Extreme Programming: A Survey of Empirical Data from a Controlled Case Study

Pekka Abrahamsson and Juha Koskela

VTT Technical Research Centre of Finland
P.O. Box 1100, FIN-90571 Oulu, Finland
{pekka.abrahamsson; juha.koskela}@vtt.fi

Abstract

Extreme programming (XP) is a well known agile software development method. While a number of experience reports have been published in recent years, agile software development in general and XP in particular have strongly been criticized for the lack of empirical data. This paper reports a survey of the empirical data obtained from a controlled case study on extreme programming in practical settings. Thus, no hypotheses were set a priori. Four software engineers were acquired to implement a web-based system (7698 Locs, 820 hours) for data management in a delivery schedule of eight weeks. Development environment was close to the agile home ground. Collected empirical data is grounded on three basic data points: Time, size and defects. Data is organized around five system releases, each which were tested by 17 customer testers. System release defect-density was 1.43 defects/KLOC, team overall productivity 16.90 Locs/hour and rework costs were 9.8% of the total development effort. The implications of this study are discussed.

1. Introduction

Agile methods and principles have gained a significant amount of attention in the field of software engineering in just few years. The roots of agile software development can be traced back as early as 1960's and even beyond [1]. The starting point for the movement, however, was actually in mid 1990's [2]. Since then, several methods [for an overview, see 3] have been developed. All of these methods claim conformance to agile principles put forward in agile manifesto (http://www.agilemanifesto.org).

Extreme programming (XP), a method developed by Beck [4], is one of the better known agile methods. While a number of XP books [e.g., 5, 6-8] and experience reports [e.g., 9, 10-12] have been published, less is known about the empirical and scientific validity of the method [13-15]. The situation is not unique in software engineering. While the necessity of empirical software engineering is acknowledged [16-18], far too often the application decisions made in practice remain without any empirical justification [19, 20]. As a result of the situation, agile methods such as XP have been placed under severe critique for e.g. embracing the hacker's culture and thus neglecting the product and process quality viewpoints [21]. Furthermore, lack of empirical data hinders the ability to apply XP and modify it for different settings and domains.

The aim of this paper is to set references for the use of researchers and practitioners in the field. This will be achieved by reporting empirical data grounded on three basic data points: Time, size and defects [22]. The data was obtained from controlled case study where the XP process and the resulting product quality have been under careful scrutiny.

A team of four developers was acquired to implement a system (code-named for eXpert) for managing the research data obtained over years at a large Finnish research institute. The development schedule and resources were fixed. Flexibility was reserved to the delivered functionality. The requirements for the system were not, however, well known before the project was initiated due to large number of potential users, i.e. 300+, and their differing views.

The reported data is organized around five system releases, each which were tested by 17 customer testers. While several points of comparison were established, the more interesting were the system release phase defect-density (1.43 defects/KLoc), team overall productivity (16.90 Locs/hour), rework costs (9.8% of the total development effort) and required actual involvement for the on-site customer (21%). The contents of the first two release cycles have been compared in [23]. This paper extends these early results to include the whole project as well as discusses the possible implications more in-depth.

The paper is organized as follows. The following section introduces in brief the purpose of the XP method. This is followed by a description of how the research was performed, the results and the discussion. The paper is concluded with final remarks.

2. Extreme Programming

Extreme Programming is one of several agile software development methods that have emerged in the past few years. XP was first introduced in [24]. The method focuses on delivering immediate business value to the customer. The XP process can be characterized by short development cycles, incremental planning, evolutionary design, and its ability to response to changing business needs. The method itself is built around what appears to be easy-to-understand set of practices, which have been fairly well documented in the literature (see references for details). These practices are planning game, small releases, metaphor, simple design, testing (test-driven development), refactoring, pair programming, collective ownership, continuous integration, 40-hour work week (also known as sustainable pace) and on-site customer, just rules and open workspace. In addition, spikes [7] is also often associated to the XP method's practices.

The XP method is designed to meet the needs of a skilled small, i.e. less than 10 developers, team that is working in a co-located office together with the customer developing software that is not safety-critical on an object-oriented technology [4, 25]. This type of situation is what can be called an ideal surrounding for the XP method or what Boehm [26] calls an agile home ground. This case study falls within this description.

3. Research design

This section describes how the research design for the study is laid out.

3.1. Research method

The title of the paper indicates the use of a case study research method [e.g., 27]. However, the boundaries between different research methodologies and data collection techniques are often overlapping to certain extent [28]. Cunningham, [29] for example, relates action research as one form of case study research. In action research the focus is more on what practitioners do rather than what they say they do [30]. This is the position taken in this study. Moreover, Järvinen [31] follows Oquist [32] and argues that action produces knowledge to guide practice, which is the principal aim of this study. In action research, the modification of reality requires the possibility to intervene [33]. The first author was in the role of management in the study and mediated the release post-mortem analysis [34] sessions, which were performed after each software release. The second author acted in the role of on-site customer and participated to planning game, acceptance testing, post-mortem analysis, project meetings and coaching activities. On average, he spent over 80% of his work time in the same room with the developers. The process change mechanism was systematized as well. The process was changed through the post-mortem analysis technique where the project team proposed changes to the implementation process. Thus, the origins for modification of reality came from the project team, not from the researchers.

The term "controlled" in the paper's title is used intentionally. Empirical studies include various forms of research strategies [35]. "Controlled" is most often associated with the experimentation approach. One central difference between research strategies is the *level of control*. Following Wohlin [36, p. 12] "experiments sample over the variables that are being manipulated, while the case studies sample from the variables representing the typical situation". If this is accepted, the experimentation approach can be seen as "a form of empirical study where the researcher has a control over some of the conditions in which the study takes place and control over the independent variables being studied" [35, p. 456]. Therefore, the use of term "controlled" in this study implies that the researchers were in a position to design the implementation environment, i.e. the typical situation (see next subsection of research setting), beforehand.

3.2. Research setting

A team of four developers was acquired to implement an eXpert system for managing the research data obtained over years at a Finnish research institute. A metaphor that better describes the intended purpose of the system is a large sized "virtual file cabinet", which holds a large number of organized rich, i.e. annotated, links to physical or web-based resources. The system has 300+ potential users and is a web-based client-server solution.

The four developers were 5-6th year university students with 1 to 4 years of industrial experience in software development. Team members were well-versed in the java programming language and object-oriented analysis and design approaches. Two weeks prior to project launch the team performed a self-study by studying two basic books on XP [i.e., 4, 7]. A two day hands-on training on XP practices, the development environment and software configuration management tools was organized to ensure that the team has a basic understanding on XP issues and the technical environment. Thus, this study focuses on a development team that is novice to extreme programming practices.

The team worked in a co-located development environment. The customer (i.e., a representative from the research institute) shared the same office space with the development team. The office space and workstations were organized according to the suggestions made in the XP literature to support efficient teamwork. Unused bookshelves, as an example, were removed in order to have a maximum amount of empty wall space for user stories and task breakdowns, architecture description, etc.

Table 1 shows the details of the technical environment used for the development of eXpert system.

Item	**Description**
Language	Java (JRE 1.4.1), JSP (1.2)
Database	MySQL (Core 4.0.9 NT, Java connector 2.0.14)
Development Environment	Eclipse (2.1)
SCM	CVS (1.11.2); integrated to Eclipse
Unit testing	JUnit (3.8.1); integrated to Eclipse
Documents	MS Office XP
Web Server	Apache Tomcat (4.1)

Table 1. Technical implementation environment

Development environment was an Eclipse integration framework (www.eclipse.org), which is an open source initiative supported by major software engineering tool manufacturers. It is based on tool developed by IBM. CVS (Concurrent Versions System) was used as project's SCM tool and JUnit testing framework for unit testing. Both the CVS client and JUnit are integrated as a default in Eclipse environment. The system was written in Java and JSP (JavaServer Pages) and it uses MySQL relational database in storing the data of links. In addition, the Apache Tomcat 4 Servlet/JSP container was used because it implements JSP 1.2 specifications from Java Software.

3.3. Data collection

Both quantitative and qualitative data were collected. As stated earlier, quantitative data was grounded on three basic data points, i.e. time, size and defect, as suggested by Humphrey [22]. While a number of other interesting data points could have been captured, these three metrics were seen to be the most beneficial for setting some references for other researchers and practitioners.

Developers collected effort usage for each defined task and XP practice with a precision of 1 minute using paper/pen and predefined excel-sheet as the primary collection tools. It was acknowledged that merely targeting for high precision *per se* does not necessarily improve data collection accuracy [22]. The on-site customer ensured on his behalf that the metrics collection procedure was actualized as planned. In addition, a daily inspection on the metrics collected was initiated. Finally, the project manager for the project was a PSP trained engineer who understood the basics of PSP style of metrics collection. PSP was not, however, utilized as the development device.

Development work size, i.e. logical lines of code [22], was collected on daily basis using automated counters for Java and JSP. Development time defects (including type, severity), post-release defects (found by 17 allocated system testers) and the number of enhancement suggestions made by testers were also systematically recorded. Work commit size was drawn from the CVS tool.

As stated, the quality of the data obtained was systematically monitored by the project manager, dedicated metrics responsible, on-site customer and the customer organization's management. As indicated by XP principles [4], customer organization placed explicit value on the data collection, thus ensuring the alignment with the agile software development principles.

Qualitative data included development diaries maintained by the developers, post-mortem analysis session recordings and developer interviews.

4. Results

Table 2 shows the data obtained from the five system releases and the correction release, which was performed after the system test. The total column shows the cumulative/average data from the releases. The first three (row 1 in the table) releases were two weeks in calendar time, the last two were one week and the correction release took two days to complete. Term "release" is used instead of "iteration" to stress the fact that the system was released to actual customer test.

Total work effort (row 2) dedicated to project work remained constant in the first three releases. However, the direct hours dedicated to tasks (row 3) was reduced from the initial 70% to 50-60% in 2-week release cycle and below 50% in one-week cycle indicating an increase in over-head in very short development cycles. None-task allocated work was the effort spent to planning-game, post-mortem analysis, data collection, project meetings, brainstorming, coaching and pre-release testing.

Row 4 shows the amount of new logical lines of code the team produced in a release. Team's productivity (row 5) varied somewhat from 9.02 to 25.12 Loc/hour. This can be partly explained. First release contained tasks not related to user functionality such as finalizing the technical set up of the development environment. In the fourth release, the development team focused on documentation as well as did a lot of refactoring work, which is meant to remove code duplicates and simplify design solutions. Activities as such do not contribute to team productivity when measured primarily in terms added new lines of code. This could have been improved by counting also modified and deleted lines of code.

Rows 6-8 reveal essential SCM data from the project. The number of code integrations remained relatively constant over the project varying between 7.9 and 10.5 code integrations in one working day. Average time between the integrations was between 21 and 40 minutes and finally average number of files associated with the integration varied between 1.7 and 3.1.

Rows 9-12 deal with the user stories, which are customer defined functional or non-functional requirements for the system. EXpert system did not have any explicit non-functional requirements defined. The team velocity, i.e. number of user stories included in a single release, increased and stabilized rapidly. The median user story size decreased from 10.1 hours to around 5 hours indicating that the requirements were broken down in a high level of accuracy. The reason for this was that the largest size of a user story in the first release was 63.1 hours, which was seen to be too large by the development team. In the second release, the largest user story size was reduced to 26.9 hours although increased back to 41.7 hours in the third release. Yet, improvement is visible when observed from the task level (rows 13-15). While only 10 tasks were defined for the first release, the second release contained already 30 tasks. Even the fourth and the fifth release, which were only one-week in calendar time, had 21 and 19 tasks defined, respectively. This demonstrates the team's increased ability to decompose requirements into more manageable level. Importantly the median size of a task was reduced from 11.7 to less than 3 hours, and the maximum size of a task was reduced from 32.3 to less than 10 on average.

Rows 16-17 are concerned with the quality of the system. 17 testers were allocated for a brief, i.e. max 45min, and intensive, i.e. testing was to be performed within four hours from system release, user functionality test.

Id	Collected data	Release 1	Release 2	Release 3	Release 4	Release 5	Correction release	Total
1	Calendar time (weeks)	2	2	2	1	1	0.4	8.4
2	Total work effort (h)	195	190	192	111	96	36	820
3	Task allocated actual hours	136 (70%)	95 (50%)	118 (61%)	51 (46%)	42 (44%)	27 (75%)	469 (57%)
4	# LOCs implemented in a release	1821	2386	1962	460	842	227	7698
5	Team productivity (loc/hour)	13.39	25.12	16.63	9.02	20.05	8.4	16.90
6	Code integrations (integrations/day)	8.1	10.1	7.9	10.5	8.2	8.5	8.9
7	Avg. time between integrations (minutes)	26	21	40	31	27	30	29
8	Avg. number of files per integration	1.7	2.4	3.1	2.6	3.0	3.0	2.6
9	# User stories implemented	5	9	9	4	3	4	34
10	# User stories postponed for next release	0	1	0	1	2	0	4
11	User story effort (actual, median, h)	10.1	8.3	7.6	5.9	5.2	2.8	6.8
12	User story effort (actual, max, h)	63.1	26.9	41.7	21.8	15.9	7.6	63.1
13	# Tasks defined	10	30	18	21	19	9	107
14	Task effort (actual, median, h)	11.7	2.9	5.9	1.7	2.6	0.7	2.7
15	Task effort (actual, max, h)	32.3	8.8	14.0	8.8	5.3	3.4	32.3
16	# post-release defects	4	5	4	4	11	-	28
17	Post-release defects/KLoc	2.19	2.10	2.04	8.70	13.06	-	1.43 (3.75)
18	# post-release enhancement suggestions made by testers	17	13	5	3	0	-	38
19	Pair programming (%)	81.7	76.3	73.0	78.8	54.2	90.4	75.9
20	Required customer involvement (%)	17.4	21.4	18.6	25.0	23.4	24.3	20.6
21	Rework costs (%)	-	8.7*	11.8	11.6	2.6	61.5	9.8

*includes also enhancements

Table 2. Exploratory data from 5+1 releases

Testers did not follow a predefined pattern for testing or reporting. They worked on a volunteer basis and were using ad-hoc testing routines and reports. Therefore, the post-release system defect densities call for special attention. Release three was tested to a dissatisfactory level, i.e., less than 10 testers reported their results. Attention was given to this issue and four (of the 17) testers were taking a more systematic approach from that point forward. These four testers performed a two-day planned system test for the whole system after the fifth release. Yet, despite of these concerns, the total defect density for the system was relatively low, i.e. 1.43 defects/KLoc. 1.43 is calculated from the 11 defects that were discovered in the system test of which four were cosmetic and two out of the remaining seven were crash defects. Overall, the defect density was evaluated to be close to satisfactory giving an indication of the product quality. In addition, testers produced altogether 38 improvement suggestions, i.e. new or improved user functionality. Majority of these suggestions emerged from the first two releases.

Pair programming (row 19) was extensively practiced in the development of the first release (81.7%) and was not significantly reduced in the following releases. In the system correction phase, over 90% of the programming work was performed in pair programming mode. It is notable that none of the project team members had any systematic experience in pair programming prior eXpert project.

While the customer shared the same office with the development team and thus was present over 80% of the total time, the actual customer involvement (row 20) was only 21% on average. This is a significant result since on-site customer is one of the most controversial topics in extreme programming methodology.

Row 21 shows the rework costs associated with the eXpert project. Altogether 45.8 hours were used to fix defects, which is 9.8% of the total development effort.

Figure 1 displays the overall effort distribution for the whole project. Data shows that in this project roughly 10% is required for planning the release contents. Project management activities, which include data collection & analysis, monitoring the progress of the project and the development of project plan required 13.4% of the total effort. This figure may be slightly overloaded since the management structure in this project included also two steering group meetings into which the whole project team participated.

As suspected, coding in terms of unit test development[1], production code, development spikes and refactoring took the majority, i.e. 54.7%, of the total effort. Yet, the proportion of actual coding is less than the expectations put forward in the popular XP literature. Project meetings took 4.5% of the total effort.

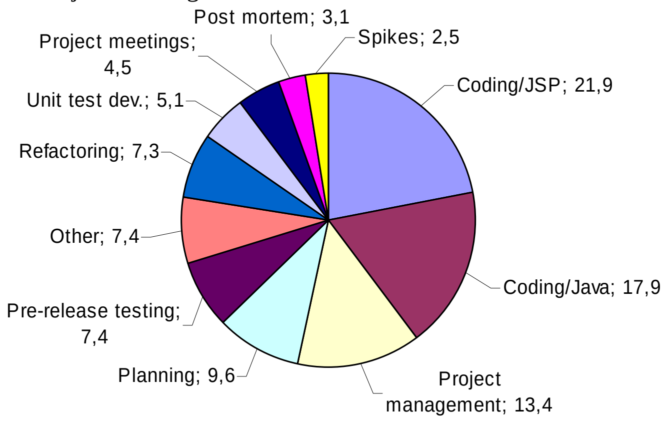

Figure 1. Effort distribution (%)

All the design documentation including architectural description were displayed on the walls of the development room. User stories and tasks were documented and displayed on the same walls as well. The simple design practice involved in the pair programming coding was not separately tracked. The effort, i.e. 7.4%, embedded into "other" pie includes discussions with the customer organization's management and some other miscellaneous tasks, such as system documentation, see below.

The system documentation (architecture description, on-line helps, user manual and database description) for the maintenance purposes were produced in the last two releases when the system architecture and the user functionality was stabilized and was seen to be less subject to constant changes. However, since the effort was tracked based on predefined XP practices and tasks, the effort spent on system documentation is loaded on a pie labeled "other". XP literature has argued that the first iteration for a novice XP team is always a significant learning effort. Literature has further shown that an ability to estimate accurately is a skill that is learned over time [22]. As expected, the team had major difficulties in decomposing the contents of the first release into tasks (only 10 tasks defined for 5 user stories). Similarly, the team had major difficulties in producing the estimates for these tasks.

[1] Note that no unit tests were developed for JSP code. Unit test development shown in Figure 1 is with respect to Java code.

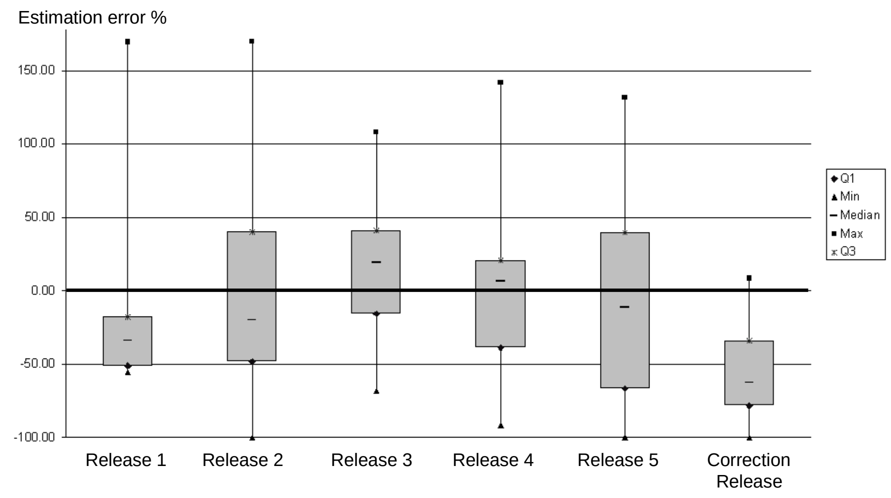

Figure 2. Estimation accuracy

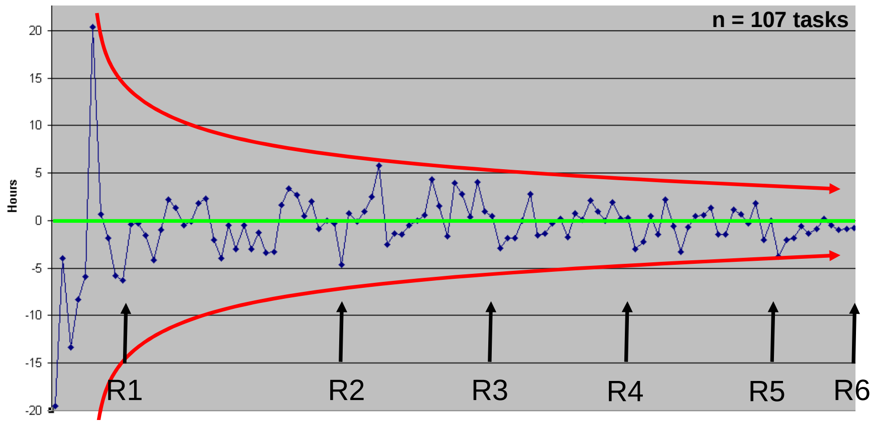

Figure 3. XP Pulse: Hours lost by faulty estimates

User story effort estimates were derived from the task estimates. Figure 2 shows a series of box plots[1] depicting the estimation accuracy development over the system releases. The data used for the drawing the box plots is the task level data. Altogether 107 tasks were defined in the eXpert project. Data below the thickened line indicates overestimation and data above the line refers to underestimation of the tasks.

When Figure 2 is carefully studied, several observations can be made. Estimation error variance remains high throughout the project. While median value shows clear improvement, i.e. it approaches value 0, estimation errors are common place even in the last releases. The concentration of the data points does not show clearly identifiable improvement. Statistical data treatment was not seen feasible beyond this point in this study.

While estimates are not accurate in terms of error percentage, Figure 3 indicates clearly identifiable improvement in terms of hours lost by faulty estimates. The thick line indicates a loss of zero hours. Data points below the thickened line indicate that a particular task was finished earlier than expected. Data points above the thickened line indicate that a particular task took longer than expected. The tendency and trend observed from the diagram indicate certain type of heart beat inherent to this type of an XP project. This is where the term “XP Pulse” originates. Thus, even though estimations are faulty to certain extent, the time lost by these “guesses” is predominantly (i.e., 92%) within +/- 5 hours altogether after the first release demonstrating a rapid ability to gain

[1] A box plot diagram visualises the five-number summary of a data set. Median (a line in the shaded area) value indicates that 50% of data points are below and 50% are above the line. Q1 (first or lower quartile) shows the median of the lower 50% of data points. Q3 (third or upper quartile) shows the median of upper 50% of data points. The minimum value indicates the lowest and the maximum the highest values in the respective data sets.

control over the project. The reason for this can be found from Table 2 (row 14) where actual median effort spent on a single task is shown. The development team intentionally strove for smaller task segments in order to improve their work control mechanisms. Actual effort spent on a single task was cut to half from the first release. Similarly, the user stories were divided into task segments, which were sized between 4-8 hours on average. Therefore, even if a certain task takes few hours extra than originally expected, it is not likely to disrupt the development process since data shows that in many cases tasks are finalized with less effort than expected.

It should be noted that overestimation can become a problem if resources are allocated on this basis. However, in this particular case, the XP team mitigated the estimation problem by defining considerably smaller tasks. This is evidenced in the data also by the number of tasks identified (Table 2, row 13). The project manager was able to take necessary actions on daily rather than on weekly basis. This data also hints to a direction that the user stories were better defined due to required small-sized task definitions.

5. Discussion

The results presented in the previous section emphasized the differences between different system releases in the eXpert project from several empirical viewpoints all of which were drawn from three data points, i.e. time, size and defect. The mere amount of valid empirical data obtained demonstrates the ability to collect data in agile software development in general and in XP in particular. The purpose of this paper is to set some references for researchers and practitioners in the field from empirical viewpoint. Less emphasis has been placed here on the qualitative data.

One of the important findings is related to the little need for actual customer involvement in the project. This finding is not in line with the XP literature. Many authors [e.g., 37, 38-40] maintain that on-site customer involvement is difficult and even unrealistic to think of due to required customer work effort. In the eXpert case, the customer was present on-site on average over 80% of the total working time. However, only 21% of his work effort was required to assist the development team in the development. Majority of customer's involvement was required on planning game (42.8%) and acceptance testing (29.9%). In the eXpert case the customer did not develop automated acceptance tests but performed them manually. However, the mere presence of the customer was highly appreciated by the development team. Customer presence appeared as a positive indicator in the post-mortem analysis sessions and interviews. The team viewed that the customer organization values the system high and this was seen to work as a motivating factor for the team. Thus, regardless of the required effort usage, on-site customer can be seen as an important stakeholder in the project. It should be further emphasized that in this case, the customer did not develop acceptance tests. He was performing these tasks manually at the end of each release cycle.

As stated earlier, the system release defect density was 1.43 defects/Kloc. The system was released on Friday at noon. By following Monday morning, an early insight was obtained to development phase release's defect density. This was monitored throughout the project (Table 2, row 17). Defect density showed slight increase when the development cycle was reduced to mere one week. Yet, the level remained very low throughout the project. This result can be seen positive from three perspectives. First, an early insight was gained to the overall product quality. If more bugs would have been discovered, necessary actions could have been taken rapidly to mitigate the problem with this aspect of quality.

Second, the testing team formed a part of the user group who will make use of the system when it is finally released. Thus, apart from testing the user functionality, they had an opportunity to influence the content of the future releases. Research has shown that user involvement in the systems development process has a positive impact on the subsequent system adoption and use [41]. This is also inline with the XP procedures. In the five system releases altogether 38 new or enhanced user functionality suggestions were received. 32 (84%) of them lead to action. The large percentage demonstrates the value of this type feedback system. It should also be noted that 30 (79%) of 38 improvement suggestions were known already after the second release. Involving users as testers can be seen as a form of requirements elicitation procedure. Moreover, the eXpert testers were able to observe how the development proceeds from the very first release to the final fully functional system. However, it should be further noted that a one week release cycle to end-user testing was not totally appreciated by the users. In some cases it was seen disturbing to have a new version at hand in one week's time.

The third advantage related to low defect density and the testing process is the rapid feedback acquired for the development team. The customer representative collected and categorized the suggestions (and bugs) reported over the weekend and presented the results on the following Monday to the development team – another explicit sign of strong customer commitment toward the implementation project.

Pair programming is one of the most researched XP practices [see e.g., 42, 43-48]. Only few studies have, however, provided data on pair programming over longer period of time. This study contributes to empirical knowledge particularly in this regard. As noted, in the first release, 81.7% of the programming effort was done in pairs. This was reduced only slightly in the following releases averaging to 75.9%. Williams [43] has argued that only after having effectively experimented with the pair programming practice, an estimation can be made where it delivers the most value and where it proves ineffective. Clearly, two weeks is not sufficient for a through evaluation of a single practice but due to the tight delivery schedule, the team was able to make decisions regarding each practice in the post-mortem analysis session performed after each release. However, the fact that the pair programming time remained well above 70% in the following releases demonstrates that the team felt comfortable with it. Williams and Kessler [44] remind that no one should be forced to use pair programming. In eXpert case, the base process provided for the team in the beginning of the project *required* the application of pair programming during the first release. After that, it was up to the team to make the decision. Developers did not track defects that were caught in pair programming mode.

Coding effort pattern that was used for pair and solo programming did not significantly differ as shown in Table 3. As noted earlier no unit tests were developed for JSP code. In the solo programming mode slightly more effort was spent on refactoring than on pair mode. Yet, the difference is not remarkable.

	Spike code	**Unit test dev.**	**Java code**	**JSP code**	**Re-factoring**
Pair	4.8%	9.8%	34.5%	38.4%	12.4%
Solo	4.1%	7.5%	27.6%	44.9%	15.9%

Table 3. Effort use % in pair and solo programming

Interestingly, the development time productivity achieved in the second release (i.e., 25.12 Loc/hour) is close to the same as e.g. PSP research [49] has consistently shown. The data obtained in this study does not show a relation between the use pair programming and the level of achieved productivity. The refactoring data, on the other hand, may reveal interesting insights. The highest levels of productivity are achieved when only 5.9% of the effort was used for refactoring. Yet, as Table 4 shows this tendency is not consistent with the rest of the releases and only indirectly provides support for the argument that extensive levels of refactoring decreases team productivity. Kivi et al. [9] suspect that most of the work towards the end of the project may be refactoring. Our findings show that the levels of refactoring increase somewhat but not to a significant degree. In fact, the last system release (i.e., Release 5) shows a reduction in the proportion of time used for refactoring. In the project post-mortem analysis session after the project had been completed the team evaluated that not enough of refactoring was done for the system and that it should be explicitly involved in the user stories as a separate task.

	R1	**R2**	**R3**	**R4**	**R5**	**CR**
Productivity (Loc/hour)	13.39	25.12	16.63	9.02	20.05	8.4
Refactoring %	13.8	5.9	18.1	18.0	11.2	0

Table 4. Productivity vs. Refactoring

In software engineering in general, accurate effort estimates are difficult to attain [22] Initial estimates can often be better regarded as “guesstimates” [50] Regarding the XP process of producing the estimates, McBreen [21, p. 60] was especially doubtful about the value of XP planning game: “The accuracy of the estimates produced during the planning game needs to be investigated, especially for organizations that are just adopting XP. [...] I wonder how long it takes a new XP team to get good at the Planning Game.” This paper gives concrete results in this regard. The estimation accuracy was improved in terms of estimation error. The estimation accuracy stabilized within +/- 25% error margin (suggested by Humphrey [22]) and mis-allocated development time due to inaccurate estimations was also reduced on average from 5.8h to 0.15h. Langr [51] argues that “Initial estimates are going to be inaccurate in any process. In XP, the team has lots of opportunities to estimate and to learn how to do it well – the team hones their estimating skills every two weeks.” This study thus supports Langr’s suggestion. Our findings indicate further that a novice XP team is very careful about making too optimistic estimates. However, it took them only two weeks to realize this. Learning to execute the planning game routines was facilitated by a clear agreement on the procedures, roles and responsibilities.

Only four (9%) out of 34 user stories were postponed for the next release. The postponement did not come as a surprise for the customer since he was present, and he gave the approval after having consulted with the management about the change in the release contents. It was agreed before the project started that the release date should not be postponed but the content can be negotiated if the team so desires. This is in line with what Humphrey [52] discusses about making a commitment and keeping it. The team commits to the release schedule and the contents are negotiated. If any changes are to be made, an early warning must be given.

Finally, in the eXpert project the rework costs were 9.8% of the total development effort. Beck’s [4] basic argument in his seminal work was that in XP the cost of change does not show a dramatic increase when the development progresses beyond certain point. Instead, it should remain stable over the project. To our knowledge no concrete figures can be found in the XP literature or elsewhere to support this argument. Our findings support Beck's claim to certain extent. Yet, a system with less than 10000 lines of code can not be seen to constitute a significant support to this issue. It rather indicates a direction that should be investigated further.

6. Conclusions

Agile movement has gained significant attention in the software engineering community in the last few years. While concrete data about the various aspects of the XP process are emerging, less data is available regarding the resulting product. This may be due to the fact that companies are not willing to reveal these details. This paper aims at setting some references for researchers and practitioners in the field by reporting concrete data from a controlled case study where a team of four developers was acquired to develop a fully operational system for a research institute's research data management and distribution purposes in a delivery schedule of eight weeks. Due to the tight schedule, the functionality was not fixed. The concrete results reported are based on five system releases and a correction release. The resulting product was tested by 17 testers who used a maximum of 45 minutes to test the defined user functionality. The data shows that the system release defect rate was 1.43 defects/Kloc, team's overall productivity was 16.90 Locs/hour, the rework costs were mere 9.8% of the total development effort, and the required actual involvement for the on-site customer was 21%. These and other findings were addressed.

It should be noted that the team collected more data about their work than is the case usually. This was achieved by placing value on the data collection. This is in accordance to basic values of agile thinking. The development team delivers business value for the customer organization. If the client organization does not require detailed data about the development process, the team may easily fall behind in data collection. For this, certain data collection rules must be established within the development team. In eXpert case, the team decided use the simplest solution and to put up a sheet on the wall where each developer will sign their name after the data has been recorded at the end of the working day to ensure that the commitment to collect and record data has been met.

We will continue to follow up and monitor the system-in-use defects. Several other research streams have also been initiated. The resulting product quality has been assessed by an external team of assessors, the used development process has been assessed by an external CMMI assessor and an independent user-centered design assessment has been performed both on the development process and the product. These results along with the qualitative data will be published separately.

We find that the data reported in this paper is of value for practitioners and researchers in the field and will provide some references for future studies as well industry application within similar domain of application. We suggest that more concrete data is needed for the agile movement to progress beyond the practicing enthusiastic. Indeed, a separate framework is needed, which guides the agile and XP data reporting. To meet these needs, Williams et al. [53] published recently a benchmark measurement framework for researchers and practitioners to express concretely the XP practices that have been selected to adopt, and the outcome thereof. This framework and studies like the one reported in this paper can be seen to provide a starting point for strengthening the empirical body of evidence in agile software development.

References

[1] C. Larman and V. R. Basili, "Iterative and incremental development: A brief history," *IEEE Software*, vol. 20, pp. 47-56, 2003.

[2] L. Williams and A. Cockburn, "Agile software development: It's about feedback and change," *IEEE Software*, vol. 20, pp. 39-43, 2003.

[3] R. Turner and B. Boehm, *Balancing agility and discipline: A guide for the perplexed*. Boston, MA: Addison-Wesley, 2003.

[4] K. Beck, *Extreme programming explained: Embrace change*. Reading, MA.: Addison Wesley Longman, Inc., 2000.

[5] G. Succi and M. Marchesi, "Extreme Programming Examined: Selected Papers from the XP 2000 Conference," presented at XP 2000 Conference, Cagliari, Italy, 2000.

[6] R. M. a. J. Newkirk, *Extreme Programming in Practice*: Addison-Wesley, 2001.

[7] R. Jeffries, A. Anderson, and C. Hendrickson, *Extreme Programming Installed*. Upper Saddle River, NJ: Addison-Wesley, 2001.

[8] R. Hightower and N. Lesiecki, *Java Tools for Extreme Programming*. New York: Wiley Computer Publishing, 2002.

[9] J. Kivi, D. Haydon, J. Hayes, R. Schneider, and G. Succi, "Extreme Programming: a University Team Design Experience," presented at CCECE 2000 - Canadian Conference on Electrical and Computer Engineering, Nova Scotia, NS, USA, 2000.

[10] P. Schuh, "Recovery, Redemption, and Extreme Programming," *IEEE Software*, vol. 18, pp. 34-41, 2001.

[11] D. Wells and T. Buckley, "The VCAPS project: An example of transitioning to XP," in *Extreme programming examined*, G. Succi and M. Marchesi, Eds. New York: Addison-Wesley, 2001, pp. 399-422.

[12] P. Sommerlad, "Adopting XP," in *Extreme programming examined*, G. Succi and M. Marchesi, Eds. New York: Addison-Wesley, 2001, pp. 423-432.

[13] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams, and M. Zelkowitz, "Empirical findings in agile methods," presented at XP/Agile

Universe 2002, Chicago, USA, 2002.

[14] G. Melnik, L. Williams, and A. Geras, "Empirical Evaluation of Agile Processes," presented at XP/Agile Universe 2002, Chicago, USA, 2002.

[15] P. Abrahamsson, J. Warsta, M. T. Siponen, and J. Ronkainen, "New directions on agile methods: A comparative analysis," presented at International Conference on Software Engineering (ICSE25), Portland, Oregon, 2003.

[16] V. R. Basili, "The role of experimentation in software engineering: Past, present and future," presented at Keynote address in 18th International Conference on Software Engineering (ICSE18), Berlin, Germany, 1996.

[17] V. R. Basili, "Using experiments to build a body of knowledge," presented at Keynote address in Seventh European Workshop on Software Process Technology (EWSPT-2000), Kaprun, Austria, 2000.

[18] W. F. Tichy, "Should computer scientists experiment more?," *IEEE Computer*, pp. 32-40, 1998.

[19] N. Fenton, "Viewpoint Article: Conducting and presenting empirical software engineering," *Empirical Software Engineering*, vol. 6, pp. 195-200, 2001.

[20] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," *IEEE Computer*, pp. 23-31, 1998.

[21] P. McBreen, *Questioning extreme programming*. New York: Addison-Wesley, 2001.

[22] W. S. Humphrey, *A discipline for software engineering*. Reading, Mass.: Addison Wesley, 1995.

[23] P. Abrahamsson, "Extreme programming: First results from a controlled case study," presented at Euromicro 2003, Antalya, Turkey, 2003.

[24] K. Beck, "Embracing change with extreme programming," *IEEE Computer*, pp. 70-77, 1999.

[25] M. Lippert and S. Roock, "Adapting XP to complex application domains," presented at The 8th European software engineering conference, 2001.

[26] B. Boehm, "Get Ready For The Agile Methods, With Care," *Computer*, vol. 35, pp. 64-69, 2002.

[27] R. K. Yin, *Case Study Research Design and Methods*, 2nd ed ed: Sage Publications, 1994.

[28] T. D. Jick, "Mixing qualitative and quantitative methods: Triangulation in action," *Administrative Science Quarterly*, vol. 24, pp. 602-611, 1979.

[29] J. B. Cunningham, "Case study principles for different types of cases," *Quality and quantity*, vol. 31, pp. 401-423, 1997.

[30] D. Avison, F. Lau, M. Myers, and P. A. Nielsen, "Action Research," *Communications of the ACM*, vol. 42, pp. 94-97, 1999.

[31] P. Järvinen, *On research methods*. Tampere: Juvenes-Print, 2001.

[32] P. Oquist, "The epistemology of action research," *Acta Sociologica*, vol. 21, pp. 143-163, 1978.

[33] G. I. Susman and R. D. Evered, "An Assessment of the Scientific Merits of Action Research," *Administrative Science Quarterly*, vol. 23, pp. 582-603, 1978.

[34] T. Dingsøyr and G. K. Hanssen, "Extending Agile Methods: Postmortem Reviews as Extended Feedback," presented at 4th International Workshop on Learning Software Organizations, Chicago, Illinois, USA, 2002.

[35] V. R. Basili and F. Lanubile, "Building knowledge through families of experiments," *IEEE Transactions on Software Engineering*, vol. 25, pp. 456-473, 1999.

[36] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Boston: Kluwer Academic Publishers, 2000.

[37] C. Farell, R. Narang, S. Kapitan, and H. Webber, "Towards an Effective Onsite Customer Practice," presented at XP 2002, Sardinia, Italy, 2002.

[38] A. Martin, J. Noble, and R. Biddle, "Being Jane Malkovich: A Look Into the World of an XP Customer," presented at XP 2003, Genoa, Italy, 2003.

[39] L. A. Griffin, "A Customer Experience: Implementing XP," presented at XP Universe, Raleigh, NC, USA, 2001.

[40] T. Bozheva, "Practical Aspects of XP Practices," presented at XP 2003, Genoa, Italy, 2003.

[41] J. Hartwick and H. Barki, "Explaining the Role of User Participation in Information System Use," *Management Science*, vol. 40, pp. 440-465, 1994.

[42] J. Nawrocki and A. Wojciechowski, "Experimental evaluation of pair programming," presented at ESCOM 2001, London, UK, 2001.

[43] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the case for pair programming," *IEEE Software*, vol. 17, pp. 19-25, 2000.

[44] L. Williams and R. Kessler, *Pair programming illuminated*. New York: Addison-Wesley, 2003.

[45] G. Succi, W. Pedrycz, M. Marchesi, and L. Williams, "Preliminary analysis of the effects of pair programming on job satisfaction," presented at XP 2002, Alghero, Sardinia, Italy, 2002.

[46] A. Janes, B. Russo, P. Zuliani, and G. Succi, "An empirical analysis on the discontinuous use of pair programming," presented at XP 2003, Genoa, Italy, 2003.

[47] S. Heiberg, U. Puus, P. Salumaa, and A. Seeba, "Pair-programming effect on developers productivity," presented at XP 2003, Genoa, Italy, 2003.

[48] K. M. Lui and K. C. C. Chan, "When does a pair outperform two individuals," presented at XP 2003, Genoa, Italy, 2003.

[49] W. Hayes and J. W. Over, "The Personal Software Process (PSP): An Empirical Study of the Impact of PSP on Individual Engineers," Software Engineering Institute, CMU/SEI-97-TR-001, Technical Report CMU/SEI-97-TR-001, http://www.sei.cmu.edu/publications/documents/97.reports/97tr001/97tr001abstract.html, 1997.

[50] P. M. Johnson, C. A. Moore, J. A. Dane, and R. S. Brwer, "Empirically guided software effort guesstimation," *IEEE Software*, vol. 17, pp. 51-56, 2000.

[51] J. Langr, "Book review: Questioning extreme programming," vol. Page accessed March 10, 2003: XProgramming.com: An Extreme Programming Resource, 2002.

[52] W. S. Humphrey, *Managing the Software Process*. Reading, Mass.: Addison-Wesley, 1989.

[53] L. Williams, W. Krebs, L. Layman, A. I. Antón, and P. Abrahamsson, "Toward a framework for evaluating extreme programming," presented at 8th conference on evaluation & assessment in software engineering (EASE 2004), Edinburgh, UK., 2004.