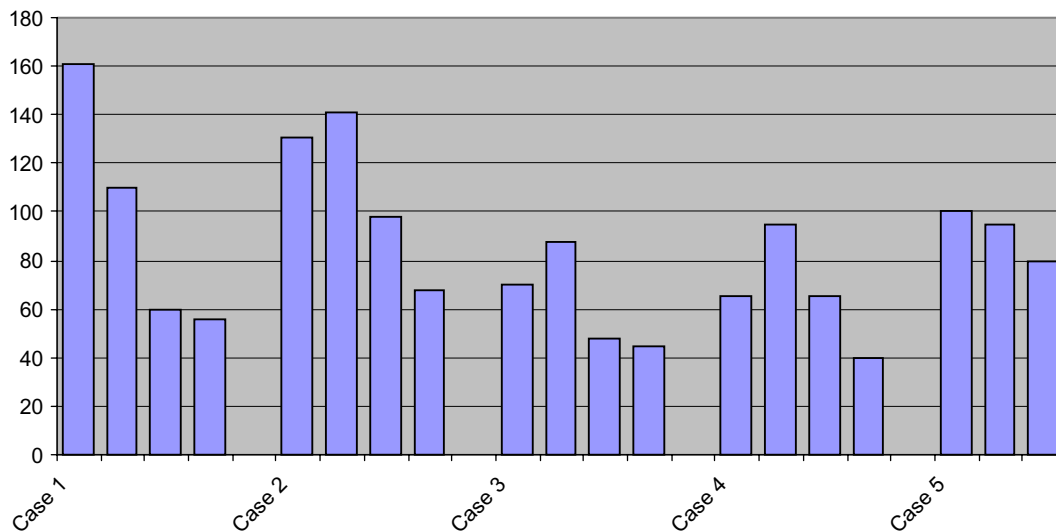


Figure 4. Duration (in minutes) of the PIW sessions in case projects

project postmortem sessions nor the developer's effort spent on implementing the improvements or the facilitator's effort in planning the PIWs and packaging their results.

5.2. Experience Collection

The KJ method (Scupin 1997) was found to provide an effective and systematic brainstorming technique for collecting experiences from the relatively small teams of the agile software development case projects, totalling under ten developers as suggested by Beck (Beck 2000). This method was also used by Dingsøy and Hanssen for applying the post-mortem review technique (Dingsøy and Hanssen 2002). One reason for selecting the KJ method over a freer discussion to collect experiences, as suggested by (Cockburn 2002, p. 184), was found in its controllability and effectiveness as a result of strict procedures. The KJ method proved to effectively support the collection and structure the experiences of the developers in the workshops for generating process enhancements and for organizational purposes.

The Case 4 team also adopted a bi-weekly stand-up PIW as a process improvement action generated in a PIW session. The team felt that it was hard to recall all the important ups and downs of the previous iterations and, thus, decided to put a PIW flap sheet on the wall of their colocated workspace, on which they could record the experiences during

the project. In the middle of an iteration, the project team would also gather for a stand-up PIW, where the team would generate their experiences on the wall. These post-it notes were, then, included in the experiences discussed in the following PIW.

In the actual PIW session, project teams took time to generate, first, the positive and then the negative experiences recorded on post-it notes and placed them individually for display on a flip chart with clarifying comments. Concurrently, the facilitator grouped first the positive and then the negative findings on separate flap-sheets and labelled the groups with explicit names. The groups of negative findings formed on the flap sheet are hereafter referred to as 'problem areas'.

The value of collecting the positive experiences was found in lifting up the team spirit, as the top group of positive findings (15.8% of all the positive experiences of all projects), which was related directly to team spirit and cooperation issues. Interestingly, planning and estimation were rated among the top two positive experiences even though they were also at the top of the problem areas (Table 4). The third most frequently cited positive issue throughout the projects was the pair-programming practice.

The negative experiences, for their part, played a central role in PIWs regarding the actual generation of process enhancements. Altogether, the findings of the projects were grouped into a total of 28 different



Table 4. Quantity of experiences in three central problem areas

| Problem area | Project | Rank of problem area | Negative experiences | % from all negative experiences | Positive experiences | % from all positive experiences |
|-------------------------|---------|----------------------|----------------------|---------------------------------|----------------------|---------------------------------|
| Planning and estimation | Case 1 | 1/13 | 13 | 22.4 | 8 | 8.7 |
| | Case 2 | 1/19 | 13 | 13.8 | 10 | 9.9 |
| | Case 3 | 5/13 | 5 | 10.9 | 6 | 14.0 |
| | Case 4 | 5/17 | 4 | 8.0 | 3 | 5.7 |
| | Case 5 | 5/12 | 4 | 11.8 | 0 | 0 |
| Metrics collection | Case 1 | 2/13 | 9 | 15.5 | 2 | 2.2 |
| | Case 2 | 2/19 | 12 | 12.8 | 9 | 8.9 |
| | Case 3 | 1/13 | 6 | 13.0 | 2 | 4.7 |
| | Case 4 | 3/17 | 5 | 10.0 | 2 | 3.8 |
| | Case 5 | 12/12 | 1 | 2.9 | 0 | 0 |
| Unit testing | Case 1 | 3/13 | 7 | 12.1 | 6 | 6.5 |
| | Case 2 | 4/19 | 10 | 10.6 | 2 | 2.0 |
| | Case 3 | 5/13 | 5 | 10.9 | 0 | 0 |
| | Case 4 | 17/17 | 1 | 2.0 | 0 | 0 |
| | Case 5 | 12/12 | 1 | 2.9 | 3 | 5.7 |

problem areas. In one session, however, typically only 2 to 13 problem areas were addressed.

Both Dingsøy and Hanssen (2002) as well as Cockburn (2002) suggest that the negative experiences of the project team should be prioritized and only the most important ones should be analyzed. In none of the PIWs, however, were the number of findings limited or the problems prioritized. Instead, all the findings were considered to be equally important, on the one hand, to ensure that no important issues were left undisclosed, and on the other hand, to avoid a situation where someone could feel that their opinions were regarded as less important. However, this was possible because of the relatively small team size

limiting the number of experiences in each PIW session to a manageable level (Figure 5).

The positive and negative findings illustrated in Figure 5 include all the individual post-it notes that were placed on the flap sheet by the team members. This means the same experience might, at most, occur as many times as there were project team participants in the workshop. In total, the Case 1 team produced 92 positive and 58 negative findings, whereas the corresponding numbers were 102 and 94 in Case 2, 42 and 46 in Case 3, 53 and 50 in Case 4, and 53 and 34 in Case 5.

Both the positive and negative experiences along with the actions needed for improvement seemed to follow a declining trend within all projects

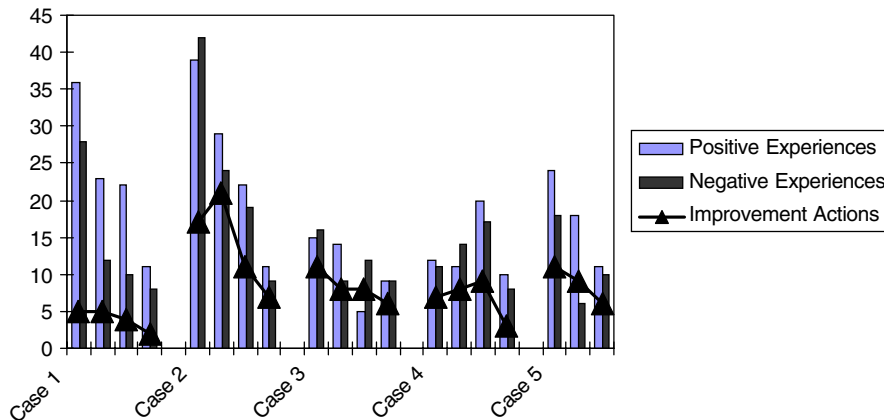


Figure 5. Quantity of positive and negative experiences, and improvement actions



(Figure 5). An exception to this was the Case 4 project, in which the project team faced a serious personnel conflict culminating at the end of the 2nd iteration. The problems can be clearly seen in the Case 4 PIW data, showing an increasing number of negative findings, until the solution of the problematic situation and also in the following PIW when there was still uncertainty among the team on, for example, the staffing of the new team. Accordingly, the solved conflict can be seen to increase the number of positive findings within the project team as well.

Four distinct problem areas were determined through the Iterative Improvement processes, concerning planning and estimation, metrics collection, technical environment and unit testing. In the following, these areas were examined closely to evaluate the effects of the Iterative Improvement Process. However, the technical environment issue will not be further addressed in this article as its problems were largely dependent on the devices and tools available on the market rather than on the learning of the project team or the state of the process itself.

Table 4 illustrates the number of negative and positive experiences generated for the top three problem areas of the projects and their ranking in all the problem areas within a project. For example, planning and estimation was the most addressed problem area of Case 1 (1/13) out of the 13 problem areas addressed during the project (1/13) (Table 4).

Planning and estimation was one of the top five problem areas in all the cases. The majority of negative experiences concerned the effort estimation. Inaccurate definition and planning of tasks during the planning game (Beck 2000) was also among the top reported problem areas. However, planning and estimation was also ranked second among the positive findings, while the planning game was found very useful in defining the content of the next iteration, and effective in designing the tasks. The positive experiences also included improved estimation accuracy and scheduling of iterations owing to the process improvements.

Metrics collection was carried out extensively in the case projects, for both project monitoring and research purposes, and was found to take a lot of time and effort. However, it should be noted that the Iterative Improvement activities did not require any extra metrics collection as the validation of the SPI actions was done by utilizing the existing metrics. Especially in the first two projects, the project

teams found the data collection too time consuming and laborious. The developers used spreadsheets to collect the effort data, along with filling up research diaries. Owing to the high ranking of this problem area, the decision was made to use a database tool for the data collection of the last three projects. Within all the case projects, the complaints concerning metrics collection declined toward the end (Figure 6). Consequently, the project team's immediate positive response to the organizational improvements in metrics collection mechanisms can be seen in the increasing numbers of positive findings (Figure 6). For example, when improvements were made to the metrics collection practices (i.e. separate pair-programming data sheet and defect list) for the 2nd iteration of Case 2, the next PIW yielded a positive comment from every team member (Table 4) on even such an annoying issue as metrics collection.

The *unit testing* problems were mostly related to the test-driven development (TDD) (Beck 2003). The qualitative data from the PIWs reveals that in

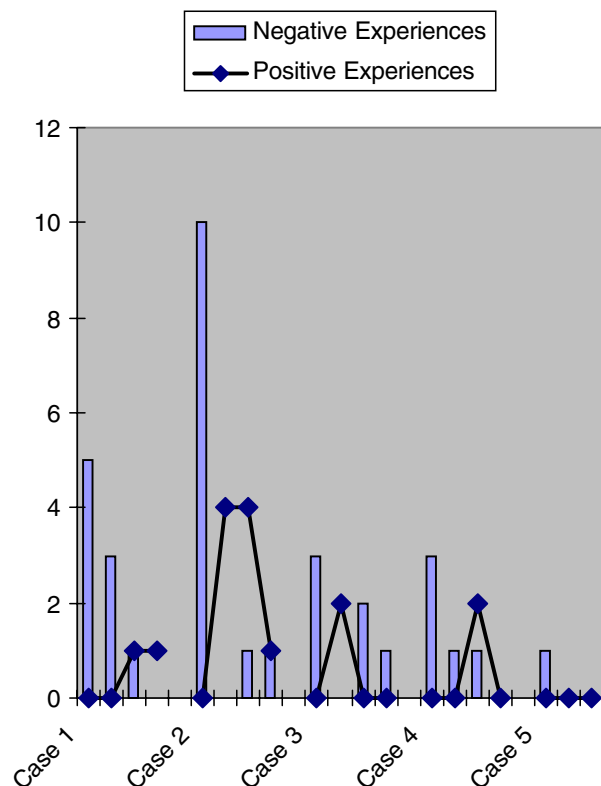


Figure 6. Improvement of 'metrics collection' in case projects



the first two projects there was simply not enough training and expertise available for applying TDD. In both projects, the client side testing of the mobile environment, especially the user interface testing, was found impossible when employing the TDD approach with the available tools. Despite the difficulties, the Case 1 team generated positive findings on TDD as a highly promising testing approach. In the later projects, in which adequate expertise and tools were already available, TDD was no longer an issue (Table 4).

5.3. Planning of Improvement Actions

The step of planning improvement actions focused on defining the underlying causes of problems (i.e. negative experiences) and their solutions, and on determining the SPI actions for the next iteration. The focus group technique (Kerlinger and Lee 2002) was selected for planning the improvement actions in the case projects. As the size of the focus group is recommended to be large enough for diverse viewpoints and small enough to be manageable, a session enabling unsolicited involvement of each participant (Kerlinger and Lee 2002) was found highly appropriate for the context of this research (Table 2). It was also found to be useful for gaining an in-depth view of a fairly large amount of findings (Figure 5) effectively, regarding both the effort (Figure 4) and the resulting improvement actions (Table 6). Yet again, the importance of the facilitator was highlighted to ensure the effectiveness of planning the improvement actions and confirming that each improvement action was concrete, specific and small enough to be implemented in the incipient iteration. In addition, of importance was the fact that during the case studies each action regarding the schedule and the responsibilities of the team members was specified.

A central tool for the facilitator in guiding the planning of improvement actions was the flap sheet where the negative findings were grouped in labelled problem areas (Figure 2) as suggested by the KJ method. This enabled the facilitator to focus the group discussion on one problem area at a time while making sure that the improvement actions were agreed and carefully planned before moving on to the next topic. The mechanisms for planning the improvement actions needed improvement throughout the case studies (Table 3). In the 1st case project, the process improvements

agreed upon by the team were recorded on a flap sheet in a free format. No clear responsibilities were set, and the implementation of the SPI initiatives were not followed-up in any way. In addition, the listed improvement actions were often too vague to act on. For example, action points such as 'Review code for comments' and 'Careful task planning' are hardly specific enough to actually improve the situation. These fuzzy action points and nonstructured planning of SPI activities did not make it possible to trace back if the SPI action had taken place or if it had been effective. In addition, no mechanism was provided acquiring external SPI support.

As a solution, a document template (Table 5) (see also Salo 2005) was generated prior to the launching of the Case 3 project to support the planning, follow-up and validation of the improvement actions. The template provided guidance for the facilitator and the project team on the issues to be considered for each agreed improvement action. The example rows in the action point template (Table 5) present two simple action points from consecutive PIW sessions, where a problem was being gradually solved.

In the template, the 'Problem Area' field links the improvement actions with the corresponding problem areas formed through the grouping of the negative experiences. The 'Problem' field specifies the problems that emerged when discussing the negative experiences within a specific problem area and withholds a summary of the underlying problem. The aim was to provide explicit knowledge of the issue so that it could be understood by the project team and, if necessary, also by external SPI stakeholders. The 'Action point' field should clearly define a specific action to be taken in the next iteration to solve the problem, whereas 'Actor' defines the responsibilities for the task. 'Validation plan' describes how and when the follow-up and validation for the action point is to be conducted. The 'Validation' field is reserved for updating the results of the validation after the piloting (i.e. in the next PIW) or, for example, links to the metrics data sheets.

The number of agreed improvement actions seems to follow a similar declining trend in both positive and negative experiences (Figure 5). The content of the improvement actions (Table 6) reveals that 44% of them can be categorized as actual SPI actions, such as changes into software development practices and tools. A total of 56%, however, can



Table 5. Action Point Template

| Problem area: Planning and Estimation | | | | |
|---|--|-----------------|---|--|
| Problem: | Action point | Actor | Validation plan | Validation |
| Task estimations too inaccurate in release 1. => release delayed. | Analyze and visualize task data (effort used-effort estimated per each task) for next PIW. | Tracker | The team together interprets the visualized effort estimation accuracy data task by task in the next PIW to evaluate causes for estimation inaccuracies. | Effort estimation data revealed too extensive and indistinct tasks => in planning, the team was not able to comprehend the content of tasks to make proper estimates but did guesswork. NEW AP =>. |
| Too extensive and indistinct tasks. | Tasks to be split enough to understand their content clearly. In the next planning, the maximum task size is 4 hours effort. Analyze effort estimation accuracy data for next PIW | Project manager | The team together interprets the visualized effort estimation accuracy data task by task in the next PIW to evaluate if estimation accuracy has improved. | Smaller task size seemed to improve the effort estimation accuracy. Splitting tasks into size of a maximum of 4 hours (when possible) works and is continued. |
| | | Tracker | | |

Table 6. Improvement actions from case projects

| Project | SPI actions with org. support | SPI actions without org. support | Other actions with org. support | Other actions without org. support | TOTAL actions per project |
|---------|-------------------------------|----------------------------------|---------------------------------|------------------------------------|---------------------------|
| Case 1 | 6 | 3 | 1 | 6 | 16 |
| Case 2 | 16 | 9 | 5 | 26 | 56 |
| Case 3 | 5 | 6 | 9 | 13 | 33 |
| Case 4 | 8 | 2 | 4 | 13 | 27 |
| Case 5 | 12 | 3 | 6 | 5 | 26 |
| TOTAL | 47 | 23 | 25 | 63 | 158 |

be categorized rather as ‘other actions’ such as acquiring equipment and fulfilling other daily needs of developers.

The top 3 topic areas of action points in all the case projects were metrics collection, planning and estimation, and issues having to do with physical environment (Figure 7). As it can be seen, the top two topics of action points are identical to the two top problem areas of the case projects (Table 4). Table 7 reveals the number of the action points performed iteratively on the top three problem areas.

Regarding *planning and estimation*, the central problems had to do with the lack of detailed planning at the beginning of the iteration, which would directly affect another problem issue, namely effort estimation. From the first project on, the project teams were suggesting that the tasks of the previous release should be analyzed in order to

improve the effort estimation accuracy. On the basis of the requests of the project teams, a metrics called ‘velocity %’ (adapted from Project Velocity metric (Beck 2000)) was systematically adopted beginning with the Case 3 project. The project velocity of the previous iteration was calculated ((team effort used on implementation tasks/total team effort) × 100) to provide a basis for estimating the amount of time to be allocated for actual software implementation (i.e. tasks) in the following iteration. All project teams from the Case 3 (2nd PIW) project onwards also took up the practice of interpreting the gaps between the estimated and realized effort for each task. This would reveal the underlying causes for inaccurate estimations (i.e. too large tasks not defined or understood well enough at the time of effort estimation). Figure 8 illustrates how the Case 3 team learned to decrease its average task size and, consequently, was also able to significantly



Table 7. Agreed project level improvement actions in three problem areas

| Problem Area | Project | Iteration 1 | Iteration 2 | Iteration 3 | Iteration 4 | TOTAL |
|-------------------------|---------|-------------|-------------|-------------|-------------|-------|
| Planning and estimation | Case 1 | 0 | 0 | 2 | 1 | 3 |
| | Case 2 | 2 | 2 | 0 | 0 | 4 |
| | Case 3 | 0 | 2 | 1 | 0 | 3 |
| | Case 4 | 1 | 1 | 0 | 0 | 2 |
| | Case 5 | 1 | 1 | 0 | - | 2 |
| | TOTAL | | | | | |
| Metrics collection | Case 1 | 1 | 1 | 1 | 0 | 3 |
| | Case 2 | 3 | 3 | 0 | 0 | 6 |
| | Case 3 | 3 | 0 | 2 | 2 | 7 |
| | Case 4 | 1 | 2 | 1 | 0 | 4 |
| | Case 5 | 3 | 3 | 0 | - | 6 |
| | TOTAL | | | | | |
| Unit testing | Case 1 | 0 | 1 | 0 | 0 | 1 |
| | Case 2 | 1 | 2 | 0 | 0 | 3 |
| | Case 3 | 2 | 1 | 1 | 0 | 4 |
| | Case 4 | 0 | 0 | 0 | 0 | 0 |
| | Case 5 | 0 | 0 | 0 | - | 0 |
| | TOTAL | | | | | |

improve its average estimation accuracy toward the end of the project.

The central SPI actions in *metrics collection* were the introduction of new data collection sheets for the Case 2 project and the decision on adopting a database tool (TaskMaster) for data collection in the last three projects. These improvements, however, cannot be seen in the quantity of improvement

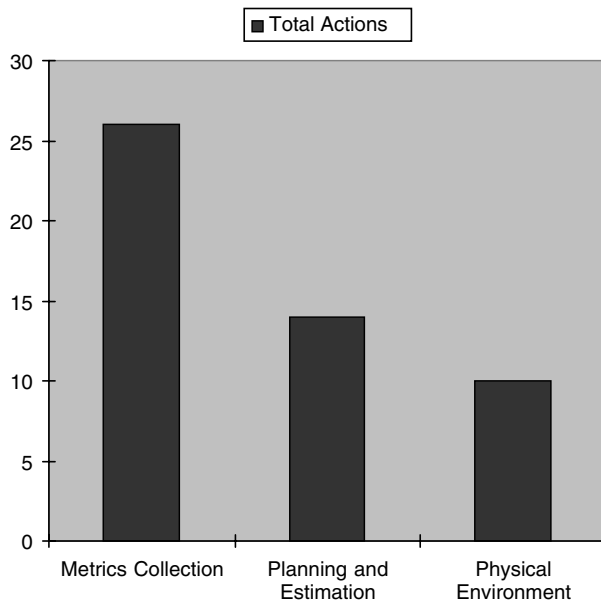


Figure 7. Top 3 action point topics in case projects

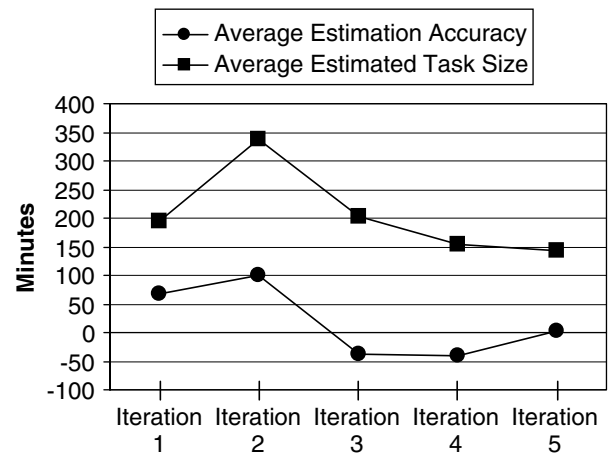


Figure 8. Evolution of average estimated task size and estimation accuracy in the case 3 project (in minutes)

actions conducted by the project teams as they were externalized and conducted by the quality team. In general, the decisions and actions regarding the improvement of the metrics collection mechanisms required organizational decision making and action taking even if they were directly affected by the improvement requests of project team generated in PIW's. The project level action points generated in PIW's (Table 7) mainly concerned improving the data collection itself to ensure more reliable and consistent data. The adoption of a new yet still slightly incomplete database tool for metrics



collection in the Case 3 project also gave rise to some improvement suggestions. However, it can be seen that even though the last case project (i.e. Case 5) still suggested some modifications to the TaskMaster tool, the metrics collection was considered the least important among the problems of the project team (Table 4).

The ranking of the 'unit testing' problem area was finally improved in the middle of the Case 3 project when the team even suggested dropping TDD from their software development practices. As this was not possible due to the organizational restrictions, the management needed to take concrete actions. To begin with, Case 3, along with the last two project teams, were provided with a coach participating in TDD activities whenever needed. Furthermore, the management decided to acquire a new tool for TDD activities and to invest in improving the TDD training of the incipient project teams prior to the project. The corresponding action points of the Case 3 project team were to allocate more time to planning test cases and on coaching. Consequently, TDD was no longer an issue in the last two projects.

5.4. Piloting

The action points were planned usually to be implemented in the following software development iteration. The improvement actions in the category of 'other actions' (e.g. acquiring a new license for a software development tool), however, did not usually require actual piloting but, rather, execution. The research data reveals that as much as 67% of actual SPI actions required external support and participation for their complete implementation, whereas the corresponding rate for the 'other actions' category was only 28% (Table 6). Fortunately, a quality team was constantly available for the project team, participating actively in the process improvement when needed. The activities of the quality team, however, also needed constant adjustment to fit the process improvement activities of agile software development (Salo and Abrahamsson 2005).

5.5. Follow-Up and Validation

From the Case 2 project on, the action points were systematically monitored and managed. The follow-up and validation step was a valuable component

in conducting the Iterative Improvement activities. The action point template adopted in the Case 3 project proved to make the follow-up more systematic, while also ensuring that the validation was not forgotten. Firstly, every planned 'actual SPI action' needed a confirmation if it had been implemented (follow-up) and if the project team regarded the change as something to be permanently adopted in the process (qualitative validation). Even though the SPI actions generated by the project team usually proved to be very useful, there were also occasions when the old way of doing things was found better or further modifications were found necessary. The 'other actions' also required assurance of their execution. In fact, after the Case 2 team agreed to adopt a follow-up of action points in the 1st PIW, it was found that as much as 35% of the action points of the 1st PIW had been left undone and would very likely have been forgotten if not postponed to the next action point list.

The quantitative validation of actual SPI actions was adopted in the last two projects. In all, 12% of the actual SPI actions were quantitatively validated in the Case 4 and Case 5 projects. Quantitative validation proved to play an important role in, for example, ensuring the effectiveness of SPI actions made to increase the estimation accuracy and enhancing pair-switching in pair-programming activities.

5.6. Storing of SPI Results

A structured document template (i.e. an Action Point Template) (Salo 2005) was used in the three last case projects for systematically storing the process knowledge after each PIW session and follow-up of action points (see more in Section 5.3). The document was filled out by the scribe of the PIW session, i.e. usually the person in the project team who was responsible for the metrics, and revised and stored by the facilitator after the session. It was then placed on the wall of the open office space as a reminder for the project team.

6. DEVELOPER'S PERCEPTIONS

The development teams participated in semi-structured focus group interviews after each project.



The analysis of the taped and transcribed interviews was conducted by evaluating the textual data in order to reveal if the developer's perceptions supported or, alternatively, diminished some of the implications made based on the quantitative research data. In particular, the focus was on acquiring the developer's attitudes and perceptions on the usefulness and suitable frequency of the adopted Iterative Improvement activities, as well as on the role of the external facilitator.

6.1. Usefulness of the PIWs

The data collected from the interviews revealed the positive attitude of developers toward the Iterative Improvement activities that were conducted throughout the projects. The software developers claimed to be willing to participate in PIW sessions and regarded them as an effective tool for actually effecting and improving the daily software development practices:

'PIW's were an efficient and honest way to improve the process because they actually forced the process to take a better direction' (Case 1 member)

'If something was poorly and complained about, it was fixed.' (Case 2 member)

'We got our voices heard. Not necessarily would we have said those things otherwise but just bitten on the bullet till the end.' (Case 2 member)

'Well it was useful as the drawbacks were said out loud and the setting of the responsibilities ensured that the things got fixed.' (Case 3 member)

'It was just necessary. I have noticed during this project that how else could it happen.' (Case 4 member)

'Even if the same things had been said on some other occasion, they would not necessarily have been registered. Now they were.' (Case 4 member)

'In the meetings, it is good to go through what we have done in those iterations. It allows everyone to voice their opinion, if there is something that someone wants to complain about or say that it was good, it is the best place to tell it, I think.' (Case 5 member)

As the extracts reveal, the developers generally agreed on the positive influences that the Iterative Improvement had on their daily work. One central factor for the developers' positive attitude toward the Iterative Improvement seems to be the possibility to actually influence their daily working practices in an organized manner and the immediate implementation and visibility of the improvements.

6.2. Frequency of Conducting PIWs Within a Project

The software developers were also asked about an appropriate frequency for holding PIWs:

'Two weeks is a long time to think back. It was good to conduct the stand-up PIW's weekly and actual workshops bi-weekly. It was also useful to collect experiences on a flap-sheet during the iterations, when remembered.' (Case 4 member)

'I think it is good to have those, e.g. once or twice per project, but not more.' (Case 5 member)

The above quotations reveal that the frequency of holding PIWs is a highly context-specific issue. For one, as software developers' opinions about how well adapted and mature a process is, are bound to be highly subjective, the same goes for their estimations regarding how great the need for improvement is during a project. Furthermore, the need for holding PIWs is dependent on their original purpose, i.e. whether the focus is solely on SPI or if the PIW's also serve the purpose of regular project meetings, thus also serving other SPI related needs and addressing the worries of the developers.

6.3. Role of the Facilitator

The comments of software developers supported the assumption that an external facilitator should be used. For one, it was seen important that the whole team could concentrate on the brainstorming together.

'It is better that the facilitator is someone outside the team so that the whole team can then open up. One of the team could not participate if this was not the case.' (Case 3 member)

The facilitator's objective role was also seen helpful in disclosing problematic issues:

'If the facilitator was a team member and only the team would participate, the comments would likely change because it would feel as if you were protesting to your team mate. Not necessarily would you like to complain to another team member but when an outsider comes along things can be said more freely.' (Case 4 member)

Furthermore, the importance of the external and objective facilitator was also seen in guiding the open discussion of the project team and helping them focus on the relevant issues at hand:



'If it has to be someone from the team, it should be the project manager. However, I think it is much better to be outside of the team, because when someone from outside of the project comes, you have a better chance to find the relevant things. Otherwise, if the workshop had been run by the team, the same questions and the problems that were discussed day by day would have been there.' (Case 5 member)

The qualitative data also indicate that a facilitator may have an effect on the results of the workshop. Thus, more importantly, the objectivity and impartial standpoint of the facilitator is essential.

7. CONCLUSIONS

Both the qualitative and quantitative data of this research indicate that the agile teams were both capable and willing to improve their development processes by participating in the activities of the Iterative Improvement Process. The presented examples of process improvements reveal that the teams were able to make small and simple, yet effective and visible improvements during the projects. Furthermore, the quantitative project improvement data indicates an increased satisfaction of project teams and, thus, also an improvement of the daily working practices within the projects. The qualitative data revealed the perceptions of developers to be very favorable for this kind of rapid and visible team centred SPI.

The research data also indicate that mechanisms are needed to conduct the SPI activities within agile project teams in a systematic manner. For example, it was revealed that without mechanisms of adequate definition, documentation and follow-up of SPI actions, as much as 35% of the agreed improvement actions remained undone. It was also revealed in the study that the importance of external support in the improvement activities of the project teams cannot be overlooked. For example, a large proportion (i.e. 33%) of the actual SPI activities of the project teams would not have been implemented without external support. Without such organizational support, it is highly likely that the motivation of software developers toward participating in Iterative Improvement would have decreased. It should be noted that the success of Iterative Improvement Process may also rely largely on the skills, enthusiasm and activeness of the

facilitator working as a change agent within the project team and between the organizational layers.

On average, the developers spent 1.9% of the total effort in participating in Iterative Improvement activities. This data excludes the developer's effort spent on implementing the process improvements and collecting metrics data. The average duration of a PIW session was less than 1.5 h, yet there seems to be an overall decline in the duration of the workshops within individual case projects with the exception of the second PIW in the three middle case projects. The duration of the last workshops varied only between 40 and 80 min. This indicates a decreased need for improving the process toward the end of the project and, consequently, decreased SPI effort requested from the developers. This data also correlates with the decline of negative experiences and SPI actions implemented during the project. Together, this may indicate that the need for holding PIWs decreases toward the end of the project as a certain level of adaptation is reached. On the other hand, as much as 56% of the improvement actions did not directly relate to SPI. Thus, PIW sessions can also serve as ordinary project meetings, at which the state of the project and the everyday needs and concerns of the project team can be addressed and acted upon. Thus, the purpose of PIWs should be considered when planning their frequency.

The positive experiences were repeatedly found to reflect the improvements made in the previous iteration. No evidence, however, was found to imply that the satisfaction of the project teams with their software development practices would decrease due to the declining trend of positive findings. On the contrary, the highly positive perceptions of the software developers and the fact that the negative experiences primarily served as a tool for improving the daily working practices of the software developers both support the assumption that the decline of the negative experiences was, in fact, a consequence of improved software development processes and daily working practices.

It should be noted that although the Iterative Improvement Process has been empirically trialed and evaluated in the agile software development context in this study, it may also be feasible in other approaches of software development where the base process contains clear intermediate points at which PIWs can be iteratively placed, where the organization appreciates the process knowledge



of its workers, and empowers and encourages the software developers to improve their daily working practices. It should also be highlighted, that total arbitrariness and isolation of project teams in adapting their software development practices may be risky. Rather this study indicates that there is a need to address the integration of Iterative Improvement within project teams and SPI activities at the organizational level. For one, the organizations currently operating in an environment where both traditional and agile approaches have been adopted may need to strike a balance among the two, such as cooperative control of SPI within management and developers. The Iterative Improvement within agile project teams is likely to provide valuable process knowledge for continuous organizational SPI purposes. It should be noted that the developers also require organizational support in conducting their Iterative Improvement activities.

ACKNOWLEDGEMENTS

This study is conducted within the ICAROS (grant number 40384/03) and Agile-ITEA (grant number 40336/04) research projects both funded by TEKES (National Technology Agency of Finland). Sincere thanks are due to all the fellow researchers and software developers of the case projects.

REFERENCES

Abrahamsson P, Hanhineva A, Hulkko H, Ihme T, Jääliñoja J, Korkala M, Koskela J, Kyllönen P, Salo O. 2004. Mobile-D: an agile approach for mobile application development. *Proceedings of the 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'04)*, Vancouver, British Columbia, Canada.

Agile Alliance. 2001. Manifesto for Agile Software Development. <http://www.agilemanifesto.org/principles.html>, Accessed 29th of April, 2006.

Basili VR. 1989. Software development: a paradigm for the future. *Proceedings of the COMPSAC '89*, Orlando, Florida, 471–485.

Basili VR. 1994. The goal question metric approach. *Encyclopedia of Software Engineering*. John Wiley and Sons: New York, USA.

Beck K. 1999. Embracing change with extreme programming. *IEEE Computer* **32**: 70–77.

Beck K. 2000. *Extreme Programming Explained: Embrace Change*. Addison Wesley Longman: Reading, Massachusetts, USA.

Beck K. 2003. *Test-Driven Development by Example*. Addison Wesley: Boston, MA, USA.

Bergman B, Fundin A, Gremyr I, Johansson P. 2002. Beyond root-cause analysis. *Proceedings of the Annual Reliability and Maintainability Symposium, The International Symposium on Product Quality and Integrity*, Seattle, WA, 140–146.

Birk A, Dingsøyr T, Stålhane T. 2002. Postmortem: never leave a project without it. *IEEE Software* **19**: 43–45.

Boehm B, Turner R. 2003. Using risk to balance agile and plan-driven methods. *Computer* **36**: 57–66.

Cockburn A. 2002. *Agile Software Development*. Addison Wesley: Boston, MA.

Cockburn A. 2005. *Crystal Clear: a Human-powered Methodology for Small Teams*. Addison Wesley: Stoughton, MA, USA.

Collier B, DeMarco T, Fearey P. 1996. A defined process for project post mortem review. *IEEE Software* **13**: 65–72.

Cunningham JB. 1997. Case study principles for different types of cases. *Quality & Quantity* **31**: 401–423.

Dingsøyr T, Hanssen GK. 2002. Extending agile methods: postmortem reviews as extended feedback. *Proceedings of the 4th International Workshop on Learning Software Organizations (LSO'02)*, Chicago, IL.

Ihme T, Abrahamsson P. 2005. Agile architecting: the use of architectural patterns in Mobile Java applications. *International Journal of Agile Manufacturing* **8**: 97–112.

Kerlinger FN, Lee HB. 2000. *Foundations of Behavioral Research*. Harcourt College Publishers: Fortworth, Texas, USA.

Kerth NL. 2001. *Project Retrospectives: A Handbook for Team Reviews*. Dorset House Publishing: New York, USA.

Klein HK, Myers MD. 1999. A set of principles for conducting and evaluating interpretive field studies in information systems. *MIS Quarterly* **23**(1): 67–94.

Lau F. 1999. Toward a framework for action research in information systems studies. *Information Technology and People* **12**: 148–175.

Lycett M, Macredie RD, Patel C, Paul RJ. 2003. Migrating agile methods to standardized development practice. *Computer* **36**: 79–85.



- Malouin JL, Landry M. 1983. The miracle of universal methods in systems design. *Journal of Applied Systems Analysis* **10**: 47–62.
- Mann C, Maurer F. 2005. A case study on the impact of scrum on overtime and customer satisfaction. *Proceedings of the Agile 2005 Conference, Marriot Denver City Center, USA*.
- March ST, Smith GF. 1995. Design and natural science research on information technology. *Decision Support Systems* **15**: 251–266.
- McFeeley B. 1996. IDEAL(SM): a users guide for software process improvement. 222.
- Morgan DL. 1984. Focus groups: a new tool for qualitative research. *Qualitative Sociology* **7**(3): 253–270.
- Nonaka I, Hirotaka T. 1995. *The Knowledge-creating Company*. Oxford University Press: New York, USA.
- Salo O. 2004. Improving software process in agile software development projects: results from two XP case studies. *Proceedings of the 30th EUROMICRO Conference, Rennes, France*.
- Salo O. 2005. Systematical validation of learning in agile software development environment. *7th International Workshop on Learning Software Organizations (LSO 2005) in conjunction with the Third Biennial Conference of Professional Knowledge Management (WM 2005)*. In *Proceedings of Revised Selected Papers of Professional Knowledge Management*, Kaiserslautern, Germany.
- Salo O, Abrahamsson P. 2004. Empirical evaluation of agile software development: a controlled case study approach. *Proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES 2004)*, Kansai Science City, Japan, 408–423.
- Salo O, Abrahamsson P. 2005. Integrating agile software development and software process improvement: a longitudinal case study. *Proceedings of the 4th International Symposium on Empirical Software Engineering (ISESE 2005)*, Noosa Heads, Australia.
- Schwaber K. 1995. Scrum development process. *Proceedings of the OOPSLA'95 Workshop on Business Object Design and Implementation*, Austin, Texas, USA.
- Schwaber K. 2004. *Agile Project Management with Scrum*. Microsoft Press: Washington, DC.
- Scupin R. 1997. The KJ method: a technique for analyzing data derived from Japanese ethnology. *Human Organization* **56**: 233–237.
- Software Engineering Institute (SEI). 2002. CMMI® for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Staged Representation (CMMI-SE/SW/IPPD/SS, V1.1, Staged). Pittsburgh, OH, 729.
- Stålhane T, Dingsøy T, Hanssen GK, Moe NB. 2001. Post mortem – an assessment of two approaches. *Proceedings of the European Software Process Improvement*, Limerick, Ireland.
- Vandeville JV. 2000. Organizational learning through the collection of "Lessons Learned". *Informing Science* **3**: 127–133.