Empirical Evaluation of Agile Software Development: the Controlled Case Study Approach

Outi Salo and Pekka Abrahamsson

VTT Technical Research Centre of Finland,
P.O. Box 1100, FIN-90571 Oulu, Finland
{Outi.Salo, Pekka.Abrahamsson}@vtt.fi

Abstract. Agile software development, despite its novelty, is an important domain of research within software engineering discipline. Agile proponents have put forward a great deal of anecdotal evidence to support the application of agile methods in various application domains and industry sectors. Scientifically grounded empirical evidence is, however, still very limited. Most scientific research to date has been conducted on focused practices performed in university settings. In order to generate impact on both the scientific and practical software engineering community, new approaches are needed for performing empirically validated agile software development studies. To meet these needs, this paper presents a controlled case study approach, which has been applied in a study of extreme programming methodology performed in close-to-industry settings. The approach considers the generation of both quantitative and qualitative data. Quantitative data is grounded on three data points (time, size, and defect) and qualitative data on developers' research diaries and post-mortem sessions.

1 Introduction

Agile software development has rapidly gained a lot of interest in the field of software engineering. Agile methodologies [see e.g., 1], including extreme programming (XP) [2], emphasize principles such as incremental software development with short iterations, adaptation to changing requirements, close communication, self-organizing teams and simplicity [3].

A fair amount of anecdotal evidence have been published on agile software development methods in the form of descriptive articles, reports, and lessons-learned [e.g., 4-6]. Typically, these sources argue for the effectiveness of agile methods and their practices in specific cases. However, systematic empirical research on agile principals and methods is still mostly missing [7]. An increasing number of empirical evidence on agile methods and their practices (e.g. pair programming and test-first approach) has emerged in the last few years to meet the "urgent need to empirically assess the applicability of these methods, in a structured manner" [7, p.198]. However, there is apparent confusion among researchers concerning experimentation *per se.* Experiments are often confused with case studies and vice versa. For example, experiments seem to lack the collection and analysis of empirical data for their confirmation [8]. The empirical research strategy also seems to be used in a somewhat vague manner in these studies [cf. 9]. In other words, the strategy chosen is not explicated, which undermines the validity of such a study. Yet there is an empirical body of knowledge available. Most of this knowledge consists of surveys [e.g., 7, 10, 11], case studies [e.g., 12-16] and experiments [e.g., 17-22]. These studies provide evidence on a variety of aspects concerning agile software development methods and techniques.

In order to generate impact on both the scientific and the practical software engineering community, new approaches are needed for performing empirically validated agile software development studies. To meet these needs, this paper presents a controlled case study approach, which has been applied in a study concerning XP methodology performed in close-to-industry settings. The approach considers the generation of both quantitative and qualitative data. Quantitative data is grounded on three data points (time, size, and defect) and qualitative data on developers' research diaries, postmortem reviews, and final interviews.

This paper is composed as follows. The following section gives a brief account of the current state of empirical software engineering literature and identifies the main empirical research approaches. This is followed by a presentation of the controlled case study approach and the principal lessons-learned from the application of the approach including empirical evidence. The paper concludes with final remarks.

2 Related literature

Empirical studies conducted in the field of software engineering aim at providing a scientific and thus more rational basis for evaluating, predicting, understanding, controlling and improving the tools, methods and techniques used in software engineering [23]. The influence of assumptions and alternative explanations can be eliminated through empirical research, which also serves for exploring and finding explanations for new phenomena in order to develop and support new theories [24].

Empirical research includes both qualitative and quantitative research approaches. The necessity and the diverse difficulty of empirical software engineering are acknowledged [e.g., 23-25] while it is all too often that, in practice, the decisions are still made without any empirical justification [8, 26]. Empirical studies include various forms of research strategies [25] and can be categorized, for example, into surveys, experimentations and case studies [e.g., 27]. Surveys are used for collecting quantitative or qualitative data from a sample group of population by interviewing or using questionnaires. Whereas case studies [e.g., 28-30] also use both qualitative and quantitative data, experiments are purely quantitative, since their focus is on the behavior of measurable variables. In the following, these research strategies are discussed based on the concepts introduced by Wohlin et al. [31] and Bratthall and Jørgensen [32].

One of the key differences between the research strategies is to be found in the *level of control*. For example, "experiments sample over the variables that are being manipulated, while the case studies sample from the variables representing the typical situation" [31, p. 12]. The experimentation approach can be characterized 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" [25, p. 456]. The control can be further divided into execution control and measurement control. While case studies lack only execution control, the survey approach lacks also measurement control. In addition, the research strategies also vary regarding their *research environment*: while a survey can usually be conducted as a desktop survey, an experiment is usually carried on in a laboratory or university environment. A survey can also be conducted on-line, yet under the control of the researcher. Case studies, on the contrary, are performed in a real-life context. The focal project aims at producing "real" outcomes. The environment in which a case study is performed is, however, uncontrollable. Other factors distinguishing the research strategies from each other are *investigation cost* and *ease of replication*. The investigation cost rises from surveys to case studies and towards large experiments dramatically. Furthermore, surveys and experiments are easier and far cheaper to replicate than case studies.

The case study approach enables an investigation of extensive phenomena, providing data about, e.g., an entire process or a project. However, the unique nature and uncontrollability of variables tend to cause problems in the generalization of results. Any findings can thus be blamed for "unknown confounding factors", and it is difficult to compare the results [33] as is also their interpretation [27]. The multiple case study approach [34] attempts to address these problems to a certain degree. Yet, the costs of this approach are often significantly higher than those of a single case study, the ability to perform replication reliably is not necessarily guaranteed, and the confounding factor problem and interpretation challenges remain as well.

The research perspective of experiment is usually relatively limited. The focus is on viewing the behavior of a specific set of variables in a defined context. As the experiments are usually conducted in a laboratory or classroom settings they have been criticized for being too unrealistic to allow their results to be transferred to industry [e.g. 33, 39]

Action research [35] can be seen as one form of case study [36]. It focuses more on what practitioners do rather than what they say they do. Moreover, action research produces knowledge for guiding practice [37], which is the principal aim of any empirical research. Unlike in the case study approach, in action research the modification of reality requires a possibility to intervene [38]. Yet, action research has similar limitations to those in the case study approach regarding the generalization of research results, and it may be even more costly due to the fact that it requires action taking and monitoring.

The rallying points of the empirical research strategies described above are to be found in their scientific and systematic approach and concern with quantitative data [27], and their strategy of seeking and validating research results through data collection, analysis and interpretation. In order to overcome the inadequacies of the different empirical research strategies various solutions have been proposed. Wohlin *et al.* [27] encourages a simultaneous use of different research methods. An interplay of various research strategies is likely to yield the most benefit [40]. Therefore, using experiments to complement case studies is often suggested [33]. As stated above, conducting multiple case studies is rarely possible because of the high costs and the difficulty of finding similar enough cases [32]. However, multiple data sources should be used in case studies to provide a higher degree of validity [32].

Wohlin *et al.* [27] have proposed some guidelines for deciding between the experiment and the case study approach. Since both approaches are suitable for comparing two software engineering methods, the choice depends on the scale of the evaluation. Wohlin *et al.* [27] argue that a case study suits industrial evaluations particularly well for the reason that it enables avoiding scale-up problems and the study itself is capable of detecting a more widespread and long-term impact. The experiment, again, should be chosen as the research approach if the research is more concerned with studying the reasons for certain phenomena or evaluating the differences between two or more methods.

Table 1 presents the phases (first column) and tasks (second and third columns, respectively) of the different empirical research strategies. In Column 2, the Quality Improvement Paradigm (QIP) [41] approach for conducting empirical studies [27] is described. The steps are drawn from [31, 42]. The last column presents the case study research approach according to Eisenhardt [43].

Table 1. Empirical research approaches presented in literature

Generic phases of empirical research	**Experimental research**	**Case study research**
Design	**Characterize** Current Situation/Baseline setting Topic selection Background research	**Getting started** Definition of research question Possibly a priori constructs Neither theory nor hypothesis
	Set Goals Formulation of goal in a quantifiable manner	**Selecting cases** Specific population Theoretical, not random, sampling
	Choose Process Setting of research context Formulating hypothesis Determining variables Identifying subjects Setting of instrumentation	**Crafting instruments and protocols** Multiple data collection methods Qualitative and quantitative data combined Multiple investigators
Implementation	**Execute** Prepare study Execute study (collect the data) Validate data	**Entering the field** Overlap data collection and analysis Flexible and opportunistic data collection methods
	Analyze Perform statistical analysis Visualize analysis results Study outcomes Accept/reject hypothesis and draw conclusions	**Analyzing data** Within-case analysis Cross-case pattern search using divergent techniques
Learning	**Package** Report findings Store data & analysis for further use	**Shaping hypothesis** Iterative tabulation of evidence for each construct Replication, not sampling, logic across cases Search evidence for "why" behind relationships
		Enfolding literature Comparison with conflicting literature

Generic phases of empirical research	**Experimental research**	**Case study research**
		Comparison with similar literature
		Reaching closure Theoretical saturation when possible

The traditional empirical research approaches (Table 1) serve their purpose in a wide range of research domains. Agile software development is, however, characterized by rapid iterative cycles and continuous changes in the process and in the product requirements. If the empirically validated scientific data about agile software development is to be generated, the research approach needs to be able to adapt to these settings. This creates the need to effectively combine the experimental, case study and action research approaches. This paper proposes a combined approach, i.e. a controlled case study approach, which will be presented in the following section.

3 A Controlled Case Study Approach

In this section the controlled case study approach is presented. It represents a research approach that, for one thing, is particularly suitable for the study of agile methodologies and, for another, generates impact on both scientific and practical software engineering communities. In conjunction with the introduction of the approach, an empirical case is laid out in which the approach has been applied. This facilitates understanding how the approach is designed to work when conducting research on agile software development.

The controlled case study approach was applied in a software development project called eXpert, in which a team of four developers implemented a system for managing the research data obtained over years at a Finnish research institute (section 4). The research goal of the study was two folded. First, the aim was to empirically evaluate the Extreme Programming (XP) method in practical settings. Second, the research aimed at applying the controlled case study approach in order to assess its suitability for studying agile methodologies. The details of the study can be found in [12].

While the controlled case study approach strives for replication (experimentation) and in-depth data collection (case study), it also has the ability to change the process (action research) in a close-to-industry setting in which also business pressure is present. It therefore 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. Furthermore, the *ease of replication* of the controlled case study approach is currently under scrutiny in a connection with a replication of the approach being executed at the moment.

A particularly interesting issue regarding the controlled case study approach is to be found in the dual goal structure of the outcome: 1) fully functional software system or a software product, and 2) research data on selected aspects. Accordingly, the focus of the research is on *evaluating the entire process* of software development when using Extreme Programming or some other agile approaches as the development method. The research considers both *quantitative and qualitative* data and respective data collection techniques.

Figure 1 presents the dynamics, relationships and phases of the proposed research approach designed for the evaluation of agile software development methods in practical settings. It is an iterative research approach incorporating an effective utilization of multiple research methods. These are the experimental and case study research strategies supplemented with the action research [see e.g., 44] perspective.

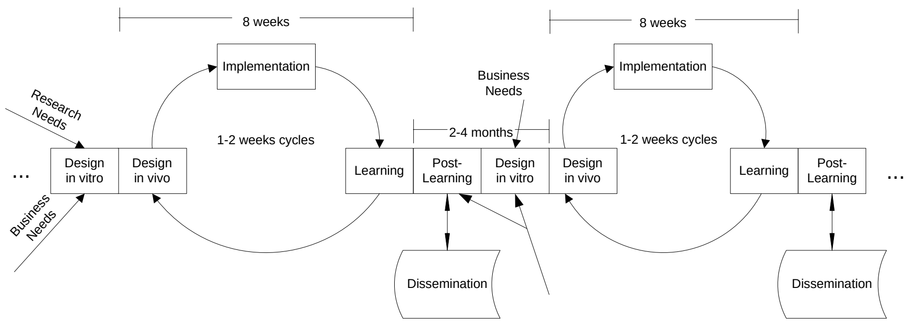

Fig. 1. Controlled case study approach

The phases, steps and outcomes of the empirical research strategy are presented in the following sub-sections. It will also be explained how the different phases and steps were applied in the empirical study (eXpert).

3.1 Design-phase

The design phase is divided into a design *in vitro* and design *in vivo* phases. The design *in vitro* phase occurs prior to the project under research. It initializes the research and enables setting the focus on the most relevant topics in terms of both business and research. It also includes the steps that need to be done only once during the research such as identification of the research needs and selection of subjects, and that does not yet involve the subjects of the study, i.e. the project team in this case. The design *in vitro* is intended to last from one to few months at the most.

The *in vivo* design phase is a fixed part of the software development project under research. It is iteratively applied at the beginning of every 1-2 week development cycle (Fig. 1). These cycles need to be synchronized with the actual iterations of the agile software development process. This ensures the ability to refocus the research, to identify additional measures and variables, and to improve data collection techniques, for instance, during the project.

Table 2 presents the steps and outcomes of the design phase of the controlled case study approach.

Table 2. Design phase steps and outcomes

Design phase	Steps	Outcome
Design *in vitro*	Identification of research and business needs	List of potential topics
	Background research	Topics with the highest scientific and business impact
	Target setting	Goal of current research
	Background research/Baseline setting	Current knowledge on chosen topic
	Setting of research context Multiple investigators	Research environment Researchers and other research parties involved
	Identifying subjects using theoretical (not random) sampling	Target project, project members
Design *in vivo*	Neither theory nor hypothesis: Determining variables Multiple data collection methods: Qualitative and quantitative data	Preliminary variables Updated variables Plan for data collection
	Setting of instrumentation	Data collection tools Project documentation templates Training material/Standards Software development tools

In the *in vitro* design phase the needs of both research and industry are determined to support the eventual topic selection. The needs are further elaborated on the basis of preliminary background research (e.g. desktop research). The eXpert project was preceded by an extensive literature review of agile/XP methods [45], highlighting potential research topics. Once the topic for the current research has been selected, a focused background research may be needed. In the eXpert study this included mapping the empirical research done in the field of agile software development methods. This survey revealed that there was very little empirical evidence available of the applicability of agile/XP processes, though some experiments and case studies had been performed regarding certain practices. Therefore, the goal of the research was set: to evaluate the XP software development process as a whole, and to set a baseline for future replications and more focused research endeavors.

Goal setting was followed by setting the research context. This phase includes selecting the environment for the research project and defining the research parties participating in the process. In addition, the subjects for the research are chosen. In the eXpert project, convenience sampling [31] was used for selecting the nearest and most convenient subjects for the study. In our case, this meant selecting the most experienced of the university students available to be included in the project team. The reason for this was that it was concluded that the validity of the research conducted with experienced students would be comparable with research conducted with practitioners in industry [e.g., 46].

Though no hypotheses are yet set, the variables and methods for their collection need to be defined at the beginning of the *in vivo* design phase (i.e., at the beginning of each iteration). Multiple data collection methods are recommended for multiple sources of information [32]. In eXpert, the applied qualitative data collection methods included group interviews, postmortem reviews and developer's diaries. Quantitative data was grounded on three data points: Time, size and defect as suggested by Humphrey [47]. Time was tracked by minute on XP practices and tasks, size was tracked in terms of lines of code and defects were categorized and recorded systematically. Thus, the generation of both qualitative and quantitative research data was ensured.

Finally, the instrumentation is chosen and prepared, including data collection tools, project documentation templates, standards, training material and guidelines as well as the physical facilities and technical implementation environment for the target project. *In vivo* design requires the ability to identify and develop new data collection mechanisms and to fine-tune the existing ones if needed. In principal, the controlled case study approach calls for redirecting the research in a systematic, controlled and recorded fashion.

3.2 Implementation phase

The implementation phase is very intense taking only eight weeks. The time frame and developer effort usage are therefore fixed. Flexibility is reserved for delivered functionality, which is in accordance with the principles of agile software development. The aim of software development project is to produce a fully functional software system or a software product[1] for an actual customer. The research process itself focuses on collecting qualitative and quantitative research data, which can be used for several purposes. The research data, in the optimum case, benefits the team as well. While the researchers are mainly interested in studying the agile software development process or some specific part of it, the team can (and should) use the data for software process improvement (SPI) [48]. If the team does not find the data useful, the data collection, even if carefully followed, is bound to become error-prone, as it has been the case in personal software process (PSP) research [e.g., 49]

Implementation begins with the finalization of instrumentation (i.e., data collection devices) and training the subjects into their tasks, i.e. the agile development process under investigation, software development tools and data collection procedures. Training should not take more than 2 days. The actual implementation time data collection is performed on a daily basis using the simplest possible mechanisms. In eXpert, paper, pen and a notebook was used. This was supplemented with a set of simple spreadsheets. The validity of quantitative data is continuously monitored during the project by the project manager, metrics responsible, on-site customer, and the management of customer organization. Data is collected not only for storing purposes but it is also actively used via analysis and visualization mechanisms. This enables the analysis to overlap the collection process in an effective and iterative manner.

Table 3 presents the steps and outcomes of the implementation phase in the controlled case study approach.

[1] Due to time constraints the systems or products produced are relatively small in terms of size and effort use, e.g. less than 10000 lines of code or 1000-1500 hours of development effort.

Table 3. Implementation phase steps and outcomes

Implementation phase	**Steps**	**Outcomes**
	Preparing study	Training Infrastructure: Installing of tools, development of templates
	Executing study	Collected research data
	Solving business problem	Working software system
	Validating data	Validated research data
	Analyzing and visualizing the data from current iteration/Overlapping data collection and analysis/Iterative tabulation of evidence for each construct	Analyzed data Visualized data

3.3 Learning Phase

The learning phase is divided into learning and post-learning phases (Fig. 1.). The learning phase includes the steps taken *during* the software development process at the end of every iteration in the Agile process. The post-learning phase takes place only *after* the actual intense software development process, when all the research data is available. Thus, the learning phase is concerned with improving the current software development process as well as the research process whereas the post-learning process aims at systematic dissemination of research data.

Table 4. presents the steps and outcomes of the learning and the post-learning phases of the controlled case study approach.

Table 4. Learning phase steps

Learning phase	**Steps**	**Outcomes**
Learning	Post-mortem reviews/analysis	Suggestions for process enhancement Process enhancements Experiences of the process Feedback for following *in vivo* design phase as e.g. suggestions for data collection improvement
	Interpretation of analyzed and visualized data from the previous iterations	
Post-Learning	Group interview	Developer insights, experiences Generalizations & conclusions => scientific dissemination => industrial application Replications & New controlled case study research projects Enhanced controlled case study process
	Analyzing and visualizing data from entire project (quantitative and qualitative)	
	Store data & analysis for further use	
	Draw conclusions/Theoretical saturation when possible	
	Report findings/Dissemination	
	Identification of future research opportunities	
	Possible involvement of external researchers for data analysis on new perspectives	

The principal mechanism of the learning phase includes postmortem reviews [29], and learning through analyzing and interpreting the collected data. Postmortem reviews are used for enhancing the XP software development process according to the experiences of the related stakeholders in the project. The postmortem review aims at detecting positive and negative issues vis-à-vis the previous iteration. In this phase, the software developers may propose various alterations, which are prioritized and agreed on. In eXpert, the postmortem reviews recurred five times, i.e. after every iteration. The postmortem review was used not only for enhancing the practices of the software development project but for adapting the research mechanisms to the project. For example, data collection tools were improved and variables were advanced during the project.

The post-learning phase includes several steps designed to feed the next project and to analyze collected data for dissemination purposes. For example, a group interview is conducted to survey the experiences of software developers. All data sources, such as post-mortem session recordings, interviews, spreadsheets and developer's diaries are analyzed and the outcomes stored for further use. Further use also includes identification of the most efficient ways of utilizing collected data. It is often only after the project has ended that some of the data use possibilities are detected. Data analysis may also include the involvement of external researchers enabling the investigation of data from their perspective, which is likely to encourage cooperation between different researchers and promote future studies. While this may be regarded as a radical suggestion, it is based on the assumption that case studies of this sort yield more data than a single team of researchers can utilize effectively.

The aim of the post-learning phase is to generate knowledge for scientific and practical use alike. The post-learning phase also enables an explicit consideration of alterations to the research process, which is why this phase lends itself to launching the planning for the following research projects.

4 Applying the Controlled Case Study Approach: Lessons-learned

The controlled case study approach was applied in the eXpert software development project to evaluate its suitability for researching agile software development methodologies. The lessons-learned section identifies the principal issues that can be used for improving the proposed approach.

Design phase

The training material and the implementation plan were the principal issues that were documented prior to the launching of the project. This was seen as an effective means of ensuring sound kick-off for the project, and the documentation was also considered something that could be effectively re-used in subsequent replications. The most important finding of the *in vitro* design phase was the sampling of project members. The validity of research with students is, to a high degree, dependent on their level of knowledge concerning software engineering practices [46]. Recruiting more experienced subjects would thus be worth some effort in advertising and persuasion. The qualitative and quantitative research data from the postmortem reviews and final interviews revealed that detailed instructions, e.g. coding standards, should have been created for the project team prior to the project, i.e. in the design *in vitro* phase.

The *in vivo* design phase proved to be an efficient means of improving the data collection capabilities. The postmortem analysis [50] technique – an improvement and reflection device - which was used for evaluating the project after each software release was found an effective way of identifying issues that needed improvement, not only from the practical business perspective but also from research point of view. The post-mortem analysis is performed in the learning phase and it feeds into the design *in vivo* phase. For example, it was identified that team presence was something that could have an influence on the project outcome. The team presence factor indicated the time spent within project facilities, since no work was to be performed outside. This was realized quite early (after the first release) and appropriate measures were taken, i.e. proper instrumentation, to ensure that also this data point was captured throughout the project.

As an example, postmortem findings on time tracking were found to be directly related with the research data collection. During the project, a total of 14 negative and 4 positive comments were given on time tracking. These comments led to various enhancements made in recurring *in vivo* design phases. For example, improvements were made on data collection instructions to ensure a higher degree of data reliability. The spreadsheet used for collecting working hours was also updated with additional column for tasks not related to actual project work, such as coffee breaks.

The exploratory approach to data collection resulted in a more extensive base for establishing the baselines of agile methodologies. Instead of concentrating on just one practice at a time, e.g. pair programming, data collection covered the whole process and provided data for a broad spectrum of analysis.

Implementation phase

It was found encouraging for the developers to realize that collected data was monitored on a daily basis by several stakeholders. In fact, this proved to underline the importance of data collection and researcher commitment, thus reducing the amount of missing data. Rapid monitoring also revealed various forms of ambiguous data, which could then be immediately revised together with the developers, and either corrected or interpreted more accurately. Based on the experiences gained, the key to success here is the ability to incorporate developers as co-researchers with their own research agenda and interest. In the eXpert case, the developers were responsible for investigating the effect of postmortem review technique within the context of extreme programming.

Yet, it should be emphasized that the principal goal and outcome of the implementation phase is working software. The data should work to yield benefits and not to hinder the progress of the team in practice.

Learning phase

Although it would have been beneficial to use the data also for software process improvement purposes, this was not achieved in the eXpert project. Due to the focus being mainly on the research aspect, process improvement relied on postmortem reviews only. Moreover, the postmortem review results are solely based on related stakeholders’ experiences and opinions. Even though they are valuable as such, they are deemed to fall short without proper tracking mechanisms and therefore lack data for their confirmation. To ensure and improve this aspect of the controlled case study approach, it is suggested that the learning phase is complemented with on-time explicit data interpretation sessions with software developers. These can be embedded in postmortem review sessions. Due to the tight implementation schedule, this may, however, prove difficult.

Regarding postmortem reviews, the bottom-line is the monitoring of whether the suggested and agreed process changes were actually carried out. It is worthwhile to record all the postmortem reviews and also to monitor whether the findings are still valid after the next iteration.

The time reserved for post-learning was a few months, which was found to be too short for effective dissemination. The eXpert-project results are currently being disseminated as the replication study is already in progress. The danger lies in mixing the results of one study with those of another. Thus, explicit emphasis needs to be placed on research discipline. It also needs to be noted that the number of research perspectives one can manage is limited and there will always be data that remains un-analyzed for a long period of time, which is thus in danger of becoming obsolete.

The learning phase, especially postmortem reviews, proved a useful mechanism for improving not only the software development process but also the research process during the project.

5 Conclusions

While agile software development methods have gained wide-spread interest in the field of software engineering, an empirical validation of the ideas presented by agile proponents is very limited. It was claimed that if Agile solutions were to generate impact on both the scientific and practical software engineering community, new approaches to empirically validated agile software development studies would be needed.

To meet these needs, this paper presented a controlled case study approach. The novelty to be found in this approach is twofold: It produces working software and it combines several research strategies aiming at producing valid research data on selected research topics. Both goals need to be treated as equally important or else the close-to-industry setting will not apply. The approach is drawn from experimentation (strives for replication), case study (strives for in-depth nature) and action research (strives for detecting and reacting to changes in the process). The controlled case study approach is explicitly designed to meet the needs of agile software development research: this is done by placing emphasis on the iterative and incremental nature of software development in very short development cycles. The approach considers the generation of both quantitative and qualitative data. Quantitative data is grounded on three data points (time, size, and defect) and qualitative data on developers’ research diaries, postmortem sessions, and final interviews.

The controlled case study approach was applied in order to be validated in an XP case study. The results of the case study are in the process of being disseminated at present and another project code-named zOmbie is currently in progress. Plans are already being made for a third replication. While eXpert and zOmbie have been conducted predominantly with student subjects, future studies have been designed to include representatives of industry developing their own software in a specified research setting.

Although the approach presented in this paper strives to increase the degree of measurement and execution control in close-to-industry settings, it does not overcome the principal limitation of case studies – the inability to generalize results. Pure experiments and traditional case studies are also definitely needed and called for in the area of agile software development. It is, however, claimed by the authors, that it is the interplay of all these approaches that will yield better results in the scientific community for the benefit of software industry. For example, the pair programming technique is currently undergoing a series of empirical investigations in terms of planned experiments in different parts of the world. Using the controlled case study approach we are able to test, verify and invalidate (or validate) parts of the findings made in these studies. This is especially the case when investigating the longer term impact of a particular technique and its interplay with other techniques in the industrial context. It would be more difficult to do this in a purely experimental setting. Furthermore, the in-depth nature of the controlled case study approach enables the identification of procedures, processes, techniques and methods that could be placed under experimental evaluation.

References

[1] B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*: Addison-Wesley, 2003.

[2] K. Beck, *Extreme Programming Explained: Embrace Change*: Addison Wesley Longman, Inc., 2000.

[3] A. Cockburn, *Agile Software Development*. Boston: Addison-Wesley, 2002.

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

[5] J. Rasmusson, "Introducing XP into Greenfield Projects: Lessons Learned," *IEEE Software*, pp. 21-28, 2003.

[6] O. Murru, R. Deias, and G. Mugheddu, "Assessing XP at a European Internet Company," *IEEE Software*, pp. 37-43, 2003.

[7] M. Lindvall, V. R. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams, and M. V. Zelkowitz, "Empirical Findings in Agile Methods," presented at XP/Agile Universe 2002, Chicago, IL, USA, 2002.

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

[9] J. Noll and D. C. Atkinson, "Comparing Extreme Programming to Traditional Development for Student Projects: A Case Study," presented at XP2003, Genova, Italy, 2003.

[10] B. Rumpe, "Quantitative Survey on Extreme Programming Projects," presented at XP2002, Alghero, Sardinia, Italy, 2002.

[11] B. Tessem, "Experiences in Learning XP Practices: A Qualitative Study," presented at XP2003, Genova, Italy, 2003.

[12] P. Abrahamsson, "Extreme Programming: First Results from a Controlled Case Study," presented at 29th Euromicro Conference, Belek-Antalya, Turkey, 2003.

[13] D. Karlström, "Introducing Extreme Programming - An Experience Report," presented at XP 2002, Alghero, Sardinia, Italy, 2002.

[14] F. Maurer and S. Martel, "On the Productivity of Agile Software Practices: An Industrial Case Study,"., 2002.

[15] M. M. Müller and W. F. Tichy, "Case Study: Extreme Programming in a University Environment," presented at 23rd International Conference on Software Engineering, Toronto, 2001.

[16] W. A. Wood and W. L. Kleb, "Exploring XP for Scientific Research," *IEEE Software*, pp. 30-36, 2003.

[17] A. Janes, B. Russo, P. Zuliani, and G. Succi, "An Empirical Analysis of the Discontinuous Use of Pair Programming," presented at XP2003, Genova, Italy, 2003.

[18] S. Heiberg, U. Puus, P. Salumaa, and A. Seeba, "Pair-Programming Effect on Developers Productivity," presented at XP2003, Genova, Italy, 2003.

[19] K. M. Lui and K. C. C. Chan, "When Does a Pair Outperform Two Individuals?," presented at XP2003, Genova, Italy, 2003.

[20] M. M. Müller and O. Hagner, "Experiment about Test-first programming," *Software*, vol. 149, pp. 131-135, 2002.

[21] L. Williams, R. R. Kessler, W. Cunningham, and R. Jeffries, "Strengthening the Case for Pair Programming," *IEEE Software*, vol. 17, pp. 19-25, 2000.

[22] M. Rostaher and M. Hericko, "Tracking Test First Pair Programming - An Experiment," presented at XP/Agile Universe 2002, Chicago, IL, USA, 2002.

[23] V. R. Basili, R. W. Selby, and D. H. Hutchens, "Experimentation in Software Engineering," *IEEE Transactions on Software Engineering*, vol. SE-12, pp. 733-742, 1986.

[24] W. F. Tichy, "Should Computer Scientists Experiment More?," *Computer*, pp. 32-40, 1998.

[25] V. R. Basili and F. Lanuble, "Building Knowledge through Families of Experiments," *IEEE Transactions on Software Engineering*, vol. 25, pp. 456-473, 1999.

[26] N. Fenton, "Viewpoint Article: Conducting and Presenting Empirical Software Engineering," *Empirical Software Engineering*, vol. 6, pp. 195-200, 2001.

[27] C. Wohlin, M. Höst, and K. Henningsson, "Empirical Research Methods in Software Engineering," in *Empirical Methods and Studies in Software Engineering, Lecture Notes in Computer Science*, R. Conradi and A. I. Wang, Eds.: Springer, 2003, pp. 7-23.

[28] A. Birk, Dingsøyr, T., Stålhane, T., "Postmortem: Never Leave a Project without It," *IEEE Software*, vol. 19, pp. 43-45, 2002.

[29] T. Dingsøyr, Hanssen, G. K., "Extending Agile Methods: Postmortem Reviews as Extended Feedback," presented at 4th International Workshop on Learning Software Organizations (LSO'02)), Chicago, Illinois, USA, 2002.

[30] B. Collier, DeMarco, T., Fearey, P., "A defined process for project post mortem review," *IEEE Software*, vol. 13, pp. 65-72, 1996.

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

[32] L. Bratthall and M. Jørgensen, "Can you Trust a Single Data Source Exploratory Software Engineering Case Study?," *Empirical Software Engineering*, vol. 7, pp. 9-26, 2002.

[33] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, and Vokác, "Challenges and Recommendations When Increasing the Realism of Controlled Software Engineering Experiments," in *Empirical Methods and Studies in Software*

Engineering, *Lecture Notes in Computer Science*, R. Conradi and A. I. Wang, Eds.: Springer-Verlaag, 2003, pp. 24-38.

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

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

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

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

[38] 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.

[39] C. Potts, "Software-Engineering Research Revisited," *IEEE Software*, vol. 10, pp. 19-28, 1993.

[40] A. Kaplan, *The conduct of inquiry: Methodology for behavioral science*. New York: Chandler, 1964.

[41] V. R. Basili, "Software Development: A Paradigm for the Future," presented at COMPSAC '89, Orlando, Florida, 1989.

[42] T. Sandelin and M. Vierimaa, "Empirical Studies in ESERNET," in *Empirical Methods and Studies in Software Engineering*, *Lecture Notes in Computer Science*, R. Conradi and A. I. Wang, Eds.: Springer, 2003, pp. 39-54.

[43] K. Eisenhardt, "Building Theories from Case Study Research," *Academy of Management Review*, vol. 14, pp. 532-550, 1989.

[44] C. Eden and C. Huxham, "Action Research for the Study of Organizations," in *Studying Organization: Theory & Method*, S. R. Clegg and S. Hardy, Eds. London: SAGE Publications Ltd, 1999, pp. 272-288.

[45] P. Abrahamsson, O. Salo, J. Ronkainen, and J. Warsta, "Agile Software Development Methods: Review and Analysis," VTT Electronics, Espoo VTT Publications 478, 2002.

[46] M. Höst, B. Regnell, and C. Wohlin, "Using Students as Subjects - A Comparative Study of Students and Professionals in Lead-Time Impact Assessment," *Empirical Software Engineering*, vol. 5, pp. 201-214, 2000.

[47] W. S. Humphrey, *A discipline for software engineering*: Addison Wesley Longman, Inc., 1995.

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

[49] P. M. Johnson, "The personal software process: A cautionary case study," *IEEE Software*, vol. 15, pp. 85-88, 1998.

[50] T. Dingsøyr, Moe, N.B., Nytrø, Ø., "Augmenting Experience Reports with Lightweight Postmortem Reviews," presented at 3rd Int'l Conference on Product Focused Software Improvement (Profes 01), Kaiserslautern, Germany, 2001.