Extreme Programming: First Results from a Controlled Case Study

Pekka Abrahamsson

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

Abstract

Extreme programming (XP) is the most well known agile software development method. Many experience reports have been published in recent years. Successful XP adoptions have however been criticized for the lack of concrete data. While some exist, the studies are often difficult to compare due to different settings and the varying level of XP adoption. This paper reports the first results (concrete data from 2/5 releases) from a controlled extreme programming case study. Four software engineers were acquired to implement a system in a tight delivery schedule of eight weeks. Development environment was close to the agile home ground. A comparison of the collected data from the first two releases is provided. Analysis shows that while the first release is a learning effort for all stakeholders, the second release shows clear improvement in all regards, e.g., estimation accuracy is improved by 26%, productivity was increased by 12 locs/hour and yet the post-release defect rate remained low, i.e., 2.1 defects/KLoc.

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 e.g., 3] have been developed that claim conformance to agile principles explicated in agile manifesto (www.agilemanifesto.org.).

Extreme programming (XP), a method developed by Beck [4], is the most well known of the agile methods. While a number of XP books[1] [5-11] and experience reports [e.g., 12, 13-17] have been published, less is known about the empirical and scientific validity of the method [18-20]. Moreover, XP has been critiqued against for embracing the hacker's culture and thus neglecting the product and process quality viewpoints [e.g., 21].

This paper reports the first results from a controlled case study where XP process and resulting product quality have been under scrutiny. A team of four developers has been acquired to implement a system (code-named for eXpert) for managing the research data obtained over years at a large Finnish research institute. The consortium had already acquired the development of a similar system from an IT provider. That system is promised to be in use late this year (pilot testing is currently ongoing). EXpert, however, aims at providing a subset of this functionality within just eight weeks. 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 since the participating stakeholders were not familiar with the other system. The author of this paper represented customer organization's management to the development team.

The results reported in the paper contain the data obtained from the first two functional system releases delivered after two and four weeks into the eight week project, respectively. The results have been very promising from the customer's point of view. Due to lack of concrete data on XP practices and process, this paper concentrates on analyzing the differences between the concrete data obtained over the two releases. Results shows that while the first release is a learning effort for all stakeholders, the second release shows clear improvement in many regards. The estimation accuracy is improved by 26% and productivity was increased by 12 locs/hour. Yet, the post-release defect rate remained low, i.e., 2.1 defects/KLoc.

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.

[1] The XP series by Addison-Wesley has published already 10+ books on different facets of the Extreme Programming with more to come. The citations here serve as examples.

Proceedings of the 29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03)
1089-6503/03 $17.00 © 2003 IEEE

2. Extreme Programming

Extreme Programming is one of several agile software development methods that have emerged in the past few years. Of all the existing agile methods, XP however is the most well known. XP was first introduced in [22] and it 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 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 [8] 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]. This type of situation is what can be called an ideal surrounding for the XP method or what Boehm [23] 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., 24]. However, the boundaries between different research methodologies and data collection techniques are often overlapping to certain extent [25], which is evidenced in this study. Cunningham [26], 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 [27]. This certainly is the position taken in this study. Moreover, Järvinen [28] follows Oquist [29] 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 [30]. The author was in the role of management in the study and mediated the release post-mortem analysis [31] sessions, which were performed after each software release. In the post-mortem analysis session 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 purposefully. This comes from an observation that the researchers were in a position to design the implementation environment (see next subsection of research setting) beforehand. Using a taxonomy proposed by Baskerville and Wood-Harper [32], the main type of research – case study or action research – methodology employed was that of participatory "action research", in which the investigator participates (or intervenes) in organizational daily work and treats subjects as equal co-workers. The level of author's involvement differed from mere observation (i.e., through obtained data) to actual implementation (i.e., post-mortem analysis session, daily meetings with the project team). The focus is on organizational development and advancing scientific knowledge on the subject matter for academia. In this case, advancing scientific knowledge implies the effort to explore and to understand the process of extreme programming in the context of developing information systems.

3.2. Research setting

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 Finnish research institute. A metaphor that better describes the intended purpose of the system is a large "virtual file cabinet", which holds a large number of organized rich (i.e., annotated) links to physical or web-based resources. The system is an web-based client-server solution.

Item	Description
Language	Java (JRE 1.4.1), JSP (2.0)
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.
Documents	MS Office XP
Web Server	Apache Tomcat (4.1)

Table 1. Technical implementation environment

The four developers were 5-6th year students with 1-4 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, 8]. 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. Table 1 shows the details of the technical environment used for the development of eXpert system. 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.

4. Results

As indicated earlier, due to a lack of empirical data from XP process, this paper concentrate on reporting concrete metrics data obtained from the first two releases. Thus, while qualitative data has been collected on XP practices and the process, they will be reported elsewhere. The concrete metrics involve effort usage for each task and XP practice with a precision of 1 minute, development work size 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. Work commit size was drawn from the CVS tool. The quality of the data obtained was systematically monitored by the project manager, dedicated metrics responsible and the on-site customer.

Collected data	Release 1	Release 2	Total
Total work effort (h)	195	190	385
Task allocated actual hours	136 (70%)	95 (50%)	231 (60%)
# LOCs implemented in a release	1821	2386	4207
Team productivity (loc/hour)	13.39	25.12	18.21
# User stories implemented	5	8	13
# User stories postponed for next release	0	1	1
User story effort (actual, median, h)	10.1	8.3	9.2
User story effort (actual, max, h)	63.1	26.9	63.1
# Tasks defined	10	30	40
Task effort (actual, median, h)	11.7	2.9	4.1
Task effort (actual, max, h)	32.3	8.8	32.3
# post-release defects	4	5	9
Post-release defects/KLoc	2.19	2.10	2.14
# post-release enhancement suggestions	11	12	23
Pair programming (%)	92	73	82
Customer involvement (%)	5	6	5.3

Table 2. Concrete data from two-week releases

Table 2 shows the data obtained from the first two releases. The total column shows the cumulative data from the two releases.

Total work effort dedicated to project work remained constant in the first two releases. However, the direct hours dedicated to tasks was reduced from 70% to 50%. None-task allocated work was the effort spent to planning-game, data collection, project meetings, brainstorming, coaching and not anticipated extra pre-release testing. The team productivity did however improve from 13.39 to 25.12 loc/hour. The first release contained tasks not related to user functionality such as finalizing the technical set up of the development environment. This explains partly the increase in team productivity.

The team implemented five user stories in the first release and eight in the second. The median user story size (hours) for the first release was 10.1 hours and 8.3 for the second. The maximum size of a user story in the first release was 63.1 hours. In the second release, the maximum size was reduced to 26.9 hours. While only 10 tasks were defined for the first release, the second release contained already 30 tasks. Similarly, the median size of a task was reduced from 11.7 to 2.9 hours, and the maximum size of a task was reduced from 32.3 to 8.8 hours.

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. Testers discovered four defects after the first release and five defects after the second release. The defect density for the first release was thus 2.19 defects/KLoc and 2.10 for the second. The defect density was found to be satisfactory giving an indication of the overall product quality. Testers also proposed 11 enhancements (i.e., new or improved functionality) after the first release and 12 following the second release.

Pair programming was extensively practiced in the development of the first release (92%) but was reduced to 70% in the second release. While the customer shared the same office with the development team and thus was present close to 100% of the total time, the actual customer involvement was only 5% in the first release and 6% in the second release.

Figure 1 displays the overall effort distribution for the first two development cycles. Data shows that in this project roughly 9% is required for planning the release contents. Project management activities, which include data analysis, monitoring the progress of the project and the development of project plan require 11% of the total effort. Coding in terms of unit test development, production code, development spikes and refactoring take 54% of the total effort. Daily project meetings took 7% and the overhead caused by data collection was only 2%. All the design documentation including architectural description is displayed in the walls of the development room. The effort spent on an "design" phase is only 2%. This is actually the time used for architecture design. The simple design practice involved in the pair programming coding was not separately tracked. The principal amount of effort was spent directly on tasks, 68% in the first release and 54% on the second release.

User stories and tasks were documented and displayed in the walls as well. The system documentation for the maintenance purposes will be produced in the last two releases when the system architecture and the user functionality has stabilized enough and is not subject to constant changes.

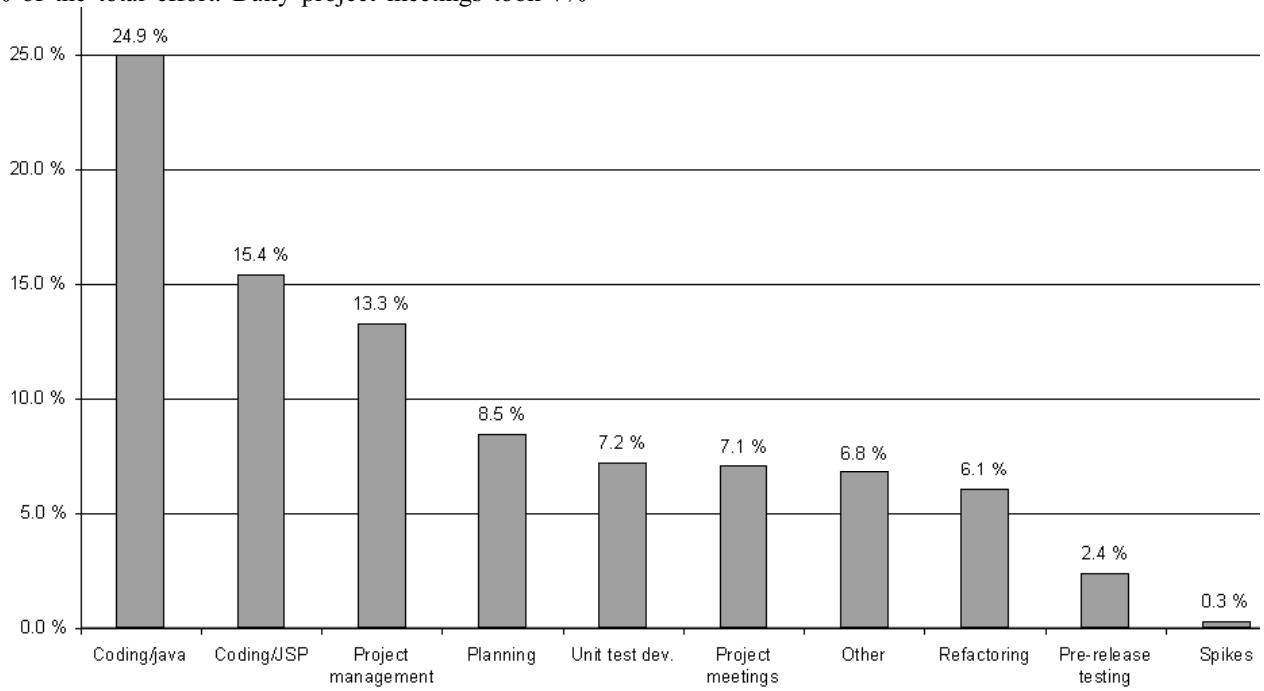

Figure 1. Effort distribution (%)

XP literature has argued that the first release for a novice XP team is a significant learning effort. This argument is supported by the data obtained from this study. Figure 2 shows how the development team learned to decompose the customer defined user stories in smaller segments, which facilitated the project monitoring as well as the story development. Actual effort spent to implementing a task was reduced also significantly. The difference is also clear in the variance indicating better story and task definition.

User story effort estimates are derived from the task estimates. Literature has shown that an ability to estimate accurately is a skill that is learned over time [33]. As suspected, the team had major difficulties in decomposing the contents of the first release into tasks that can be accurately estimated as shown in Figure 3. Estimation accuracy does not appear to improve for the second release but when the actual effort expenditure (in terms of hours, lower graph) caused by a faulty estimate is inspected, the median value (based on 30 tasks) is close to zero hours (i.e., for the second release). This indicates that while the estimates were not accurate (note, tasks were mostly overestimated), the problem was mitigated by having defined considerably smaller tasks. The project manager was able to take necessary actions on daily rather than on weekly basis.

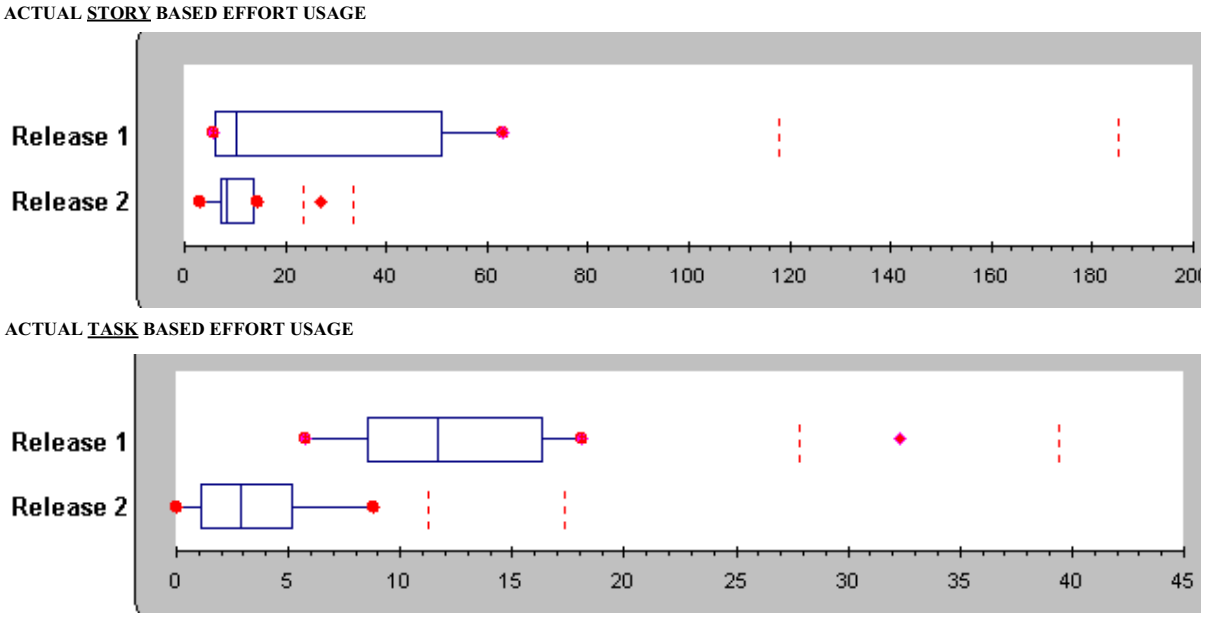

Figure 2. Actual effort spent for user stories and tasks

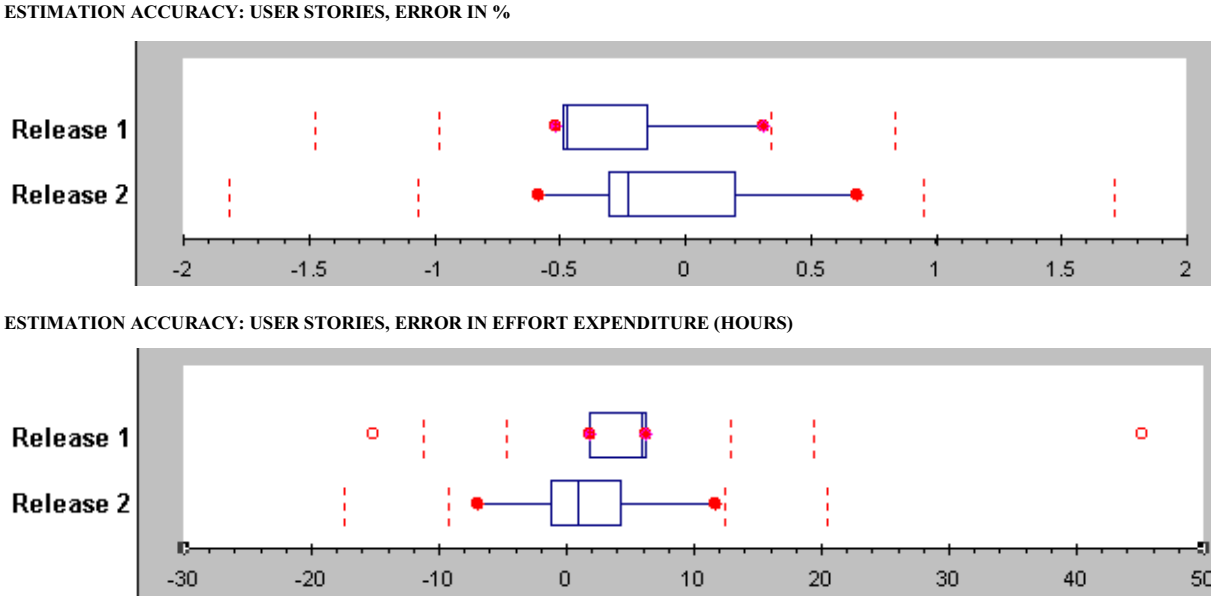

Figure 3. Estimation accuracy

5. Discussion

The results presented in the previous section emphasized the differences between the two system releases in the eXpert project from several viewpoints.

An important finding that can be identified already at this stage of the project is the little need for actual customer involvement in the project. This finding is not in line with the XP literature. Many authors [e.g., 34, 35, 36] maintain that often customer's work effort is very high. In this case, while customer was present close to 100% of the total time, only 5.3% of his work effort was required to assist the development team in the development. However, the mere presence of the customer was highly appreciated by the development team. They 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 noted that in this case study, customer did not develop acceptance tests. He was performing this task manually at the end of each release cycle.

As stated earlier, the post-release defect density was at an acceptable level, i.e. 2.19 and 2.10 defects/Kloc, respectively. 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 defects. Second, the testing team forms 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 [37]. In the two releases altogether 23 new or enhanced user functionality suggestions were received. Also, the eXpert testers were able to observe how the development proceeds from the very first release to the final fully functional system. 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 sign demonstrating customer commitment to the project.

Pair programming is one of the most researched XP practices [see e.g., 38, 39-44]. In the first release, 92% of the programming effort was done in pairs. This was reduced to 73% in the next release. Williams [39] 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-release analysis. However, the fact that the pair programming time remained above 70% in the second release demonstrates that the team felt comfortable with it. More importantly, 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 [45] has consistently shown. The fact that the pair programming was less practiced in the second release and at the same time the productivity was increased is not necessarily related since the first release contained tasks not related to delivering user functionality, e.g., database configurations, software installations.

In software engineering in general, accurate effort estimates are difficult to attain [33]. Initial estimates can often be better regarded as "guesstimates" [46]. Regarding the XP process of producing the estimates, McBreen [21, p. 60] was 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 some concrete results in this regard. The estimation accuracy was improved in terms of estimation error (median error reduced from –48% to –22%) and mis-allocated development time due to inaccurate estimations was also reduced from 5.8h to 0.8h. To support this finding, Langr [47] argued 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 argument. Our findings indicate that a novice XP team is very careful about making too optimistic estimates. However, it took them only two weeks to realize this. The team in this study continued to overestimate in the second release but the accuracy was improved dramatically. Learning to execute the planning game routines was facilitated by a clear agreement on the procedures, roles and responsibilities.

Only one story implementation has been postponed so far. This 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 [48] 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. One of the goals for the project was that no overtime work is needed. So far, this has been achieved.

Finally, and most importantly, the development team has convinced the customer organization that they are able to deliver the system within the time limits defined. The only concern is that the customer organization becomes eager to want more functionality that the development schedule allows.

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 reports the first results from a controlled case study where a team of four developers was acquired to develop a fully operational system for managing the research institute's research data in eight weeks. Due to the tight schedule, the functionality was not fixed. The concrete results reported are based on the first two system releases. 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 post-release defect rate was 2.1 defects/Kloc, which can be seen acceptable for a product yet at production state. The comparison of the results from the first two releases shows furthermore that there is a rather steep learning curve when adopting the XP process for a team that is novice in XP practices. The first release thus is basically about learning while the second release already shows improvement in all regards. For example, the user and task definition skills are improved as well as the estimation skills. For the specific details see section three.

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 is designed to deliver business value for the customer organization. The research institute had thus a dual goal for the project. The issue of data collection was addressed before the project and periodically re-enforced by having the management to ask to see some of the basic data. While the data collection and recording took only about 2% of the total development time, it was found to be an issue producing discomfort. If the client organization does not require detailed data about the development process, the team may easily lag behind in data collection. For this, certain data collection rules must be established within the development team. In our case, the team decided 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.

The data obtained can be used by other organizations working in similar – i.e. close to the agile home ground – environment. Especially, attention should be paid in designing how much effort can be allocated to tasks. This study has shown that at least in the beginning of an XP project, no more than 50-60% of work effort should be allocated directly to tasks. Also, project management appears to require more effort than initially thought.

We will continue to follow up the development process and will report the concrete results from the project as a whole when the system is delivered. The resulting product quality will be inspected from different perspectives including the system code quality. Qualitative data about the process and practices are being collected systematically throughout the development project and all participating stakeholders will be interviewed to evaluate the progress. We believe that the data reported in this paper is of value for practitioners and researchers in the field. More concrete data is needed for the agile movement to progress beyond the practicing enthusiastic.

While software professionals seek a rational basis for making a decision which method they should adopt, the scientific and empirical foundation for such a rationalization is often missing. Fenton [49] reminded that methods introduced continue be based more on faith than on empirical data. There is no quick solution to this problem. As an answer, Fenton suggested that only by gradually contributing to the empirical body of knowledge within the specific area of application are we as researchers and practitioners in a better position to make our decisions on the methods and practices we want to use. This paper has served for this specific purpose.

Acknowledgments

The author would like to thank the development team for their effort in collecting the data, Mr. Juha Koskela at VTT Technical Research Centre for his considerable effort as in the role of an on-site customer for the project, and the three anonymous reviewers for their comments on the early version of the paper.

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] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, *Agile software development methods: Review and Analysis*. Espoo, Finland: Technical Research Centre of Finland, VTT Publications 478, http://www.inf.vtt.fi/pdf/publications/2002/P478.pdf, 2002.

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

[5] W. C. Wake, *Extreme Programming Explored*: Addison-Wesley, 2001.

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

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

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

[9] R. Hightower and N. Lesiecki, *Java Tools for Extreme*

Programming. New York: Wiley Computer Publishing, 2002.
[10] K. Beck and M. Fowler, *Planning extreme programming*. New York: Addison-Wesley, 2001.
[11] L. Crispin and T. House, *Testing extreme programming*. New York: Addison-Wesley, 2003.
[12] 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.
[13] J. Grenning, "Launching XP at a Process-Intensive Company," *IEEE Software*, vol. 18, pp. 3-9, 2001.
[14] P. Schuh, "Recovery, Redemption, and Extreme Programming," *IEEE Software*, vol. 18, pp. 34-41, 2001.
[15] 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.
[16] P. Sommerlad, "Adopting XP," in *Extreme programming examined*, G. Succi and M. Marchesi, Eds. New York: Addison-Wesley, 2001, pp. 423-432.
[17] K. Boutin, "Introducing extreme programming in a research and development laboratory," in *Extreme programming examined*, G. Succi and M. Marchesi, Eds. New York: Addison-Wesley, 2001, pp. 433-448.
[18] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle*, et al.*, "Empirical findings in agile methods," presented at XP/Agile Universe 2002, Chicago, USA, 2002.
[19] G. Melnik, L. Williams, and A. Geras, "Empirical Evaluation of Agile Processes," presented at XP/Agile Universe 2002, Chicago, USA, 2002.
[20] 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, USA, 2003.
[21] P. McBreen, *Questioning extreme programming*. New York: Addison-Wesley, 2001.
[22] K. Beck, "Embracing change with extreme programming," *IEEE Computer*, pp. 70-77, 1999.
[23] B. Boehm, "Get Ready For The Agile Methods, With Care," *Computer*, vol. 35, pp. 64-69, 2002.
[24] R. K. Yin, *Case Study Research Design and Methods*, 2nd ed ed: Sage Publications, 1994.
[25] T. D. Jick, "Mixing qualitative and quantitative methods: Triangulation in action," *Administrative Science Quarterly*, vol. 24, pp. 602-611, 1979.
[26] J. B. Cunningham, "Case study principles for different types of cases," *Quality and quantity*, vol. 31, pp. 401-423, 1997.
[27] D. Avison, F. Lau, M. Myers, and P. A. Nielsen, "Action Research," *Communications of the ACM*, vol. 42, pp. 94-97, 1999.
[28] P. Järvinen, *On research methods*. Tampere: Juvenes-Print, 2001.
[29] P. Oquist, "The epistemology of action research," *Acta Sociologica*, vol. 21, pp. 143-163, 1978.
[30] 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.
[31] 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.
[32] R. L. Baskerville and A. T. Wood-Harper, "Diversity in information systems action research methods," *European Journal of Information Systems*, vol. 7, pp. 90-107, 1998.
[33] W. S. Humphrey, *A discipline for software engineering*. Reading, Mass.: Addison Wesley, 1995.
[34] C. Farell, R. Narang, S. Kapitan, and H. Webber, "Towards an Effective Onsite Customer Practice," presented at XP 2002, Sardinia, Italy, 2002.
[35] 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.
[36] L. A. Griffin, "A Customer Experience: Implementing XP," presented at XP Universe, Raleigh, NC, USA, 2001.
[37] J. Hartwick and H. Barki, "Explaining the Role of User Participation in Information System Use," *Management Science*, vol. 40, pp. 440-465, 1994.
[38] J. Nawrocki and A. Wojciechowski, "Experimental evaluation of pair programming," presented at ESCOM 2001, London, UK, 2001.
[39] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the case for pair programming," *IEEE Software*, vol. 17, pp. 19-25, 2000.
[40] L. Williams and R. Kessler, *Pair programming illuminated*. New York: Addison-Wesley, 2003.
[41] 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.
[42] 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.
[43] S. Heiberg, U. Puus, P. Salumaa, and A. Seeba, "Pair-programming effect on developers productivity," presented at XP 2003, Genoa, Italy, 2003.
[44] K. M. Lui and K. C. C. Chan, "When does a pair outperform two individuals," presented at XP 2003, Genoa, Italy, 2003.
[45] 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
[46] 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.
[47] J. Langr, "Book review: Questioning extreme programming," vol. Page accessed March 10, 2003: XProgramming.com: An Extreme Programming Resource, 2002.
[48] W. S. Humphrey, *Managing the Software Process*. Reading, Mass.: Addison-Wesley, 1989.
[49] N. Fenton, "Viewpoint Article: Conducting and presenting empirical software engineering," *Empirical Software Engineering*, vol. 6, pp. 195-200, 2001.