Empir Software Eng (2006) 11: 269–301
DOI 10.1007/s10664-006-6404-6

Investigating the extreme programming system—An empirical study

Panagiotis Sfetsos · Lefteris Angelis · Ioannis Stamelos

Editor: James Miller

Abstract In this paper we discuss our empirical study about the advantages and difficulties 15 Greek software companies experienced applying Extreme Programming (XP) as a holistic system in software development. Based on a generic XP system including feedback influences and using a cause-effect model including social-technical affecting factors, as our research tool, the study statistically evaluates the application of XP practices in the software companies being studied. Data were collected from 30 managers and developers, using the sample survey technique with questionnaires and interviews, in a time period of six months. Practices were analysed individually, using Descriptive Statistics (DS), and as a whole by building up different models using stepwise Discriminant Analysis (DA). The results have shown that companies, facing various problems with common code ownership, on-site customer, 40-hour week and metaphor, prefer to develop their own tailored XP method and way of working-practices that met their requirements. Pair programming and test-driven development were found to be the most significant success factors. Interactions and hidden dependencies for the majority of the practices as well as communication and synergy between skilled personnel were found to be other significant success factors. The contribution of this preliminary research work is to provide some evidence that may assist companies in evaluating whether the XP system as a holistic framework would suit their current situation.

Keywords Agile methods · Extreme programming system · Cause-effect model · Feedback model · Developer perception · Manager perception · Empirical study · Stepwise discriminant analysis · Planning game · Pair programming · Test-driven development · Refactoring · Simple design · Common code ownership · Continuous integration · On-site customer · Short release cycles · 40-hour-week · Coding standards · Metaphor

P. Sfetsos
Department of Information Technology,
Technological Education Institute, 54101 Thessaloniki, Greece

L. Angelis · I. Stamelos (✉)
Department of Informatics,
Aristotle University, 54124 Thessaloniki, Greece
e-mail: stamelos@csd.auth.gr

1. Introduction

Identifying and handling problems such as complexity, conformity, changeability and invisibility is essential in each type of software process improvement (Brooks, 1987). It is known from software engineering theory that the first question a company should ask is: "What are the problems with our current processes?" (Sommerville and Sawyer, 1997). Research studies trying to answer this question are pointing out the importance of understanding the enabling and the inhibiting factors involved in software process improvement, particularly those that managers and developers can control (El-Emam et al., 1999). The processes are dynamic in nature, incorporating both forward paths and feedback loops involving managers, customers and developers individually and in groups. Participants observe, interpret, communicate, decide and act on the basis of their overall perception, instructions, habits and other inclinations, experience and biases (Wernick, 1996). These concepts substantially stress the dynamic behaviour of processes.

Systems thinking is a discipline for seeing wholes rather than tiny things focusing on relations and behaviours in complex systems (Senge, 1990). It was first Beck (2000) and Jeffries et al. (2001) who defined XP as a system. Wiki (2003) XP page also defines XP system as: "a discipline of software development based on four values and twelve practices which balance each other for a successful completion of a project". Further analysis of the state-of-art shows that XP is a system where:

- values and practices are interacting
- common sense practices are applied in extreme levels (Beck, 2000; Paulk, 2001)
- strict rules must be followed (i.e., in test-first, pair programming, etc)
- teamwork is strongly emphasised
- human roles are in the center of the system
- some of the practices take the form of values (i.e., 40-hours week)
- some of the practices give less precise suggestions on how to proceed in the software development (for instance metaphor, coding standards etc)
- the practices are context dependent and their appropriateness depends on a multitude of factors including the team, the organizational environment and the project.

The most important factor pointing out the dynamic behaviour of the XP process is the "people" factor. XP involves managers, customers and developers in a collaborative process, helping them succeed in a project. The ability to successfully implement the XP process varies from company to company and is heavily based on tacit knowledge, skilled and motivated employees and frequent communication. Beck (1999) states, "XP is an intensely social activity, and not everyone can learn it". Companies are not identical and all have to fulfill several requirements for management and employees working in the XP mode: they have to be communicative, social, responsible and skilled. Compositions and interactions between values and practices and their feedback in the XP system led us to consider each software company as a complex system needing a deeper scientific understanding. Reviewing literature concerning the XP we found that there were few empirical studies in this research field. Therefore, there was a strong need for obtaining empirical data concerning the adaptation and integration of the XP practices. We had to

investigate whether different sized companies producing different software products report different kinds of problems. The most pressing research question we had to ask early was "Which are the success factors, difficulties and uncertainties—affecting or caused by—the implementation of the XP practices?" Many other research questions rose from this fundamental question such as:

- What are developers and managers doing when practices do not meet the needs of their projects? Do developers and managers improvise combining their own versions of some practices, or modify existing practices to better suit their objectives?
- Do developers perceive and apply differently some practices than managers do?
- Do larger companies apply the XP practices the same way the small companies do?

To answer these research questions we specified a generic XP system, incorporating the holistic approach of XP. Based on this holistic approach we developed a research tool, a simple cause-effect model including factors affecting the XP process. This tool helped us analyse the XP practices in the light of different dimensions, such as feedback, clarifying at the same time the XP system resources needed. In the first part of the study various factor classes, in form of open lists, were proposed to developers and managers. Managers and developers expressed their opinion and added new factors. Based on these findings, questions regarding the factors were formulated and the second part of the empirical study was conducted, using the same subjects. In total 30 managers and developers from fifteen companies, sized from small to large, and using XP as their main development process, in Northern Greece, were exhaustively investigated. Data were gathered using the sample survey technique with questionnaires and interviews, in both parts of the investigation. In order to perform such a complicated analysis we had to analyse each practice separately and all practices simultaneously building up prediction models. This statistical procedure was followed in order to find (a) whether there are differences between small and large companies and (b) if there are differences, which are the variables–factor questions that can distinguish the two participating groups (managers and developers). The remainder of this paper is organised as follows. In Section 2 we present empirical work related XP research. In Section 3 we analyse the structural blocks of the XP Systems thinking while in Section 4 we analyse feedback in the XP process. We present the structure of the XP cause-effect model in Section 5. We describe research setting in Section 6, the methodology in Section 7. In Section 8 we present the analysis of data and validation–reliability discussion. We present our results in Section 9. Finally, we draw our conclusions and summarize our findings in Section 10.

2. Related Work

Organizations need evidence that XP practices work in their environment before they promote and deploy them on a larger scale. This need becomes greater in large organizations because of their complexity and the need to integrate new processes with existing ones. However, empirical research studies focusing on the assessment

of Extreme Programming from a holistic perspective are few. In a recent study (Lindvall et al., 2004) researchers based on the experiences of several large organizations concluded that "agile practices match the needs of large organizations, but integrating new practices with existing processes and quality systems that govern the conduct of software development requires further tailoring." Aveling (2004) surveyed four software companies to investigate if XP practices were adopted in full. Results have shown that XP is often used as a tool kit of practices, all of which offer value, some of them depend on other practices and some of them are politically difficult to adopt. To the same purpose, El-Emam (2003) surveyed project managers, chief executive officers, developers, and vice-presidents of engineering for 21 software projects. Results have shown that none of the companies adopted agile practices in a "pure" form. Project teams adopted selectively some practices and tailored some others to operate in their working environment. A survey to determine whether agile methods/XP reduce costs and improve development time was performed by Reifer (2002). The results, collected from 14 firms and 31 internal small projects (pilot studies) using XP practices, have shown that these projects had average or better than average budget performance and schedule adherence. Projects in the software and telecommunications industry reported high product quality. E-business reported above quality ratings while aerospace industry reported a below quality rating. Rumpe and Schröder (2002) performed a survey, asking companies running XP about the practices they follow and also the future plans and personal background. They presented a statistical description of their data and concluded that it was early to come up with final conclusions (more empirical studies were needed).

Three case studies performed by the same research group, provide empirical evidence of the efficacy of XP practices and their implementation in software organizations. The first case study was performed with an industrial team that had plan-driven risk factors at Sabre Airline Solutions to evaluate the effectiveness of XP practices (Layman et al., 2004a). Study compared the team's business-related results (productivity and quality) to two published sources of industry averages and found that the team yielded above-average post-release quality and average to above-average productivity. A second similar case study was performed with an agile team, this time at the same organization (Layman et al., 2004b). The study compared the business-related results of a product developed by the team using traditional methods and the same product further developed using XP. Results showed a 65% reduction in the product's pre-release defect rates, a 35% reduction in the post-release defect rates, and a 50% improvement in productivity in the XP release. The third case study was performed with a small team (7–11 team members) at IBM (Williams et al., 2004). Two software releases helped this team to transition and stabilize its use of a subset of XP practices. Findings of the study were that the team improved productivity, reduced pre-release defect density by 50%, and achieved a 40% reduction in the post-release defect density when compared to the same metrics from an earlier release.

There are some empirical studies which focus on the economic evaluation of the XP projects. Two studies about the economic feasibility of XP from a project economics point of view were presented by Müller and Padberg. In the first study (Müller and Padberg, 2002), the authors used the concept of net present value to study for which project settings a lightweight programming paradigm such as XP can be an advantage over a conventional development process. Results have shown that

XP is the right paradigm to choose if the project is small to medium scale, the product is of high quality, and time to market is the decisive factor. In the second study (Müller and Padberg, 2003), the authors developed a novel economic project model which allows to study the impact of the key XP practices—Pair Programming and Test-Driven Development—on the business value of a project. Results have shown that the economic value of XP strongly depends on how large the XP speed and defect advantage really are. Results have also shown that the market pressure is an important factor when assessing the business value of XP. Williams and Erdogmus (2002) present another study about the economic feasibility of pair programming which is also based on the concept of net present value. Study concluded that that there is an overall economic advantage of 40 percent for pairs over single programmers. Maurer and Martel (2002) reported a case study of a nine programmer web application project. The team showed strong productivity gains after switching from a document-centric development process to XP.

Many other empirical studies focus on isolated practices. Pair programming and test-driven development have been investigated to a greater extend. Empirical studies on pair programming have shown that pair speed and defect advantage (reduced defect density) are the potential benefits of Pair Programming (Cockburn and Williams, 2000; Nosek, 1998; Williams et al., 2000, 2003a). Padberg and Müller (2003) developed an economic model based on the standard concept of the net present value for analyzing the cost and benefit of pair programming. They found that when the market pressure is strong, then pair programming speeds up the project balancing the increased personnel cost. Erdogmus and Williams (2003) present a comparative economic study that strengthens the case for pair programming. Hulkko and Abrahamsson (2005) studied in four projects the impact of pair programming on product quality. The results indicated that pair programming does not provide as extensive quality benefits as suggested in literature, and on the other hand, does not result in consistently superior productivity when compared to solo programming.

Some empirical studies were devised to investigate the distinction between test-driven development and traditional test-last development from the perspective of developer productivity and software quality. Müller and Hagner (2002) describe an experiment that compared test-first programming to traditional test-last programming with solo programmers. Results have shown that there is little or no advantage to test-first programming in either productivity or reliability. A similar experiment is described by Geras et al. (2004). Results have shown that there is little or no difference in developer productivity in the two processes, but there are differences in the frequency of unplanned test failures (advantage in debugging time). Erdogmus (2005) in a study on TDD found that test-first students on average wrote more tests and, in turn, students who wrote more tests tended to be more productive. It was also observed that the minimum quality increased linearly with the number of programmer tests, independent of the development strategy employed. In another study (Erdogmus and Wang, 2004) on TDD, present a process measurement approach for TDD that relies on the analysis of fine-grained data collected during coding activities. George and Williams (2003) also studied test-driven development but not in isolation of other agile practices. Results have shown an advantage of the agile process by a significant margin, however, there is little evidence to support saying that this difference was due to the test-first practice alone. It seems more likely that the synergy of all the agile practices yielded the

positive result. Williams et al. (2003b) reported in a case study of industrial programmers at IBM, that creating functional tests immediately after design and then using them to guide the construction effort, resulted in 40% fewer defects during functional verification. Finally, Melnik and Maurer (2005) conducted a study on using agile methods in software engineering education. This study explores the perceptions of students from five different academic levels of agile practices. Student opinions indicated the preference to continue to use agile practices at the workplace if allowed.

3. The Structural Blocks of the XP Systems Thinking

As mentioned in the introduction, XP discipline leans to systems thinking, when we are considering its implementation as a holistic framework applied in software companies. It is known that systems thinking cause a shift of mind (Senge, 1990):

- seeing the structure that underline complex situations
- seeing interrelationships rather than linear cause-effect chains and
- seeing processes of change rather than snapshots

For the XP discipline, this consideration could perfectly identify and express the underlying body of practices and values, explaining also the successes and failures of the XP system. System means a grouping of parts that operate together for a common purpose or more precisely a plethora of variables which are organized in a circle or loop of cause-effect relationships, known as a feedback process (Kauffman, 1980). To understand the structure of the XP system is to understand how the involved parts are related. Grasping the structure of the XP system helps understand it in a way that permits to enhance it with additional factors. Every 'system' (e.g., see Hirschheim et al., 1995) must be seen as a collection of people, processes, data, models, technology and some partially formal language, that together form a coherent structure which serves some organisational purpose or function. In this notion of a system, even though there are many differences in the implementation processes, human roles are in the center of the system. XP practices are more people-oriented than process-oriented (Fowler, 2001) and are heavily based on tacit knowledge, skilled and motivated employees and frequent communication. Moreover, XP practices operate at different levels: team, project, organisation, short and long term, technical and global processes. It is obvious that XP system sets several requirements for management and employees working in an XP environment: they have to be communicative, social, responsible and skilled. Some of the practices, such as code review (see pair programming), refactoring, simple design, coding standards, are well-known from other processes, but applied more intensely and more frequently than in strict and predefined processes. Thus, more human and cultural effort is necessary for the implementation of these practices.

Many other social and technical factors derived from the nature of some of the XP practices must included into the system definition. Practices such as metaphor, coding standards, on site customer and 40-hour week demand some extra social, technical or economical effort. Moreover, the development of the software is done iteratively and phases sometimes overlap, so other factors affecting the implementation of the practices may enter dynamically into the system. Changes in resources,

people, tools and technology, are inevitable and must also be taken into consideration. All those factors affecting the implementation of the XP system must be included in the XP system definition either as pre-conditions or post-conditions, or as input variables. In the next two sections some concepts and principles underlying our XP System model will be proposed and analysed. System variables and other definitions will be provided in subsequent sections.

4. Feedback in the XP Process

The concept of feedback system, as we mentioned above, will be the basis for structuring the XP systems thinking. Software processes, are complex multi-loop, multi-agent, multi-level feedback systems (Lehman, 1996). A feedback system has a closed loop structure that brings results from past action of the system back to control future action. XP is a software evolution process involving the values and practices that induce interaction and feedback in the system (Fig. 1).

The implementation of the XP practices involving managers, customers and developers, strengthen the properties of the XP system. Practitioners communicate, interpret, decide and act on the basis of their skills and experience affecting the feedback control, which in some situations arises as indirect, unplanned, or even unconscious result. The most significant properties of a feedback system are negative and positive feedback influences. Negative feedback in the XP process (called *negative factors* in our study, see next section) hinders the achievement of the goal of the system, while positive feedback (called *positive factors* in our study, see also next section) generates process growth, wherein action builds a result that generates still greater action. The composite effect of positive and negative feedback is a complex function involving path characteristics, gains and delays affecting the global system behaviour. Thus, a model incorporating feedback influences should be more suitable as a proper evaluating tool towards mastering XP system construction and behaviour.

5. The Structure of the Cause-Effect Model

From theory it is known that a model is an abstract representation of the system to be built from a certain viewpoint, serving as a communication mechanism and using to answer questions about the system. The cause-effect model, we constructed, covers the main structural and behavioural aspects of the XP system, illustrating also the feedback that governs its behaviour. After the quotation of our research questions we constructed the model based on the XP literature (Butler et al., 2001;

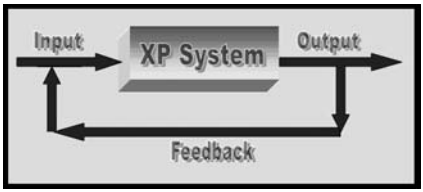


Fig. 1 Graphical presentation of the XP feedback system

Gittins et al., 2001; Rumpe, 2002) and on similar empirical studies. The model built and used, as a research tool, is a first attempt to describe:

- the important features of the system in terms of values, practices,
- other factors affecting the application of the XP, and
- the feedback interactions that these features introduce in the system.

In addition to the four values and twelve practices already mentioned, we added a new aggregated variable-type that we called "*factors*" in the preliminary XP cause-effect model. As shown in Fig. 2, factors include skills (e.g., Efficiency in programming, Work in team) and interactions or hidden dependencies between practices, as well as other pre-conditions (in the form of 'needed factors,' e.g., Customer collaboration, Partnership relations) and post-conditions (in form of reflective positive results, e.g., Tacit knowledge transfer, Quick software development).

Interactions and dependencies include the twelve practices. All the other affecting factors were proposed by the authors of this paper, based on literature review, and completed by the participants in the first part of the empirical study. In the proposed model, the factor variables can take values of type "good–bad (G/B)," "yes–no (Y/N)" and "increase–decrease (I/D)." Here, we must clarify that in this preliminary study it is impossible to include all affecting factors. Most of the factors are tacit, hidden and conceived differently by development teams in different companies. Factor specification is a difficult process, and the purpose of this pilot study is to identify which of them are the most significant for the successful implementation of the practices and the feedback loops they introduce into the system. The most significant factor variables are those involved as pre-conditions and post-conditions. Pre-conditions are factors needed for the implementation of the

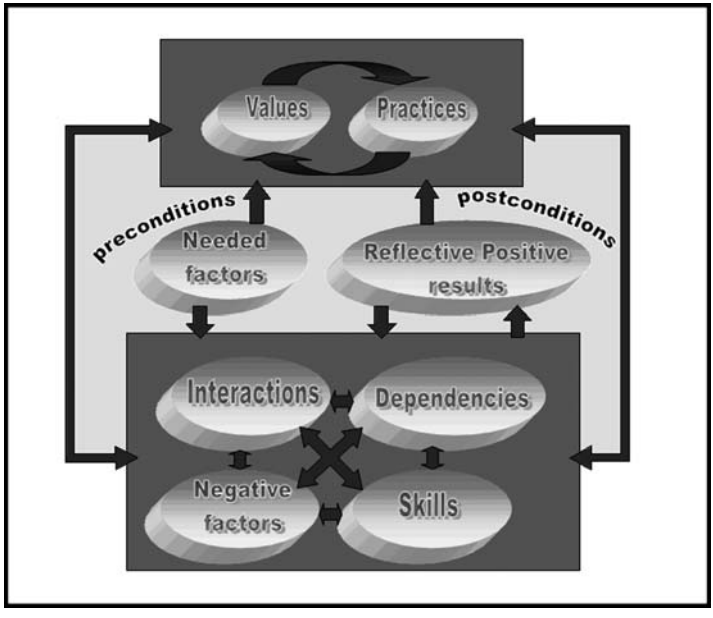


Fig. 2 The structure of the XP cause-effect model

practices, while post-conditions are the reflective positive results causing feedback loops in the system. Interactions and dependencies represent the relations among the twelve practices. Skills can also be characterised as needed factors but we preferred a more generic solution, defining them as a separate factor class. Negative factors are affecting negatively the system and restrict or hinder the implementation of the practices (for a detailed description of the factor variables, see Appendix A).

6. Research Setting

The research study has been carried out between September 2002 and February 2003, investigating fifteen software companies that apply the XP–practices in Northern Greece. The companies are listed in Table 1, and referred by a code name for confidentiality reasons. The selected companies, sized from 5 up to 200 staff members, are an active part of the software production industry. Five companies have been considered as 'large' for the purposes of this study. The term 'large' reflects the potentiality of these companies to apply XP, which is best suited to small development groups. These companies:

- had well-established structures and processes,
- used mixed methodologies,
- could contracted large projects,
- used distributed teams, outsourcing and scale-up strategies (teams with up to 20 developers, team of teams, more developers from other production lines, etc.) (Cockburn, 2001).

Companies are characterized in terms of their developed products. The products are classified according to application domain and type. Fifteen developers and fifteen managers (a manager and a developer from each company) were

Table 1 Participating organizations

Company	Products	Customer	Business model	Staff	Company Age
Company 1	Support systems	Internal	C*	120	>20
Company 2	Business systems	Business	M*/C	5	10
Company 3	Support systems	Internal	C	50	>20
Company 4	Business systems	Business	M/C	5	5
Company 5	Internet/communication	Business	M/C	40	10
Company 6	Business systems	Business	M/C	200	>30
Company 7	Image processing	Business	M	6	8
Company 8	Business systems	Business	M/C	8	7
Company 9	Business systems	Business	M/C	10	5
Company 10	Business systems	Business	M/C	200	>20
Company 11	Business systems	Business	M/C	10	9
Company 12	Business systems	Business	M/C	5	20
Company 13	Multimedia	Business	M/C	10	8
Company 14	Internet/communication	Business	M/C	5	4
Company 15	Business systems	Business	M/C	9	10

* C=Contract; M=Market.

thoroughly investigated using the sample survey technique with questionnaires and interviews.

7. Methodology

Qualitative research methods were used to capture data, in two separate stages, allowing both qualitative and quantitative type of analysis. We used *semi-structured* interviews (Taylor and Bogdan, 1984; Patton, 1990) (see Appendix B) and questionnaires (see Appendix C), including a mixture of open-ended and specific questions, designed to elicit not only expected information, but also unknown pieces of information (Seaman, 1999). In interviews we used *open-ended questions* allowing individual variations, informal conversation and verbatim transcripts (Patton, 1990). The questions for both interviews and questionnaires were developed to elicit the issues specified in our research questions with the help of the XP theory and the review of related empirical studies.

Questions targeted the software development process, XP practices, and the effectiveness of the participants. Interviews revealed a lot of information about the factors affecting the implementation of the XP practices as well as information about how the practices were being followed by developers and managers. Qualitative data were coded and analysed using the Glaser-Strauss' *constant comparison method* to elicit unwanted trends and biases (Glaser and Strauss, 1967). The interview cycle was conducted as follows:

1. The researchers presented a predefined factor list (see filled lists in Appendix A), helping participants to answer the questions.
2. Participants were permitted to deviate from the given question in order to express their complete opinions proposing also new factors.
3. The researchers checked that questions were answered by participants. Interviews were transcribed verbatim and field-memos were produced with their comments and suggestions.
4. The researchers summarized the meeting and suggestions from the participants (if any) filled-in the list and handled it back to the participants.
5. The summarized report of the meeting was proofread by the participants.

The researchers collected the reports, and compiled a comprehensive questionnaire and completed the factor list, based on the contents of the reports. During the second stage of the study, interviews were conducted again to propose to the interviewees the factors they considered essential for the implementation of each practice. The completed questionnaires provided the data for our study. Before the analysis of the data, the reviewers examined both the process and data for consistency.

8. Analysis of the Data

The analysis of the data was completed in two separate statistical procedures. In the first procedure, data selected for each practice were analysed separately, using **Descriptive Statistics (DS)**. DS application includes counts, percentages and various graphic displays (see Results and Appendix D). In the second procedure, data

selected in the second part of the study were analysed using the **stepwise Discriminant Analysis (DA)**. DA was used for the 15 responses for each practice (separately for managers and developers) in order to find (a) whether there are differences between the two company groups (large and small), and (b) in case there are differences, which of the variables–questions can distinguish the two groups. The stepwise DA builds each time a model of group membership based on the answers to the questions of the interviewees. The Wilk's lambda was used as the criterion for entering or removing variables in the model. All the dichotomous (yes–no) variables were considered as candidates for the model. However, the procedure resulted each time in a model with only few variables (answers), which can be considered as the most useful for discriminating among groups. A discriminant function (DF) was generated for each model, based on a linear combination of the variables from the results of the study where the membership was known in advance. It is important to keep in mind that the DA considers all the variables simultaneously and not individually. So, it is a statistical procedure used to describe and interpret the differences between the two groups in a much more positive approach than a test for a null hypothesis (Krzanowski, 1993). The performance of DF, can be judged by the percentage of correct classifications of the companies or the developers respectively in the sample. However, this percentage is optimistic since the classified cases are the same ones used to build the model. In this regard, the cross-validation method is used to decrease the bias. Each case is removed from the data and a classification model is computed from the rest. Then the removed case is classified according to the model. The percentage of the correct classifications by this method is reported too. The analysis of data was made with the SPSS statistical tool.

8.1. Validity and Reliability

The need to use qualitative methods in our study is firstly to focus on multiple and dynamic realities and secondly on people's behaviour and skills. It is known that '*qualitative methods don't answer directly to research questions but can be used either to better understand any phenomenon, or to gain new perspectives on things already known, or to gain more in-depth information that may be difficult to convey quantitatively*' (Strauss and Corbin, 1990). In quantitative methods the concept of validity of the conclusions is highly dependent on how well specific threats have been manipulated (Basili et al., 1986; Wohlin et al., 2000). In qualitative methods the respective concept of trustworthiness must be examined (Lincoln and Guba, 1985). Trustworthiness includes four criteria which must be analysed:

Credibility, as an internal validity in quantitative methods, is the accuracy with which the researcher has represented the views of the subjects in their conclusions. Otherwise, the question here is: How confident can we be about the "truth" of the findings? In this study, the responders did not provide factual statements but strong perceptions, based on real experience, i.e., XP projects they had managed or developed within their company. To answer the above questions and increase the probability that credible findings will be produced we included the following activities:

Prolonged engagement (Lincoln and Guba, 1985, p. 301). We spent sufficient time to build trust with the interviewees and to illuminate cultural aspects of

the investigated companies. Besides, the open ended-questions in interviews helped us avoid misinformation in the factor-formulation activity.
Triangulation (Lincoln and Guba, 1985, p. 305). We used multiple researchers and sources (interviews and questionnaires). Questions were both direct concerning the factors affecting the system and indirect concerning their experience on the XP process and the implementation of the practices. Triangulation also occurred by comparing the managers opinions and the developers opinions.
Peer debriefing (Lincoln and Guba, 1985, p. 308). The interview cycle was analytically explained in Section 7. Different researchers were used to increase the control on the inquiring process. After the analysis cycles the results were reported back to the subjects of the study.
Factor refinement. A preliminary list of factors was proposed by us, but it was modified and completed by the participants in the first stage of the study.
Member checks (Lincoln and Guba, 1985, p. 314). The interviewees were asked to corroborate both the factors and the findings and clarify their comments (see Section 7).

Transferability, as an external validity in quantitative methods, is the applicability of research results to similar settings. The key question here is: Can we apply these findings to other contexts or with other groups of personnel? Transferability of findings in qualitative studies is limited. However, some analytical generalizations can be drawn (Lincoln and Guba, 1985, p. 316). To allow others generalize or transfer findings to similar contexts and settings we thoroughly describe the research context and the assumptions that are central to the study, providing all data concerning companies and personnel, the methods used and the analytical results and conclusions (available also as a file in a web page- http://sweng.csd.auth.gr/agile_pubs.html -download). We tried to increase variation in the study's parameters, choosing companies with different domain and context, and producing different products. We also varied the size and age of companies to cover a broad part of software industry using XP in their development process. However, we acknowledge that the number of the surveyed companies could be considered as a threat to transferability. This may have an impact on the generalizability of the study.

Dependability, as a reliability in quantitative method, refers to the quantitative view of reliability and is based on the assumption of replicability or repeatability. The key question here is: Would the findings be repeated if the study were replicated with the same or similar subjects in the same or similar context? In a quantitative study, repeated administration of measures provides the same results while in qualitative studies, repeated interviews and other forms of data gathering emphasize the variability of human experience, which is carefully tracked by the investigator. Trackable variability provides an adequate level of dependability of findings. The main method to ensure trackable variability is a dependability audit (internaly done—see Confirmability below), where reviewers examine both the process and data of the research for consistency (Lincoln and Guba, 1985, p. 317). Other methods include an analytic description of research methods (documentation), triangulation, and code-recode procedures. Researcher's view on XP system, the research questions and the cause-effect model used are documented. Data collection procedure has been reported in Section 7. An important safe-

guard against the data being interpreted under subjective bias was that the factors, used in the cause-effect model, were identified by their repetitive nature in interviews. Data analysis using two different Statistical methods (stepwise DA and DS) is also reported (in Section 8). The results, enhanced with the aspects of the participants in interviews, are presented.

Confirmability expresses the objectivity in quantitative methods, and it is the process of checking the researcher's interpretations and conclusions for plausibility. The key question here is: Is the study designed to show what it is intended to show? Confirmability, like dependability, is primarily achieved through the use of audit trails. In an inquiry audit, the auditor examines both the dependability of the process and the confirmability of the product (Lincoln and Guba, 1985, p. 316–318). An internal audit by a researcher familiar with qualitative research (second author of the study) began at the onset of the research, and continued through data gathering and analysis. The items thoroughly examined were raw data, data analysis information (working theories, reduced data), data reconstruction (factors), synthesis information (theories, inferences), process notes, personal notes, and preliminary developmental information (Lincoln and Guba, 1985, p. 320–321). Researchers view was clearly presented in Sections 3–5. Factors were identified by their repetitive nature in interviews providing safeguards against researcher's pro-inclusion stance and biases. Trying to reach the '*area of neutrality*,' stated by Lincoln and Guba (1985), researchers remained flexible and open to new possibilities throughout the process, striving to report what is found in a balanced way.

In summary, this pilot study is performed in industry, according to a structured method and reported as openly as confidentiality issues allow. Limitations concern the number of investigated companies and the sample of affecting factors. Although we selected companies with different domain and context, the number of surveyed companies could be considered questionable and may threaten transferability of our findings. The small number of interviewees may also affect the quality of the results. Furthermore, we acknowledge that the identification of all factors affecting the XP system is a difficult process. Therefore, we can not assert that the specific problems are relevant to all companies wanting to improve their XP software processes, although most XP organizations have similar needs and face such problems.

9. Results

Results for each practice contain:

- **DA results** from the **company's view** (answers from managers) and **developer's view** including:

  the Fisher's *linear discriminant function variables* (most important).
  the classification correctness of the companies (both with and without cross-validation).

- A thorough annotation of the results, including elements of the **qualitative analysis,** enables us generalise and classify the **enabling** and **limiting factors** for large and small companies (Tables 2 and 3 respectively).

Table 2 Enabling and limiting factors for large companies

Practice	Enabling factors	Limiting factors
Planning game	Divide and conquer (task breakdown, upfront architecture, use cases, etc.) for large and complex projects More formal communication Rotation of skilled developers among teams	Company's organizational issues (mixed and well defined processes, distributed work across multiple teams, communication and coordination problems)
Pair-programming	Creation of a collaborative and supportive environment with collocated teams Continuous rotation of developer pairs	Distributed work access across multiple teams Cultural problems (traditional-agile teams)
Test-first	Communication and collaboration (developers-customer)	Lack of automated testing tools Skilled developers-customers
Refactoring	More up front design and design patterns	Lack of refactoring tools Collision with other quality control systems
Simple design	Unit-testing	Lack of theory-guidance Lack of documentation in distributed development Time restrictions in development process The shift of priorities in development process
Code ownership	Team-oriented practices	Volume and type of projects Cultural problems (traditional-agile teams)
Continuous integration	Team-oriented practices Object oriented analysis	Cultural problems (traditional-agile teams) Type of projects (WEB, internet, etc.) Difficulties at the beginning of the project
On-site customer	Surrogate customer (when needed) More planning approaches (design flexibility) More team practices	Type of projects (WEB, internet, etc.) Lack of capable and knowledgeable customers
Short release cycles	More upfront architecture GUI and interfaces architect Divide and conquer (valuable user stories first, valuable task breakdown, etc.)	Large and complex projects Distributed work across multiple teams
40-hour week	Effective organization and strict implementation of both working tasks (iteration time, release time, etc.) and team meeting schedules (fixed time meetings, etc.) Change project's scope or schedule	Company's organizational issues Type | Volume of projects

Table 2 (Continued)

Practice	Enabling factors	Limiting factors
Coding standards	Team-oriented practices Object oriented practices	Company's organizational issues
Metaphor	Enhanced metaphors (high level designs, UML–diagrams, etc.)	Cultural problems (traditional-agile teams)
	More architectural design	Unclear practice theory Limited experience

- A **classification** of the practice as *technical*-and/or *social* oriented as defined by managers and developers in interviews. The term technical is used here to mean the technical aspects of software development (analysis, design, implementation, and test-first/automated testing). The term social refers mainly to XP values including team capabilities, job satisfaction and human comfort. Technical orientation has to do with the outcome of the development process (the software product). Social orientation has to do with the social factors affecting the development process.
- **The DS results** complement the judgment of DA findings and have two parts. Firstly, the degrees of practice applicability (Fully use + Partially + Not at all = 100%, i.e., 15/15) and secondly, the degrees of practice evaluation (Helpful as is + Improvable + Hard to apply as is = 100%, i.e., 15/15). "Fully use" means that practice was applied in all projects, while "Partially use" means that practice was applied in some projects.

9.1. Planning Game

DA results have shown that the factors discriminating large from small companies, concerning this practice, are the environmental differences and differences in the development processes (company's organizational issues) that exist between the categories of companies. Large companies, applying mixed and well defined development processes (fair amount of planning and documentation, systems for assuring quality, etc), meet problems in some large or complex projects. Distributed work across multiple teams and cultural differences between agile and traditional teams hinder communication and coordination among teams. Small companies, contracting usually small projects, are not facing problems large companies face. What typically affects them are problems associated with staffing (need of skilled and experienced developers) and human-cultural problems (few resources to solve personnel problems). For complex and large-scale applications, some of the enabling factors proposed by developers and managers are; use cases, task breakdown, up-front architecture, more formal communication and rotation of skilled developers. For small companies, developers and managers proposed collocated teams, communication and coordination among members, and the capability to keep user stories simple. DA reveals strong interaction with acceptance testing especially for

Table 3 Enabling and limiting factors for small companies

Practice	Enabling factors	Limiting factors
Planning game	Smooth software development (collocated teams, communication and coordination among members) Keep user stories simple	Project complexity Staffing problems (skilled and experienced developers) Human-cultural problems
Pair-programming	Smooth software development Creation of a collaborative and supportive environment	Efficiency in modifying code (experienced developers) Distributed teams and developers Human problems
Test-first	Communication and collaboration (developers-customer)	Skilled developers-customers Lack of automated testing tools
Refactoring	Developers' collaboration Efficiency in using refactoring tools	Lack of refactoring tools Lack of guidelines (theory)
Simple design	Efficiency in programming Skilled developers (efficiency in modifying code)	Lack of guidance Lack of documentation in distributed development
Code ownership	Smooth software development	Volume and type of projects
Continuous integration	Smooth software development Object oriented analysis	Type of projects (WEB, internet, etc.) Difficulties at the beginning of the project
On-site customer	Surrogate customer (when needed)	Type of projects (WEB, internet, etc.) Lack of capable and knowledgeable customers
Short release cycles	More upfront architecture GUI and interfaces architect Divide and conquer (valuable user stories first, valuable task breakdown, etc.)	Large and complex projects
40-hour week	Effective organization and strict implementation of both working tasks (iteration time, release time, etc.) and team meeting schedules (fixed time meetings, etc.)	Company's organizational issues Type | Volume of projects
Coding standards	Smooth software development Object oriented practices	Limited experience
Metaphor	Smooth software development Enhanced metaphors (high level designs, UML–diagrams, etc.)	Human-cultural problems Unclear practice theory Limited experience

small companies. In interviews, managers and developers pinpointed that communication and synergy starting in this early phase of the development process, are been improved substantially thanks to on-site customer and the other team-oriented practices. Planning game was characterised, by managers and developers, as a pro-

cess-oriented practice with both technical and social impacts. DS has shown that practice was fully applied by developers and supported by managers in the ratio of 11:15 and 10:15 respectively. Both managers and developers estimated that the practice was helpful in the ratio of 14:15 and 12:15 respectively.

9.2. Pair-Programming

Cultural problems highlighted by differences between agile and traditional teams, and problems caused by distribution of work across multiple teams in large and complex projects are the most significant limiting factors for large companies. Small companies also experienced problems associated with distributed teams, but human problems (unpleasant conditions or relations among staff) and the need for experienced developers (*skill-factor:* efficiency in modifying code) are considered as strong limiting factors. Managers and developers indicated the need to choose the right people, collocate teams, continuously rotate developer pairs and create a collaborative environment to support teamwork as the most significant actions to be undertaken by large and small companies. DA and interviews have shown that pair programming operates as magnifying lens enhancing both advantages and problems for small companies. Small companies having pair programming as their key practice gain more (smooth software development), but on the other hand they are facing more difficulties by its implementation because of their limited resources. In interviews, managers and developers stated that pair programming is one of the most significant success factors in XP, providing on-spot-design and continuous code reviews, spreading knowledge through team and reinforcing domain knowledge with continuous skills training (coding, tools, design patterns, refactoring). Problems have been reported on the difficulties some programmers have when working with others and the difficulties they face at the beginning of application (*acclimated time*). In addition, they underlined the need for more formal documentation when work is distributed across multiple teams. Pair programming was characterised, by managers and developers, as a team-oriented practice with both technical and social impacts. DS has shown that pairing was fully applied by developers more often than companies in the ratio of 12:15 and 9:15 respectively. Two out of three managers and developers considered that the practice was helpful (ratio: 12/15) and only one in every fifteen developer faced difficulties in its application.

9.3. Test-First

DA results, for this practice, have shown that the continual communication and collaboration between developers and between developers-customers are the most significant enabling factors for both large and small companies. Other DA findings were the strong interaction with practice-refactoring (for small companies), the strong interaction with practice-small releases (for large companies) and project simplicity (for large companies). In interviews, developers and managers stated that test-first development and in general test-driven development is one of the most significant success factors in XP. These practices provide frequent feedback, maintain

simplicity and help developers to steer their project work. The need for skilled developers-customers and the lack of automated-effective testing tools were reported by managers as limiting factors for both large and small companies. Developers pinpointed that it is tricky for them to master writing tests at the very beginning of the project, since it is difficult to determine the number of tests that had to be run in the early development phases. Test-first development was characterised, by managers and developers, as both process- and programming-oriented practice with strong technical impacts. DS has shown that the practice was fully applied by developers and supported by managers in a ratio of 13/15. One third of the managers and two-fifths of the developers estimated that the practice could be further improved.

9.4. Refactoring

Developers' collaboration and efficiency in using refactoring tools were indicated by managers and developers in DA as the enabling factors for small companies. For large and complex projects, in large companies, the use of more upfront design and design patterns are proposed by managers and developers as the enabling factors for refactoring. In interviews, managers and developers from both large and small companies complained about the lack of effective refactoring tools and that existing XP theory does not provide detailed guidelines on how to accomplish successful refactoring. Managers and developers from large companies verified that in some large projects, refactoring was used as a technique to enhance reuse and that in some others used as a recoding XP practice, clashed with existing quality control systems. Developers indicated the interaction of refactoring with the practice-continuous integration (burdensomely) and pinpointed two rules for successful refactoring: duplicated code removal and the application of design patterns or heuristics. They also indicated that the practice achieves simplicity, eliminates complexity and improves quality. Refactoring was characterised, by managers and developers, as programming-oriented practice with strong technical impacts. DS has shown that the practice was fully applied by developers and supported by managers in the ratio of 14:15 and 13:15 respectively. About one third of the managers and developers estimated that the practice could be further improved.

9.5. Simple Design

Limiting factors are the same for large and small companies as DA results have shown, namely the lack of theoretical guidance of the practice and the lack of documentation in distributed development. Managers and developers sighted the need of skilled developers efficient in programming and with capabilities in code modification and tool usage as enabling factors for small companies. For large companies unit-testing, which coordinates requirements with code production, considered as the most significant enabling factor for simple design. In interviews, managers and developers pinpointed that they all wanted simplicity in design and implementations, but they also stated that simplicity is an abstract concept, difficult to attain in the face of complexities. Developers named time restrictions

in development process and the shift of priorities in development process, as two other limiting factors for practice implementation in large companies. Simple design was characterised, by managers and developers, as programming-oriented practice with technical impacts. DS has shown that the practice was fully supported by four-fifths of the companies and fully applied by three-fifths of the developers. Two-thirds of the managers and about one-third of the developers estimated that the practice was helpful, but both sides also asserted that it could be improved.

9.6. Common Code Ownership

Concerning common code ownership, team-oriented practices, as DA results have shown, are the most important enabling factors, for both large and small companies. Collocated teams comprising of well communicating and coordinating members were found to be helpful in practice implementation, smoothening out software development especially in large companies which apply mixed processes and face cultural problems (traditional-agile teams). Results have also shown that in many cases the volume and the type of projects hinder the implementation of the practice in both large and small companies. In interviews developers and managers have confirmed that common code ownership, while interacting with most of the other practices, works best in combination with pair programming, coding standards, and especially with test-first/automated testing. Common code ownership was characterised, by managers and developers, as team-oriented practice with technical impacts. DS have shown differences in the application of the practice between companies and developers in the ration of 9:15 and 12:15 respectively. This variance confirms the results from the DA analysis above. Managers and developers considered that the practice was helpful in the ratio of 10:15 and 13:15 respectively.

9.7. Continuous Integration

DA results have shown that managers and developers proposed almost the same enabling and limiting factors for both large and small companies. Team-oriented practices and Object Oriented Analysis were indicated by managers and developers, in DA, as the enabling factors for both large and small companies. Managers and developers sighted cultural problems (traditional-agile teams), the type of projects (WEB, internet, etc.), and the difficulties teams are facing at the beginning of the project as limiting discriminating factors for large companies. Managers and developers verified that almost the same limiting factors are facing small companies, emphasizing also the interaction with the practice test-first/automated testing. Developers from large companies (DA results) confirmed that the practice improves code quality, enabling the implementation of late changes (interviews). Continuous integration was characterised as team- and technical-oriented practice, by both managers and developers. DS has shown that the practice was fully applied by developers and supported by managers in the ratio of 12:15 and 13:15 respectively, while they considered that the practice was helpful in the ratio of 13:15 and 12:15 respectively.

9.8. On-Site Customer

For both large and small companies, a business representative-the "*Customer*," being constantly near the team, helps in:

- Requirements' understanding,
- Decisions' taking,
- Development process (increasing team's experience), and
- Projects' success.

However, in reality, such access to customers is often difficult. In large-scale projects, the problem is increased as the complexity of the application domain is often beyond the experience of a customer or customers. DA results have shown that when direct access to customers is not feasible, then the use of surrogate customers who are domain experts, the use of more planning approaches (design flexibility–metaphors, etc.), and the encouragement of team practices are some alternative enabling factors, for both large and small companies. The type of projects (WEB, internet, etc.) and customer capabilities (experience, knowledge, etc.) are the most significant limiting factors, for both large and small companies. Managers and developers denoted that practice leads to a common vision of how the program works (strong interaction with practice-metaphor) and that software development becomes flexible and less costly (positive results). In interviews, managers and developers assured DA results, namely that a capable on-site customer brings rapid feedback and that developers experience new programming ideas, without wasting valuable development time, leading to a better and more usable final product. Besides, they also assured that the full use of the practice, as described in theory, was difficult to achieve and only some large projects provided a full time on-site customer. Companies often used representative users as customers. Most of the small companies invent different and simpler ways to communicate with customers when it was needed. They used telephone, fixed appointments and internet communication. On-site customer was characterised as process-and socio-technical oriented practice, by both managers and developers. DS results have shown that practice was fully used by managers and supported by developers in a small ratio of only 6:15. Managers and developers estimated that the practice was helpful in a ratio of 9:15 and 13:15 respectively.

9.9. Short Release Cycles

Small and frequent releases provide early and immediate business value to both customers and developers. However, both DA results and interviews have shown that to provide working software in short period times it is not an easy practice, demanding the deployment of specific development practices and team capabilities. Both managers and developers suggested as effective enabling factors, for both large and small companies, the use of more upfront architecture, the use of dividing and conquer techniques (implement first the most valuable user stories, valuable task breakdown, etc.), and the use of GUI and interface architecture. The need for experienced and skilled personnel is one of the needing factors proposed by both managers and developers, especially for small companies. The fast and less costly

software development were the most important positive results denoted by managers and developers for both large and small companies. Large and complex projects hinder practice implementation, and for large companies problems are amplified when project work is distributed among multiple teams. In interviews, developers and managers indicated that short releases, as the natural outcome in their development process, helps them organise better their working schedule and enriches their skills and experience. Short release cycles was characterised as both team- and process-oriented practice with technical impacts, by both managers and developers.

DS has shown that the practice was fully applied by developers and supported by managers in the ratio of 13:15 and 11:15 respectively, while they considered that the practice was helpful in the ratio of 12:15 and 13:15 respectively.

9.10. 40-Hour Week

Managers and developers, through DA and interviews, confirmed that this practice is one of those that were devised to address the human side of software development expressing team's willing for a sustainable level of productivity. They proposed the same enabling and limiting factors for both large and small companies. Effective organization and strict implementation of both the working tasks (iteration time, release time, etc.) and team meeting schedules (fixed time meetings, etc.) were suggested as enabling factors. They considered that company's organizational issues and type|volume of projects are the most important limiting factors especially for large companies. Another interesting DA finding was the strong interaction with the practice–test first | acceptance testing, especially for small companies. In interviews, managers from large companies complained that they often face stressing situations, when either the production schedule is too tight (business controls both time and scope) or the team can not finish its work on time. In these stressed situations, managers prefer to change the project scope or schedule. Some of the small companies apply an 8-hour-work day, but not on a standard basis schedule. In these companies, management preserved partnership relations with programmers and the practice was applied but in a more relaxed time schedule. 40-hour week was characterised as teamand social oriented practice, by both managers and developers. DS has shown that the practice was fully applied by developers and supported by managers in a small ratio of 7:15 and 9:15 respectively, while a large company did not apply the practice.

9.11. Coding Standards

For both managers and developers, as DA and interviews have shown, the use of rules in writing code emphasizes communication through the code and ensures readability of the system. Team-oriented practices (smooth software development for small companies) and Object Orientation are suggested, by managers and developers, as enabling factors for both large and small companies. Large company's organizational issues and the lack of experience for small companies were suggested as limiting factors. An interesting DA finding especially for small companies was the strong interaction with the practice–pair programming. Managers denoted in inter-

views that apart from pair programming, coding standard has a strong interactive and dependent relation (characterised as natural binding) with simple design and common code ownership. Coding standards was characterised as programming- and team-oriented practice with technical impacts, by both managers and developers. DS has shown that the practice was fully applied by developers and supported by managers in the ratio of 12:15 and 11:15 respectively. Managers and developers estimated that the practice was helpful in the ratio of 13:15 and 12:15 respectively.

9.12. Metaphor

Managers and developers, through DA and interviews, confirmed that metaphor is devised to steer the project and together with planning game and refactoring to create a stable and simple system structure. However, they also verified that metaphor was the hardest to apply practice, especially in large companies. They defined as limiting factors, for large and small companies, the cultural problems (traditional-agile teams), the lack of experience, and the unclear practice theory. Diverse process-practices, such as enhanced metaphors (high level designs, UML–diagrams, etc.) and more architectural design for large and small companies were suggested as the enabling factors for smoothing out software development process. Manager satisfaction and the quick transfer of ideas were mentioned as positive results, while the practice strongly interacts with planning game (especially in small companies). In interviews, managers and developers assured that metaphor improves communication and influence the process by little moves. They also reported that they used metaphor as a background for system's design and as a guiding scheme for naming classes, methods, etc. Managers, from large companies, denoted that it was difficult to apply the practice in a constant way (they often replaced or enhanced metaphors) and that in certain projects they deliberately avoid the involvement of the developers. Metaphor was characterised as team- and technical-oriented practice, by both managers and developers. DS has shown that the practice was fully applied by developers and supported by managers in a small ratio of 6:15 and 8:15 respectively. Managers and developers found that it was difficult for them to apply the practice as it is described in literature in the ratio of 6:15 and 3:15 respectively.

10. Conclusions and Future Research

The main purpose of this empirical study was to explore enabling and limiting factors for the use of XP practices. Significant affecting factors and their impact on the XP practices in different contexts, proposed by the authors and enhanced by the participants in the first part of the study, have been presented and comprehensively analysed. A cause-effect model, on which a generic XP system builds up, consisting of XP values, practices and affecting factors, was developed and used as our conceptual framework. This model was used in evaluating the practices along two dimensions: large companies versus small companies and managerial versus development staff. With this holistic consideration we tried to further understand the XP system approach and to gather answers to research questions we placed in the beginning of the study. Although the responders did not provide hard data they had been able to develop strong opinions which were expressed and captured through

the data collection phase of the study. Results from stepwise Discriminant Analysis and Descriptive Statistics, combined with the findings of qualitative analysis of the data gathered with interviews, show the advantages and difficulties caused by the implementation of the XP practices in different sized companies, undertaking different kind of projects.

Based on the perceptions of the participants, the results indicate that most of the discriminating factors, either enabling or limiting, derive mainly from the environmental differences and differences in the development processes that exist between these companies. All these factors lead companies to develop their own tailored XP method and way of working to meet their specific requirements. Because both large and small companies face various problems with pair programming, common code ownership, on-site customer, 40-hour week and metaphor, they usally implement those practices in different ways. Problems with pair programming concern cultural differencies among development teams, the type of projects and the distribution of the developers. On-site customers were provided only in large projects leading companies to invent different and simpler ways to communicate with customers when it was needed (telephone, internet etc.). Large companies tended to break 40-hour week and to prevent the involvement of the developers in both common code ownership and metaphor for different reasons. Small companies strongly demanded skilled and experienced developers, but at the same time they were more flexible in applying XP practices. They applied 40-hour-week but in a more relaxed time–schedule, if needed. Managers and developers underlined the lack of a detailed description for some practices, such as simple design, coding standards and metaphor. The interaction between the majority of the practices was considered to be actually a factor contributing to their success. Pair programming and test-driven development were found to be significant success factors for the rest of the entire XP system. Pairing encourages the tacit transmission of knowledge and promotes continuous training, while test-driven development brings feedback helping developers and managers to decide how to steer the project. Finally, communication and synergy between skilled personnel, initiated in planning game and then repeated in most of the other practices, are other significant success factors.

This study is an initial attempt, aiming at the characterization of current state-of-the-practice in one software industry, as a starting point for further research. It appears that certain practices must be further specified and enriched with guidelines helping their application, to become fully operational in the context of the XP System. Besides, companies may assess whether the XP system framework, enriched with social and technical affecting factors, could help them develop software in a more qualitative, faster and efficient way. The findings of this study should also help internal or external coaches and researchers, in the field of XP, to better understand how different sized companies are approaching, from a holistic view, the implementation of the practices. Another research path could be the characterization of transition strategies towards agile methods and the relation of these strategies to the factors we identified.

Acknowledgments We wish to thank managers and developers of the companies participating in this research work. We also wish to thank the editor and the anonymous reviewers for the time they spent reviewing our paper, for their helpful comments and the valuable feedback they provided. This work is funded by the Greek Ministry of Education (25%) and European Union (75%) under the EPEAK II program "Archimedes II".

Appendix A: The Completed Factor-List

Preconditions–Needed Factors

Description	Values*
1. Human-cultural factors	Y/N or G/B
2. Developers' collaboration	Y/N or G/B
3. Customer collaboration	Y/N or G/B
4. Managers' collaboration	Y/N or G/B
5. Project team collaboration	Y/N or G/B
6. Customer capabilities (technical-business)	Y/N or G/B
7. Developer capabilities	Y/N or G/B
8. Manager capabilities	Y/N or G/B
9. Partnership relations	Y/N or G/B
10. Company's organizational issues size–role separation etc.)	Y/N or G/B
11. Documentation	Y/N or G/B

* G/B=Good/Bad; Y/N=Yes/No.

Postconditions–Reflective Positive Results

Description	Values
1. Increased experience	Y/N or I/D
2. Improved cost estimation	Y/N or I/D
3. Faster transfer of ideas	Y/N or I/D
4. Tacit knowledge transfer	Y/N or I/D
5. Smooth software development	Y/N or I/D
6. Fast software development	Y/N or I/D
7. Less costly software development	Y/N or I/D
8. Project simplicity	Y/N or I/D
9. Design flexibility	Y/N or I/D
10. Code quality	Y/N or I/D
11. Customer satisfaction	Y/N or I/D
12. Developer satisfaction	Y/N or I/D
13. Manager satisfaction	Y/N or I/D
14. Customer feedback	Y/N or I/D
15. Pleasant working conditions	Y/N or I/D

* Y/N=Yes/No; I/D=Increase/Decrease.

Skills

Description	Values
1. Efficiency in programming (OOP)	Y/N
2. Efficiency in modifying code	Y/N
3. Efficiency in decomposition (user stories)	Y/N
4. Efficiency in writing effective tests	Y/N
5. Efficiency in using tools	Y/N
6. OOA	Y/N
7. OOD (Patterns-Heuristics)	Y/N
8. Communication	Y/N
9. Team working	Y/N

Negative Factors

Description	Values
1. Limited experience	Y/N
2. Not enough stories	Y/N
3. Big project (s)	Y/N
4. Volume and type of projects in large companies	Y/N
5. Distributed developers	Y/N
6. Low team cohesion	Y/N
7. Difficulties in applying XP in early project phases	Y/N
8. Intensity of the development process	Y/N
9. Missing tests in various phases	Y/N
10. Time pressure	Y/N
11. Project complexity	Y/N
12. Type of project (WEB based-non WEB based)	Y/N
13. Unclear practice theory	Y/N
14. Limited funds | resources	Y/N
15. Unpleasant conditions or relations	Y/N

Interactions-Dependencies

1. Planning game
2. Pair-programming
3. Test-first/automated testing
4. Refactoring
5. Simple design
6. Common code ownership
7. Continuous integration
8. On-site customer
9. Short release cycles
10. 40-hours week
11. Coding standards
12. Metaphor

Appendix B: The Semi-Structured Interview

Name of practice:_____

1. Choose from the list and/or propose the most significant **needed factors** for this practice implementation.

From the list	You propose
1. ☐	_____
2. ☐	_____
3. ☐	_____
4. ☐	_____
5. ☐	_____
6. ☐	_____
7. ☐	_____

2. Choose from the list and/or propose the most significant **reflective positive results** from this practice implementation.

From the list	You propose
1. ☐	_____
2. ☐	_____
3. ☐	_____
4. ☐	_____
5. ☐	_____
6. ☐	_____
7. ☐	_____

3. Choose from the list and/or propose the most significant **negative factors** for the practice implementation.

From the list	You propose
1. ☐	_____
2. ☐	_____
3. ☐	_____
4. ☐	_____
5. ☐	_____
6. ☐	_____
7. ☐	_____

4. Choose from the list and/or propose the most significant **needed skills** for the practice implementation.

From the list	You propose
1. ☐	_____
2. ☐	_____

3. ☐ _____
4. ☐ _____
5. ☐ _____
6. ☐ _____
7. ☐ _____

5. Choose from the list the practice/-s **interacting** with the examined practice.

1. ☐
2. ☐
3. ☐
4. ☐
5. ☐
6. ☐
7. ☐
8. ☐

Don't know/Don't answer: ☐

6. Choose from the list the practice|-s with which the examined practice has **dependence** relations.

1. ☐
2. ☐
3. ☐
4. ☐
5. ☐
6. ☐
7. ☐
8. ☐

Don't know/Don't answer: ☐

7. Do you believe that **human issues** and **culture** affecting the implementation of the examined pratice?

Yes ☐
No ☐
Don't know/Don't answer: ☐

8. Please list three practices you didn't apply recently. Please provide reason for this and report problems.

9. Do you have improvement suggestions for any of the XP practices?

10. Have you used additional concepts, tools, roles, teaming activities, techniques, deliverables, standards, habits or modeling languages?

11. Please characterize each practice (technical, social or both).

12. Further comments.

Appendix C: The Questionnaire

Questionnaire for Company and Developer

Company's Name: _____
Country: _____ City: _____
Address: _____

Products: _____
(i.e., Business, Support, Internet/Communication, Multimedia, etc.)

Customer: _____
(i.e., Internet, Business, etc.)

Business Model: _____
(i.e., Market, Contract, etc.)

Age of Company:	Exactly: ☐ years	Greater than: ☐ years	
Total Employees:	Exactly: ☐	Greater than: ☐	
Role in Company:	Teamleader: ☐	Manager: ☐	Owner: ☐
	Developer: ☐	Coatch: ☐	Other: ☐
Current project:	Internet/Web: ☐	DBMS/SQL: ☐	Consulting: ☐
	Software Development (general): ☐	Bank/Insurance: ☐	
		Tool/Framework: ☐	
	Biotechnology: ☐		
	Maintenance/Refactoring: ☐	Other: ☐	
Programming Languages:	Java ☐	C++ ☐	Delphi ☐
	V. Basic ☐	C ☐	Pascal ☐
	Cobol ☐	Smalltalk ☐	Lisp ☐
	Other ☐		
Agile Methodologies Only:	Yes ☐		No (mixed methodologies) ☐
Successful XP projects: ☐	Failed XP projects: ☐		Total XP projects: ☐
Persons in Current proj.: ☐	Pairing? Yes ☐ No ☐		Pairs in cur. project: ☐

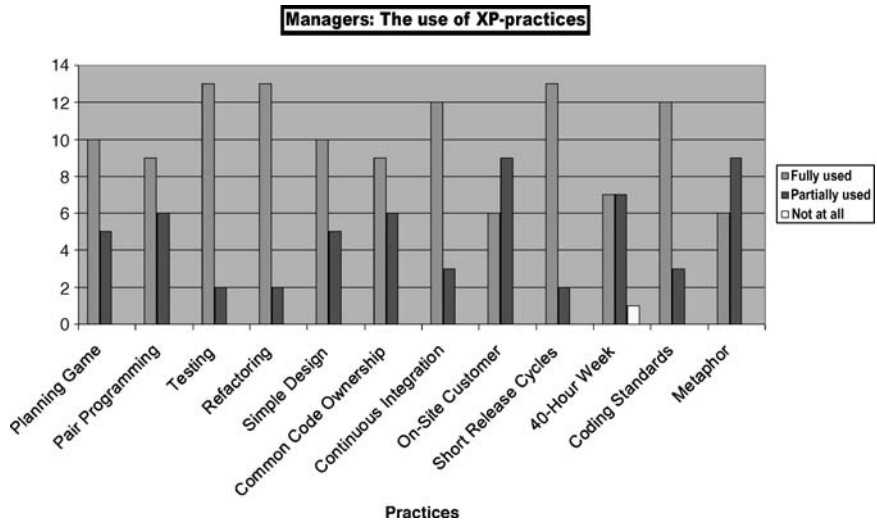
XP-Practices. *Which XP-Practice do you use?* *Do you find the practice:*

	Fully used	Partially used	Not at all used	Helpful	Improvable	Hard to apply
	(cross only one of three)					
Planning game	☐	☐	☐	☐	☐	☐
Pair programming	☐	☐	☐	☐	☐	☐
Test-driven development	☐	☐	☐	☐	☐	☐
Refactoring	☐	☐	☐	☐	☐	☐
Simple design	☐	☐	☐	☐	☐	☐
Common code ownership	☐	☐	☐	☐	☐	☐
Continues integration	☐	☐	☐	☐	☐	☐
On-site customer	☐	☐	☐	☐	☐	☐
Short release cycles	☐	☐	☐	☐	☐	☐
40-hour-week	☐	☐	☐	☐	☐	☐
Coding standards	☐	☐	☐	☐	☐	☐
Metaphor	☐	☐	☐	☐	☐	☐

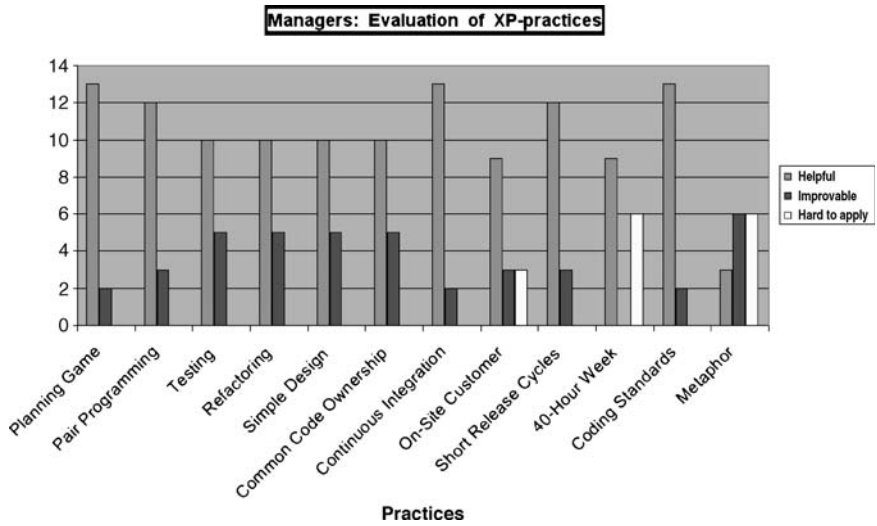
Further comments

Appendix D: DS Results for All Practices

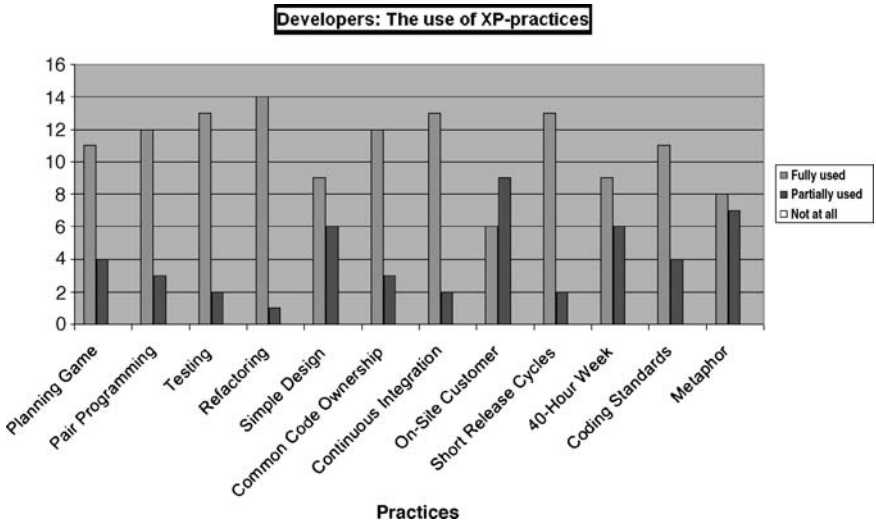
Companies: Total Use of XP-Practices



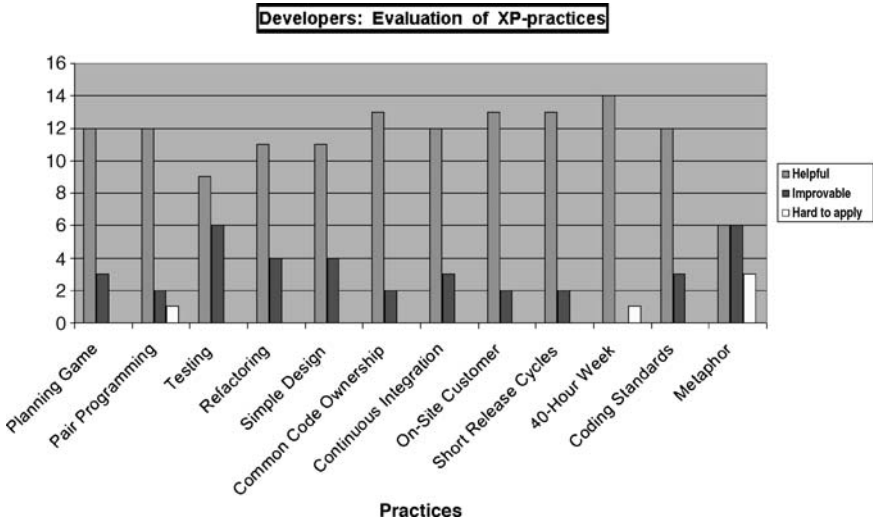
Companies: Total Evaluation of XP-Practices



Developers: Total Use of XP-Practices



Developers: Total Evaluation of XP-Practices



References

Aveling B (2004, June) XP Lite considered harmful? Fifth International Conference on Extreme Programming and Agile Processes in Software Engineering. Garmisch–Partenkirchenm, Germany

Basili VR, Selby RW, Hutchens DH (1986, July) Experimentation in software engineering, IEEE Trans Soft Eng 12(7):733–743

Beck K (1999) Embracing change with extreme programming. Computer 13(10):70–77

Beck K (2000) Extreme programming explained: embrace change. Addison-Wesley, Reading, Massachusetts

Brooks FP (1987) No silver bullet: essence and accidents of software engineering. Computer 20(4):10–19

Butler S, Hope S, Gittins R (2001, May 20–23) How distance between subject and interviewer affects the application of qualitative research to extreme programming. Proceedings of 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering. Villasimius, Sardinia, Italy

Cockburn A, Williams L (2000, June) The costs and benefits of pair programming, extreme programming and flexible processes in software engineering. XP2000, Cagliari, Italy

Cockburn A (2001) Agile software development. In: Cockburn A, Highsmith J (eds) The Agile Software Development Series. Addison-Wesley Longman, Reading, Massachusetts

El Emam K, Fusaro P, Smith B (1999) Success factors and barriers for software process improvement. Better software practice for business benefit: principles and experience. Conference, Location, IEEE Computer Society

El-Emam K (2003) Finding success in small software projects. Agile Project Management, Vol. 4

Erdogmus H (2005, January) On the effectiveness of test-first approach to programming. Proceedings of the IEEE Transactions on Software Engineering, 31(1). NRC 47445

Erdogmus H, Wang Y (2004, August 15–18) The role of process measurement in test-driven development. Proceedings of XP Agile Universe 2004. Calgary, Alberta, Canada. NRC 47454

Erdogmus H, Williams L (2003) The economics of software development by pair programmers, The Engineering Economist 48(4): 33. NRC 47101

Fowler M (2001) The new methodology [on-line]. Available at URL http://www.martinfowl er.com/articles/newMethodology.html [referred on 18.1.2002]

George B, Williams L (2003) An initial investigation of test driven development in industry. Presented at ACM Symposium on Applied Computing. Melbourne, Florida

Geras A, Smith M, Miller J (2004) A prototype empirical evaluation of test driven development. Proceedings of the 10th International Symposium on Software Metrics (METRICS '04)

Gittins R, Hope S, and Williams I (2001, May 20–23) Qualitative studies of XP in a medium sized business. Proceedings of 2nd International Conference on Extreme Bibliography 149 Programming and Flexible Processes in Software Engineering. Villasimius, Sardinia, Italy

Glaser BG, Strauss AL (1967) The Discovery of grounded theory: strategies for qualitative research. Aldine Publishing Company, Chicago

Hirschheim R, Klein HK, Lyytinen K (1995) Information systems development and data modelling: conceptual and philosophical foundations. Cambridge University Press, Cambridge

Hulkko H, Abrahamsson P (2005) A multiple case study on the impact of pair programming on product quality. ICSE '05–USA

Jeffries R, Anderson A, Hendrickson C (2001) Extreme programming installed. In: Beck K (ed) The XP Series. Addison Wesley, Upper Saddle River, NJ

Kauffman DL Jr (1980) Systems I. An introduction to systems thinking. Carlton, Minneapolis, Minn

Krzanowski WJ (1993) Principles of multivariate analysis. Oxford University Press, UK

Layman L, Williams L, Cunningham L (2004a) Motivations and measurements in an Agile case study. ACM SIGSOFT Foundation in Software Engineering Workshop Quantitative Techniques for Software Agile Processes (QTE–SWAP). Newport Beach, CA

Layman L, Williams L, Cunningham L (2004b) Exploring extreme programming in context: an industrial case study. Agile Development Conference. pp 32–41

Lehman M (1996) Feedback in the software evolution process, keynote address. CSR 11th Annual Workshop on Software Evolution: Models and Metrics. Dublin, 7–9th Sept. 1994, also in Information and Softw. Technology, sp. is. on Software Maintenance. 38(11): 681–686

Lincoln YS, Guba EG (1985) Naturalistic inquiry. Sage, Thousand Oaks

Lindvall M, Muthig D, Dagnino A, Wallin C, Stupperich M, Kiefer D, May J, and Kähkönen T (2004, December). Agile software development in large organizations. Computer, IEEE
Melnik G, Maurer F (2005) A cross-program investigation of students' perceptions of agile methods. ICSE '05–USA
Maurer F, Martel S (2002, Jan/Feb) Extreme programming: rapid development for web-based applications. IEEE Internet Computing 6:86–91
Müller M, Padberg F (2002, May) Extreme programming from an engineering economics viewpoint. International Workshop on Economics-Driven Software Engineering Research (EDSER), Orlando, Florida, USA
Müller M, Padberg F (2003, September) On the economic evaluation of XP projects. European Software Engineering Conference (ESEC). Helsinki, Finland
Müller M, Hagner O (2002) Experiment about test-first programming. Presented at Conference on Empirical Assessment in Software Engineering. Keele
Nosek J (1998, March) The case for collaborative programming. Communications of the ACM 41(3):105–108
Padberg F, Müller M (2003, September) Analyzing the cost and benefit of pair programming. International Symposium on Software Metrics (Metrics). Sydney, Australia
Patton MQ (1990) Qualitative evaluation and research methods (2nd edition). Sage Publications, Inc., Newbury Park, CA
Paulk MC (2001) Extreme programming from a CMM perspective. Software 18(6): 19–26
Reifer DJ (2002, August) How to get the most out of extreme programming/agile methods. 2nd XP and 1st Agile Universe Conference. Chicago, IL. pp 185–196
Rumpe B, Schröder A (2002, May 26–30) Quantitative survey on extreme programming projects. Proceedings of the 3rd International Conference on Extreme Programming and Flexible Processes in Software Engineering. Alghero, Sardinia Italy
Seaman CB (1999, July/August) Qualitative methods in empirical studies of software engineering. IEEE Trans Soft Eng 25(4):557–572
Senge PM (1990) Fifth discipline—the art and practice of the learning organization. Doubleday, New York, USA
Sommerville I, Sawyer P (1997) Requirements engineering a good practice guide. John Wiley & Sons Ltd., Chichester
Strauss A, Corbin J (1990) Basics of qualitative research: grounded theory procedures and techniques. Sage Publications, Inc., Newbury Park, CA
Taylor SJ, Bogdan R (1984) Introduction to qualitative research methods. John Wiley and Sons, New York
Wernick P (1996, April) A belief system model for software development: a framework by analogy. PhD Thesis, Department of Computer Science, University College London
Wiki (2003) http://www.c2.com/cgi/wiki? [05.11.2003]
Williams L, Erdogmus H (2002, May) On the economic feasibility of pair programming. International Workshop on Economics-Driven Software Engineering Research EDSER-4, Orlando, Florida, USA
Williams L, Kessler R, Cunningham W, Jeffries R (2000, July–Aug) Strengthening the case for pair-programming. IEEE Software 17(14): 19–25
Williams L, McDowell C, Fernald J, Werner L, Nagappan N (2003a) Building pair programming knowledge through a family of experiments. IEEE International Symposium on Empirical Software Engineering (ISESE). pp 143–152
Williams L, Maximilien E, Vouk M (2003b) Testdriven development as a defect-reduction practice. Presented at 14th International Symposium on Software Reliability Engineering
Williams L, Krebs W, Layman L, Antón A (2004) Toward a framework for evaluating extreme programming. Empirical Assessment in Software Engineering (EASE). pp 11–20
Wohlin C, Runeson P, Höst M, Ohlson M, Regnell B, Wesslén A (2000) Experimentation in Software Engineering: An introduction. Kluwer Academic Publishers, USA

Panagiotis Sfetsos is a Lecturer of computer science at Technological Education Institute of Thessaloniki, Greece, Department of Informatics, since 1990. He is a Ph.D. student at the Programming Languages and Software Engineering Lab of the Department of Informatics, Aristotle University of Thessaloniki, Greece, since January 2003. His research interests include Agile Methods, Extreme Programming, software evaluation, measurement, testing and quality. He received his B.Sc. in Computer Science and Statistics from the University of Uppsala, Sweden, in 1981, and then worked for several years in software development at industry and education.

Lefteris Angelis received his BSc and Ph.D. degree in Mathematics from Aristotle University of Thessaloniki (A.U.Th.). He works currently as an Assistant Professor at the Department of Informatics of A.U.Th. His research interests involve statistical methods with applications in software engineering and information systems, computational methods in mathematics and statistics, planning of experiments and simulation techniques.

Ioannis G. Stamelos is an Assistant Professor of computer science at the Aristotle University of Thessaloniki, Dept. of Informatics. He received a degree in Electrical Engineering from the Polytechnic School of Thessaloniki (1983) and the Ph.D. degree in computer science from the Aristotle University of Thessaloniki (1988). He teaches language theory, object-oriented programming, software engineering, software project management and enterprise information systems at the graduate and postgraduate level. His research interests include empirical software evaluation and management, software education and open source software engineering. He is author of 60 scientific papers and member of the IEEE Computer Society.