Is Extreme Programming Just Old Wine in New Bottles: A Comparison of Two Cases

Merisalo-Rantanen Hilkka, Helsinki School of Economics, Finland
Tuunanen Tuure, Helsinki School of Economics, Finland
Rossi Matti, Helsinki School of Economics, Finland

ABSTRACT

This paper explores Extreme Programming (XP) as an information systems development approach and argues that it is mainly old wine in new bottles. We take an interpretive and critical view of the phenomenon. We made an empirical study of two companies that apply an XP style development approach throughout the information systems development lifecycle. The results of our research suggest that XP is a combination of best practices of traditional information systems development methods. It is hindered by its reliance on talented individuals, which makes its large scale deployment as a general purpose method difficult. We claim that XP can be useful for small teams of domain experts, who are physically close to and able to communicate well with the end-users and who are good designers and implementers. However, these skilled and motivated individuals with high working moral can exhibit high productivity regardless of the methods used, if they are not overly constrained by bureaucracy.

Keywords: Agile methods, Extreme Programming, information systems development, case study
INTRODUCTION - FROM METHODOLOGIES TO METHODS AND AGILITY

Ever since the first major software systems were developed, a chronic “software crisis” has been seen either looming ahead or haunting the community (Brooks, 1975). Solutions have been sought mostly from raising the productivity of programmers, making systems less defective (e.g. process management and development approaches (Boehm, 1988; McConnell, 1996)) or developing systems by methods that treat the end-users as equals to the designers in the development process (e.g. Participatory Design (Bjerkenes & Bratteteig, 1995; Grudin, 1991)). In this paper we first discuss these approaches for organizing information systems development (ISD). This leads us into an understanding of emerging way of developing information systems, Extreme Programming (XP), which can be seen as an amalgamation of these practices.

Hirschheim et al. (1995) have defined an ISD methodology (ISDM) as an organized collection of concepts, methods, beliefs, values and normative principles supported by material resources. In turn, Iivari et al. (1998) have argued that there are five different ISD research paradigms: professional work practices, trade unionist, interactionist, Speech Act based, and Soft Systems Method (SSM). The first and the second ones are interesting for the topic of our paper as they are more practice oriented and focus on the way of working. The professional work practice approach promotes increased professionalism of IS designers. According to this approach, methods can support inspired practitioners but they cannot replace experience, i.e. different occasions need different working practices. Trade unionist approach takes a more democratic perspective. It argues that ISD is a process that tries to develop conditions for effective worker participation in IS and the quality of work.

Extreme Programming tries to address end-user participation and increased quality of work by emphasizing the use of professional work practices and ethical software development. Thus it is in sharp contrast with the early code-and-fix paradigm of ISD. This method has been blamed to contain many problems, starting with poorly understood requirements and problematical structure of coding and resulting in great expenses when fixes are needed later on (Boehm, 1988). The waterfall model emerged as a systematic, sequential solution to software development problems (Brooks, 1975; Hirschheim, Klein, & Lyytinen, 2003). The IS product was not delivered until the whole linear sequence had been completed. As projects became larger and more complex, problems like stagnant requirements and badly structured programming started to arise.
Overlapping the phases (Fairley, 1985; Pressman, 2000; Sommerville, 2001) and the introduction of the more incremental spiral model (Boehm, 1988; Iivari, 1990a, 1990b) resolved many of the difficulties mentioned earlier. This model presents the software process as a spiral, where each of the loops can be considered to represent one fundamental development step. Thus, the innermost loop might be concerned with requirements engineering, the next with design and so on (Sommerville, 2001). The spiral model assumes a risk-driven approach to the software development rather than a primarily document-driven (waterfall) or code-driven (prototyping) approach (Boehm, 1988). Each cycle incrementally increases the system’s degree of definition and simultaneously decreases its degree of risk (Boehm, Egyed, Kwan, Port, & Madachy, 1998).

The iterative models were augmented with more dynamic approaches with less bureaucracy. For example, in incremental development software is developed in small but usable pieces that can be delivered early on to a customer. Each increment is an operative subset of the final software system and builds on the increments that have already been developed (Pressman, 2000).

Parallel to ISD organization changes, the design craft itself has been evolving. It has been argued (McKeen, Guimaraes, & Wetherbe, 1994, pp. 427-428) that user participation improves the quality of the system in several ways such as “… providing a more accurate and complete assessment of user information requirements … providing expertise about the organization the system is to support … avoiding development of unacceptable or unimportant features, and improving user understanding of the system …”. Nevertheless, there was no common definition of how users should be involved (Carmel, Whitaker, & George, 1993). To solve this problem, many approaches arose, most notably Participatory Design (PD) (Bjerkenes & Bratteteig, 1995) and Joint Application Development (JAD) (Clemont & Besselaar, 1993). While taking a different view of end-users’ role, both stress the involvement of users in the development process and design decisions. New methods and tools to help the communication among IS designers and users are continuously developed (e.g. Shoval & Kabeli, 2001; Liu, Pu & Ruiz, 2004). One of the key arguments of this discussion has been how to reconnect the designer and user again (Grudin, 1991).

Last aspect that XP raises is the empowerment and productivity increase of developers. Traditionally these have been sought by raising the abstraction level of the software development tools (e.g. through high level languages and CASE). However, programmers have often seen these more as an obstacle (Ramesh & Jarke, 2001). One suggested solution is the employment of work practices that let the most talented developers unleash their power (e.g. surgical teams
(Brooks, 1975) and pair programming, which, according to Williams & Kessler (2002), dates back to Brooks in the 1950s.

To conclude, XP seeks to solve many of the problems of traditional software development by combining the best practices from the past research and practice of ISD. Firstly, XP aims at employing participatory design by really engaging the business or end-users into the IS development process. Secondly, XP seeks to add flexibility to the development process and to organize the work into small packages with clear deliverables. Finally, XP tries to squeeze maximal productivity out of the developers by using concepts such as pair programming.

In this study we explore the concept of Extreme Programming as a new approach to IS development. We argue that instead of being a new development model, XP can be described as a way of working, and it actually does not bring anything completely new into the ISD discussion. However, we argue that XP may add some value into the development process discussion as it connects prototyping and end-user oriented development in a way that could deliver systems that are a better match for the end-user needs. We explore these arguments in the following by first looking at XP and its roots in agile methods in section two. In the third section we describe the methodology used in studying our target organizations. In section four we present the case studies. In section five we discuss the findings from the case companies. In the final section we draw conclusions based on the cases and point out future research challenges.

**AGILE METHODS AND EXTREME PROGRAMMING**

Agile software development methods can be defined as using human- and communication-oriented rules in conjunction with light, but sufficient, rules of project procedures and behavior (Cockburn, 2002). These four rules are: individuals and human interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan (Agile Manifesto, 2003). It is argued that these methods emphasize the soft, or human, side of software development over the institutional aspect. The emphasis on communication and programmers’ morale is common to all agile methods. In accordance with Conrad (2000), agile methods focus on people as the primary drivers of development success. In the following, we focus on key principles of one agile method, Extreme Programming (XP), first introduced by Kent Beck (1999). For a more detailed overview of agile methods in general see for instance (Abrahamsson, 2003; Abrahamsson, Warsta, Siponen, & Ronkainen, 2003).
According to Beck (1999), “XP is a lightweight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly changing requirements.” (p. xv) and “XP is a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software.” (p. xvii). In turn, Abrahamsson et al. (2003, p. 245) have defined XP as “A collection of well-known software engineering practices... The novelty of XP is based on the way the individual practices are collected and lined up to function with each other.”

XP addresses risk and value of software at all levels of the development process. According to Beck customers (or managers) can pick three out of four control variables (these are cost, time, quality and scope) and the development team decides on the fourth (Beck, 1999). Technical people are responsible for work estimates, technical consequences of business decisions, development process and detailed scheduling within a release. Team size should be in maximum about 12 designers and the software not excessively complex. (Beck, 1999)

The project management strategy of XP maximizes the value of the project by providing accurate and frequent feedback about progress, many opportunities to dramatically change the requirements, a smaller initial investment and the opportunity to go faster. In XP cost, time and quality of a component are regarded as fixed control variables decided by customers and managers. Within these limits the development team focuses on the variable development scope, i.e. on the functionality of the parts. The programming strategy of XP is to keep the code easy to modify. (Beck, 1999)

The 12 principles or rules of the XP methodology are: planning, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, on-site customer, coding standards and a 40-hour week. (Beck, 1999)

The principal values are communication, simplicity, feedback and courage. The effect of stressing testing, pairing and estimating in the development process is that programmers, customers and managers have to communicate. Simplicity means doing the simplest thing which could possibly work, and adding complexity later if it is really needed. Feedback works on different time scales: minutes, days, weeks and months. Courage is needed to change the basic architecture or to code a piece of software again from scratch. Basic principles for decision-making are derived from these values: rapid feedback, simplicity, incremental change, embracing change and quality work. (Beck, 1999)
As seen in Figure 1, an XP project begins with a task called an architectural spike. The outcome of this task is a system metaphor, i.e. the infrastructure, the standards and the working habits of the XP project. Beck (1999) says that a metaphor is a simple story of how the whole system works, for instance an outsourcing contract, or software architecture. It helps everyone in the project to understand the basic elements and their relationships, and it is easy to communicate and elaborate. Both business and technical people participate actively in the definition of the system metaphor.

User stories are task descriptions, also called user requirements or user needs, and possibly also descriptions of expected benefits. End-users write user stories in plain text using their own terminology. Developers estimate the ideal development time of the story, which can vary from one to three weeks. User stories must be combined or broken down if these limits are not reached. A spike solution is programmed if it is needed to make the estimates more accurate. A release plan lays out the overall project. It specifies which user stories are going to be implemented for each release and when each release will be finished. If the velocity of the development changes dramatically, a new release-planning meeting should be held to re-estimate and re-negotiate the release plan.

Figure 1 Extreme Programming Project, adapted from (Conrad, 2003; Extreme Programming Organization, 2002)
The iteration task of an XP project produces a new version or release of the program in progress for acceptance tests. A program or piece of code is integrated into the system either after it has passed all the unit tests or after some smaller part of the planned functionality has been finished. Each developer must integrate and release his/her code every few hours or at least once a day. This kind of continuous integration avoids and detects compatibility problems early, and everyone always works with the latest version of the system. This approach also avoids many of the problems of too rigorous and formal approaches by stressing increments and iteration over rigor and waterfall development (Joosten & Purao, 2002).

Acceptance tests are run on the latest version of the system to ensure the functionality of the system. End-users are responsible for acceptance tests, and they specify the test scenarios based on user stories. They also review the test scores and prioritize the corrections needed.

Finally, after the end-user or customer has approved of a small unit of functionality, it is released into the customer’s environment. Small frequent releases give a possibility to get feedback from the users early on and to make changes into the release plan if necessary. We have complemented Figure 1 with the feedback arrows to emphasize the possible effects of the end-user feedback. Very often the feedback may be the driver for modified or new user stories that is also common with more traditional ways of development (Boehm, 1988). Similarly feedback may result in changes to the release plan. Furthermore, it is possible, although exceptional, that architectural spike of the project is revised.

In XP coding is done in pairs on one workstation, and pairs are changed continuously. The code should be collectively owned, and each programmer is allowed to change the code, one programmer at a time. The code is refactored continuously to improve its quality and to make it as simple as possible without making any changes into its functionality or its features.

**METHODOLOGY OF THIS STUDY**

Some recent research has focused on the planned and systematic adoption of XP in different contexts (Abrahamsson, 2003; Abrahamsson, Salo, Ronkainen, & Warsta, 2002; Back, Milovanov, Pores, & Preoteasa, 2002; Elssamadisy & Schalliol, 2002; Reifer, 2002; Salo & Abrahamsson, 2004). However, we were not able to find too many studies focusing on the natural evolution of ISD practices towards more agile approaches. Notable exceptions are Aoyama (1998), Vanhanen et al. (2003) and Murru et al. (2003), who describe institutionalization of agile practices in organizations. Hence, we wanted to study further why and how XP is adopted and used in everyday software production. Furthermore, we were interested to see whether the
method was intentionally selected or if it had gradually evolved based on the methods used before.

We decided to take an interpretive, but at the same time critical approach (Myers, 1997). We followed the guidelines of Klein and Myers (1999) and adapted qualitative research as a means of trying to understand this complex and fast moving IS research topic. We turned to the case study approach that Wynn (2001) has advocated as the most appropriate qualitative method in studying social processes and trying to understand users in the local level. In the case descriptions we adapted the principles of interpretive case studies presented by Walsham (1995), in contrast to the positivist approach to case studies. These principles are: reporting details of the selected research sites, the reasons why these sites were chosen, the number of people interviewed, the interviewees’ hierarchical or professional position, secondary sources of data, data gathering period, how field interviews and other data were recorded, the description of the analysis process, and finally, how the iterative process between field data and theory took place and evolved over time.

The case companies were selected in two phases. We began by actively seeking companies that use agile development practices and tried to identify potential candidates for our study. We did this by gathering information from other researchers of agile methods in Finland and discussing with ISD personnel in several potential case companies concerning the development methods in use. Thereafter, two companies employing XP style practices were selected. The case companies were intentionally selected from different industries, a manufacturing company versus a software developer consultancy. The companies also differed greatly in reasons for selecting this kind of approach to IS development and the drivers behind its adaptation. The first case firm had gradually evolved their own method, or way of working, whereas the second case company had made a more or less deliberate decision to employ agile development practices.

In each company we focused on one major system or software central to the company. The system is in maintenance phase and under continuous renewal after several years of development. We conducted semi-structured theme interviews with two IT-managers, a business development manager and a senior consultant. We also received written documents as well as other complementary information on the IS development processes. Later on, the data were complemented by telephone discussions and e-mails. The data collection was conducted during spring 2003. The interviews were tape-recorded and transcribed, and later validated with the interviewees. The interviewees also verified and accepted the final version of the case
The questions of the semi-structured interviews are available from the authors on request.

The data analysis was done by comparing the interview data to the general ISD process literature with focus on agile methods. More specifically, we sought to understand how companies complied with the requirements of XP, as proposed by the XP manifesto (Beck, 1999). This iterative process is reported in the following sections in more detail.

**CASES**

In this section we present two cases of employing XP like practices in software production, a factory system and a communications application portfolio. Each case begins with a short description of the company and the system. Thereafter, the drivers of the development of the case system and the methods used are described. Then the software development process is delineated following Figure 1, an Extreme Programming Project, in Section 2. Finally, some views of the interviewees concerning the contemporary software process and its future are represented. The findings of the cases are presented and discussed in Section 5. The development organization, users and tools are described in more detail in Appendix 1 for the first case, and in Appendix 2 for the second case. A cross-comparison of the two cases will be found in Appendix 3.

**Case 1: Factory System**

*Case Company*

The case organization is the high barrier division of an international group in the flexible packaging industry. The high barrier division manufactures vacuum and modified atmosphere packaging for food industry - being the leading supplier of packaging films for fresh food in Europe. The division operates globally and has a comprehensive network of sales offices and agencies in over 30 countries employing 500 people. The share of exports exceeds 90 %, and turnover in 2001 was EUR 123 million. This particular factory has produced packaging films for almost 30 years. Plastic film extrusion in Finland started in the 1960s, and just a few years later the case organization was involved in the plastic films business. Manufacturing of co-extrusion films began at the beginning of the 1980s.

The information technology function of the division is located in Finland in the factory premises. In 1993 a separate information technology unit was merged with business development unit, which is also in charge of the development of production planning, logistics and procurement functions. The information technology function employs 12 people divided into three teams. The first team of six employees, the information systems development team, is in charge of the
factory system. The second team is responsible for management information systems, statistics and packaged software used in these activities, and the third team is responsible for hardware, networks and packaged software except those used in management information systems and statistics.

Factory System

The studied system, later called as the factory system, has been developed in-house. The first small application was developed in 1986. The factory system consists of three main parts: a sales system, mills or a production system and a business reporting system, the main applications being sales, production (i.e. factory), maintenance, purchasing and statistics. The factory system has about 500 users, that is, all employees of the division. The sales part has about 165 users in 10 countries and 30000 invoices are written per year. About 1000-1200 products (materials) are offered for about 1500 customers as 15000 customer products (width and length of the reel, printing). An online analytical processing (OLAP), multidimensional analysis and reporting software package is integrated into the factory system.

The factory system enables the factories and sales network to monitor production and deliveries in real time. The logistics services simplify the ordering process. The material requirements of the customers are monitored and predicted in real time. This means that storage needs are reduced and customers are assured of getting their material at all times. Delivery reliability is based on cyclical production and standardized, uniform grades delivered in a standard pack size.

Customers can consult the case organization on matters relating to the choice of film and the design of the package. The whole production is conducted to the customer needs, which is still exceptional in this field. The development organization, users and tools are described in more detail in Appendix 1.

Drivers

The motto of the factory system development is:” To know the business, to stay in it and to be the best in the business. The system is tailored to the business, not to the company.” Key driver of the development work is the continuously changing and increasingly complex business environment. In addition, the development perspective is clearly ‘bottom-up’, as user needs drive the continuous development of the system.

The key factor for the success of the development work is the domain knowledge of the team members. Each of them has a long experience in other companies of the group and in different jobs, as well as a deep understanding of the business. Expertise with the tools used is not seen as
equally important. The persons must be extrovert, speak the language of the users, be actively in contact with users and be easy to approach. Responsibility, initiative and desire as well as the ability to inform others are also seen as important characteristics.

*Architectural Spike, i.e. Methods and Standards*

The current development tools (see Appendix 1) were selected around 1986, when a decision was made to move over from minicomputers to microcomputers and client-server architecture. One designer was responsible for selecting new tools for this critical, 24/7 system. The first small application with the current tools was developed and taken into production in 1986, with an expert from the vendor participating in the development and training the first users who gradually took over the development work.

The original factory system with current tools was developed during 1986-1990 as a project following the waterfall development model. Because of the long and slow development phase, the system had gone out of date by the time it was finished. The contemporary working method was introduced in 1990, when the development of the current factory system began. Since then, the method has evolved and become more and more streamlined.

The working methods and standards are discussed and, if necessary, changed by developers weekly. Coding rules especially are strictly standardized from the number of empty rows between the program parts to the starting position of a certain row type. This enables common code and makes the reading of the code quicker.

Nowadays the method is used throughout the development lifecycle. No other development methods are used and no project work is done either, because of their slowness and inflexibility.

*User Stories, Release Planning and Spikes*

Requirements are actively collected from many sources. Business goals, objectives and structures are received from management as policy statements. Requests for technical changes are usually raised by the system administration. User requirements are actively collected, and daily communication between developers and end-users, both official and spontaneous, in the factory premises is extremely valuable. Now, in the production phase, user requirements are received from users through system help desk or as feedback given directly and informally to the developers. Help desk and training services are available in German and French in addition to Finnish and English. All the feedback is registered. The request for a change or a new requirement may also stem from the lack of a function in the system or the inability of the system
to serve a function. Also the need to reduce staff from a function or the shortage of employees in a function may give an initiative to system improvement.

Developers go through user feedback daily to find out and fix errors. These, and small improvements, are usually installed immediately. Requirements and feedback are also gathered up, and the management looks through this list regularly and decides on the future development. A certain part of the system will be renewed if there are plenty of negative comments concerning it. The number of users is often used as a decision criterion. Costs or time schedule have less value in decision making. For a major renewal, both a short term (1 month) work plan and a long term (6 months) iteration plan are drafted. All the designers work only part time with major renewals. A spike is programmed if necessary, to ensure the feasibility of the planned solution.

*Iteration, Refactoring, Testing, Releases and Pair Programming*

A new program is initiated first by coding a program skeleton with only a few basic functions. This semi-finished program is installed into the production environment to make sure that it will meet the basic requirements. At the beginning, the developer uses the skeleton parallel to the old one, if it exists, and collects experience of its operation. The skeleton is continuously changed and extended, that is, it will never be finished. Its development will be stopped for a while once a desired service level is attained. The developers know by experience and domain knowledge, when this level is reached.

A developer may make small and simple changes and error corrections directly into the production environment independently. If changes are needed in other parts of the system or in the database, they are made first in a test environment, a copy of the production environment, on the developer’s own microcomputer. When all the changes have been finished, they are installed into the production environment. It is the responsibility of the developer to make sure that changes possibly made by other developers directly into the production environment in the meanwhile stay in use and perform as expected. All changes are registered into a change log file.

A developer tests his own code himself. In addition, those two team members responsible for training and help desk will test larger changes before taking them into production.

All changes must be made and installed directly into the production environment of the system incrementally, because the factory operation depends entirely on this system and operates 24/7 shifts. A development phase will take from hours to a few days or sometimes a few weeks, but never months.
Pair programming is not used in the case company and the developers do not share work premises. Error and problem solving and spikes are done in pairs, if necessary.

**Documentation**

Documentation has been reduced into an absolute minimum and only the key, i.e. useful documents are drafted and maintained. Working methods, standards and coding rules are documented as an instruction sheet. Developers are responsible for database diagrams, entity-relationship diagrams and change log file. Program code is the most important document for the developers. Trainers are responsible for the user’s manual or system help. This manual is an Acrobat PDF-file consisting of instructions for managing special cases or functions each of them being at most one sheet long.

**Future**

This development method and the efficient development tools in use meet the needs of the constantly changing business best. They enable quick and continuous modification and evolution of the system. The development work is done in small incremental pieces, so nothing or very little is wasted and hardly anything useless is done.

The way of working is also inexpensive. The total information technology expenses, according to the company, are far below the average in the sector, because the system is built in-house and practically no software or license fees are paid or external work force needed.

The information systems development will be continued in this well-working manner. Every other method would require more bureaucracy and strict responsibilities. The present employees have deep knowledge of the business domain, enabling them to work independently and with low hierarchy. In addition, the employees argued that they would probably suffer from lack of motivation if some other working practices were used.

**Case 2: Communications Application Portfolio**

**Case Company**

The case organization is a Finnish corporate communications agency, which was founded in 1986. It is a part of an international network of advertising, marketing, communications and interactive agencies. The agency provides services ranging from communications research, strategic planning, crisis communications, public relations, public affairs, corporate communications, and investor relations to print publications. The customer companies come from diverse industries as well as from various governmental and municipal organizations. The agency
employs around 50 professionals, and in 2001 its turnover was EUR 6 million with a 15 % operating profit.

An increasing pressure to move and extend traditional communications to incorporate a digital presence and form led to the establishing of a digital communications unit in May 2001. The business strategy of the unit is to support and extend traditional communications and PR activities offered by the agency. The expected turnover of the unit during 2002 is approximately EUR 1 million with an operating profit of about 10 %. The unit employs 12 people divided into three teams of four: consulting, graphics and technology. The technology team is in charge of the application portfolio, user interface design, website production, updating and maintenance services and ASP-service (application service provision), which is the most common way to use the customer software.

*Communications Application Portfolio*

The communications application portfolio is a software toolkit developed in-house. The development work began in May 2001 and took about 18 months. Now the portfolio is in maintenance phase and used in customer software development. The portfolio consists of four applications: extranet, content management, monitoring application and crisis communications, which are described next.

The extranet is a low risk and low return application used only in project management for coordinating and facilitating communications between the company and its customers. The extranet has a supporting role, but it is essential for the success of customer projects. It needs only a little non-native code and customization.

The content management application provides a web-interface for the creation, management and maintenance of different forms of content. It is a communications solution and has often a critical function in customer’s activities. Some customization is required with each individual implementation.

The monitoring application is primarily used as a business intelligence tool to collect and store digital information. It acts as a search engine querying specified keywords from predefined websites and news groups. Matches are stored in its database and a summary of the findings is presented to the user. Customization is always required.

The crisis communications application is targeted at an extremely narrow niche audience. It is a collection of web-services to help administer and manage a crisis, from a communications point-of-view. It includes e.g. a digital version of the customer’s crisis manual, holding statements,
decision support trees, press contact information and crisis scenario planning information. It holds the greatest future potential, as currently the case company is the only provider of such an application. The user interface is standardized, but all the underlying services are customized. The development organization, users and tools are described in more detail in Appendix 2.

Drivers

The initiative for the communications application portfolio development came from the need to reuse and standardize the existing modules and programs. The main focus of the development is the consolidation of separate key software modules and individual applications, which have arisen from either individual development projects or as part of customer projects. The gains sought are: a common platform reducing development and maintenance costs and time, differentiation from competitors and flexible pricing mechanisms.

The key factor for the success of the development work is the experience and expertise of the team members with the tools and technology used. Expertise with the business is not equally important. Also the team spirit, the programmers being familiar with each other before starting to work together, is essential.

Architectural Spike, i.e. Methods and Standards

The technology philosophy is derived from the unit’s strategy. When the unit was founded, the key question regarding technology was not which one to use and how to implement it, but rather how to acquire skilled people that fit the organization. A collective decision among the original employees, two consultants and a programmer was made on the technologies and platforms (see Appendix 2). The main reason for selecting these tools was that the employees were familiar with them beforehand. No rigorous selection process was conducted. The methods, standards and working habits employed in software development evolved along with the portfolio development without any conscious decision.

The selected way of working is used in product development and maintenance phases as well as in customer implementations. The original communications application portfolio was not developed as a project. Customer implementations are always carried out as projects with a nominated project organization using a waterfall model with clear sequential phases.

No written standards or rules concerning the process or the code exist. Rather all developers have the freedom to decide about their own work. However, unwritten rules exist, and these have been adopted easily, because the programmers have a common background.
User Stories, Release Planning and Spikes

A requirement or need, either functional or technical, can come from employees or customers (50/50). Each customer has a nominated contact person, who receives the feedback. New technical requirements often evolve from technical development of the tools or from system administration. A quick situation analysis is performed to assess the feasibility of the requirement.

A programmer independently makes small improvements and error corrections. More complex and far-reaching decisions, like new functions, interfaces or integration needs are dealt with collectively. Also decisions concerning the implementation of these features as development investments or customer projects are made together, based usually on the number of potential customers and end-users. Spikes are seldom programmed.

Iteration, Refactoring, Testing, Releases and Pair Programming

The development process has four distinct phases: initial, planning, development and maintenance phases. The decision in the initial phase to proceed with a requirements specification leads to the planning phase. In the planning phase, the first project plan either leads to a document outlining the technical specifications of the new product or to the rejection of the project. For approved projects, a detailed second project plan for the development phase is drafted. The primary decision is the choice between implementing it as a customer project or as a separate product development project. After the development phase, a maintenance phase outlines the 12-month development roadmap. The phase continues with iterations and feedback from customers.

Each programmer codes and tests his own component or piece of program. Thereafter, internal users representing the customer test the system. External customers test the production version of the system before it is launched into production use. Iterations are continued until the application or customer software is acceptable. The code is also continuously refactored, and thoroughly documented.

Pair programming is not used systematically in the case company, although the programmers are sitting at the same table. Error and problem solving is done in pairs, if necessary.

Programmers update changes into the ASP customer software gradually. For customers using the software on their own computers, changes are occasionally distributed as an update package.

Documentation

The programmers draft all documentation concerning the application portfolio. Some technical documents are drafted in the planning phase and some systems or integration descriptions in the
development phase. Program code is the most important document for the programmers, and it must be documented in detail. Program size should be about 2000 - 4000 LOC. No change or problem log is drafted, or updated. However, document or version management system and a component library help to manage physical changes.

The training material and on-line help for end-users are drafted collectively. An implementation manual for main users of the applications is also drafted jointly.

Future
The communications application portfolio is sufficient for the contemporary business needs. Significant technological changes might cause the renewal of the portfolio. The personnel may be regarded as permanent in the team.

The total information technology expenses of the team have not been monitored, but the salaries form the main part. IT investments are probably rather low because of the use of mostly free open source software. No software or license fees are paid and no external personnel are used.

The information systems development will be continued accordingly at least as long as the existing personnel are employed. Every other method would require more bureaucracy and strict responsibilities and more personnel would be needed. The employees claimed that they would most probably face a downturn in work motivation if some other working practices were used.

FINDINGS AND DISCUSSION
In this study we compared the ISD processes of two case companies and their application of Extreme Programming (see Appendix 3). We had chosen one traditional industrial case and the other, which could be classified as a new economy case. Interestingly, in the more traditional case, the tools and techniques of XP had been employed for over ten years and they had been applied in quite a systematic fashion, though they had never made a deliberate decision to “use XP”. In the newer company the XP process had more or less emerged as a novel way of solving time and budget constraints. The developers were aware of XP practices, but did not choose to engage in it ”by the book”. This company, with a younger developer staff, had seen agile practices as a natural way of doing things, as they did not see the value of more bureaucratic methods.

Many essential features of XP can be found in the working methods of the case organizations as listed in Table 1. The table first lists Extreme Programming features slightly adapting the principles and values of XP according to Beck (1999). For each XP feature we identify whether it
is used in one of the cases. Furthermore, we identify references from vintage ISD literature to support our claim that these techniques have been in use for a long time.

Findings from the Cases

Table 1 Findings from the cases

<table>
<thead>
<tr>
<th>Extreme Programming features</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Related ISD literature</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-user participation</td>
<td>+</td>
<td>-</td>
<td>User centered design (Ehn, 1988; Andersen et al., 1990; Grudin, 1991; Greenbaum &amp; Kyng, 1991; Clemont &amp; Besselaar, 1993; McKeen et al., 1994; Smart &amp; Whiting, 2001)</td>
</tr>
<tr>
<td>Standardization of development tools</td>
<td>+</td>
<td>+</td>
<td>Design to tools (McConnell, 1996)</td>
</tr>
<tr>
<td>Clear decision rules, roles and responsibilities</td>
<td>+</td>
<td>-</td>
<td>Professional work practices (Andersen et al., 1990)</td>
</tr>
<tr>
<td>Distinct phases</td>
<td>+</td>
<td>+</td>
<td>Phased ISD (Hirschheim et al., 2003; Sommerville, 2001)</td>
</tr>
<tr>
<td>Iterative development and implementation in short cycles</td>
<td>+</td>
<td>+</td>
<td>Incremental prototyping (Boehm, 1988; Luqi &amp; Zyda, 1990; Iivari, 1990a; Boehm et al., 1998)</td>
</tr>
<tr>
<td>Testing</td>
<td>+</td>
<td>+</td>
<td>(Evans, 1984)</td>
</tr>
<tr>
<td>Documentation in code</td>
<td>+</td>
<td>+</td>
<td>Literate programming (Knuth, 1984)</td>
</tr>
<tr>
<td>Commonly owned program code</td>
<td>+</td>
<td>+</td>
<td>No secret code (Shabe, Peck, &amp; Hickey, 1977)</td>
</tr>
<tr>
<td>Specialization on a specific application or on the database or system structure</td>
<td>+</td>
<td>-</td>
<td>Design to tools (McConnell, 1996)</td>
</tr>
<tr>
<td>Pair programming</td>
<td>-</td>
<td>-</td>
<td>Fred Brooks in the 1950s (Williams &amp; Kessler, 2002)</td>
</tr>
<tr>
<td>Programmer morale</td>
<td>+</td>
<td>+</td>
<td>(Brooks, 1975)</td>
</tr>
<tr>
<td>Continuous feedback</td>
<td>+</td>
<td>+</td>
<td>(Boehm, 1988)</td>
</tr>
<tr>
<td>Project work</td>
<td>-</td>
<td>+</td>
<td>(Paulk et al., 1993)</td>
</tr>
<tr>
<td>Other methods used (waterfall)</td>
<td>-</td>
<td>+</td>
<td>(Brooks, 1975; Hirschheim et al., 2003; Sommerville, 2001)</td>
</tr>
</tbody>
</table>

As can be observed in Table 1, both case companies apply XP techniques, apart from pair programming, extensively. In Case 1, XP techniques were used systematically throughout the development lifecycle. The method is a result of systematic evolution from stricter methodological practices, which were found too restricting and slow. No other development methods were used in Case 1. Project work was also perceived as too slow and inflexible, and nowadays development work is not managed as projects. In Case 2, the programmers utilized the application portfolio in customer projects, so in this aspect the method resembles end-user programming. The key end-users, however, are the customers, and customer implementations follow waterfall model and are organized as projects.
Way of Working

In the first case the way of working was adopted as early as in 1990 and it has evolved and streamlined gradually and systematically. There is a great resemblance between XP and the development method used in the 1960s and 1970s, when systems were tailored for each organization’s own use by IT personnel of its own. At those times, like in the case organization, the basis for future development was both the requirements of business and user needs. Requirements of the management were not a separate matter but they were satisfied through the requirements of the business. In the second case, the information systems development was a new activity in the organization, and the tools and the way of working were introduced and implemented at the beginning.

In both companies the developers liked this way of working and we have anecdotic evidence that the internal and external customers were also satisfied with the results. However, it should be noted that both companies exhibit a key problem of all radically new “methods” - they are quite person dependent. In the first case we found that XP works best with experienced developers that know their domain and their development tools. In the second case with less experienced developers we found that the XP development model had more or less emerged instead of a planned approach. XP in this latter fashion closely resembles Capability Maturity Model’s (CMM) level 0, i.e. chaos.

Development Organization and Personnel

In Case 1, information technology unit is part of the business development and this is crucial for the success of the way of working. On the other hand, the team’s physical proximity to the users helps to maintain the knowledge of the business and of user needs and reduces the dependency on individual developers.

In the first case the domain knowledge of the team members as well as their excellent communication skills were found extremely important. Without these kinds of persons, the chosen approach would probably have little possibilities to succeed. This was clear also in the second case, where the expertise of the team members with the tools and technology used as well as their own community were extremely important to enable this way of working. The development method was highly dependent on individual programmers, but therefore it suited perfectly the organizational culture of the firm. This finding is consistent with findings about the so-called “Internet speed” development (Baskerville, Ramesh, Levine, Pries-Heje, & Slaughter, 2003).
Continuous feedback, both official and unofficial, was one of the key factors of success. In Case 1, very little feedback on the general success of the system is received from current users. Generally positive feedback is received from users who have left the organization or from newcomers, who have the possibility to compare this system with others. There is no change resistance, and users propose changes and improvements to the system actively. They also understand that everything is not reasonable to fulfill, and this fact keeps the method working.

The tools employed facilitated the use of XP in both cases. They supported fast delivery and easy modification of prototypes. This closely resembles the ideas put forth by early advocates of incremental prototyping (Luqi & Zyda, 1990) and user centered design (Ehn, 1988), and furthermore design to tools (McConnell, 1996).

**Comparison with Other Cases and Agile Methods in General**

In this paper we took a different approach from other recent case studies of XP (Abrahamsson, 2003; Abrahamsson et al., 2002; Back et al., 2002; Elssamadisy & Schalliol, 2002; Reifer, 2002; Salo & Abrahamsson, 2004), which concentrated on the planned and systematic adoption of XP in laboratory cases or in pilot projects. See Salo et al. (2004) for empirical studies on agile methods categorized into surveys, case studies and examples. Visconti et al. (2004) provide a list of empirical studies of successes of agile methods after unsuccessful use of traditional methods. Boehm and Turner (2003) provide a detailed account of several cases of employing agile practices.

We selected cases where the methods had evolved into an XP like way of working although it was not intentionally and consciously selected as a method. Aoyama reports evolution and experiences in a telecommunications software family development over a time period of ten years (Aoyama, 1998), very similar to our first case. Likewise, Vanhanen et al. (2003) report the evolution of agile practices in a Finnish telecom industry in three projects, one of which has a life span of over fifteen years, again very similar to our first case. In all three projects, agile practices were used (evolved or intentionally adopted) because they represented a natural and useful way to develop software. They found that the longest running project applied most widely and systematically agile practices, also similar to our findings. Yet another research comparable to our first case is presented by Murru et al. (2003) who report the experiences of two sequential projects: first adopting XP and then adapting it for the second project based on the experiences of the first project.
Opinions differ significantly on the relationship between traditional and agile methods. Some researchers argue that agile methods present an alternative to process-centered approaches (Baskerville et al., 2003; Boehm, 2002; Murru et al., 2003) while others see agile and process-centered methods as complementary (Boehm & Turner, 2003; Paulk, 2001). A third group of researchers see agile processes as a step further in software process improvement regarded from the CMMI point of view (Kähkönen & Abrahamsson, 2004; Turner, 2002). Increasingly both researchers and practitioners see agile and traditional plan-driven methods as complementary so that different development situations are best supported by different development methods (Boehm & Turner, 2003; Howard, 2003; Känsälä, 2004). Boehm and Turner (2003) propose a multidimensional model for selecting the appropriate software development method according to the type of the project.

To sum up, there are about a dozen software development approaches that are classified / regarded as agile. Common to all agile methods is the emphasis on the output of the software development process, working software and maximizing its value for the customer. All agile methods, including XP, have their strengths and weaknesses, and different methods are suitable for different business and software development situations. The field is continuously developed further by the academics (Nawrocki, Jasinski, Walter, & Wojciechowski, 2002; Visconti & Cook, 2004). Agile methods like all software development methods are also continuously evolving through adaptation by practitioners in daily use (Wynkoop & Russo, 1995). Our results corroborate the above results by other researchers. The two cases of this research illustrate how practitioners adapt and apply methods. The research provides reasons why practitioners turn to agile methods. It also indicates that the method selection discussion should not be limited to which method is better than the other but instead the discussion should focus on the drivers, constraints and enablers that affect the selection of the method.

**Old Wine in New Bottles**

As hypothesized in the beginning, we found that the features of XP can be related back to the ISD literature as shown in Table 1. Even though we have argued that the features of XP can be traced back to traditional ISD methods we do not want de-emphasize the positive aspects of XP usage. For example, in the first case there is no change resistance, and users propose changes and improvements to the system actively, so there is true user participation. It was also claimed that the end-users understand sufficiently well that everything is not reasonable to fulfill, and this fact keeps the method working. In the second case, it was argued to suite perfectly to the young organizational culture and hence strengthening the team spirit of the development team. All in all,
we can conclude that it is well and good that a fresh name gives a lease of life for good practices. However, the authors of XP books and articles should notice that many of their ideas can be traced back to Nygåård’s ideas (Dahl & Nygaard, 1966).

**CONCLUSIONS**

In this study we explored the concept of Extreme Programming as a new approach to IS development. We used a qualitative case study approach as recommended by Klein and Myers (1999) and Wynn (2001) for studying social processes and trying to understand users in the local level. In the case analysis we adapted the principles of interpretive case studies presented by Walsham (1995). We found support for our claim that XP is more of a new bag of old tricks than a totally new way of doing things. We also found out that XP is actually a new name for an old way of working when developing a tailored system in-house. It formalizes several habits that appear naturally in a setting like our first case: close customer involvement, short release cycles, cyclical development and fast response to change requests. In other words, it combines the best practices of Scandinavian approach (Bjerkenes & Bratteteig, 1995; Grudin, 1991) in general and user centered design in particular into a package that is both acceptable and applicable for developers. The so-called Scandinavian approach to information systems development has been advocating user centeredness and professional work practices since the mid-eighties, and its roots can be traced back to the origins of object-oriented development (Dahl & Nygaard, 1966). However, it seems that these ideas are easier to accept when they come from within the software development community and they have a name that connects them with heroic programming efforts.

It is somewhat disturbing that these practices rely heavily on people and seem to be at times an excuse for not using more refined approaches. We maintain that XP can be useful for small teams of domain experts, who are able to communicate well with customers and are very good designers and implementers. One could argue that XP canonizes, and to a certain degree formalizes, the good practices used by these exceptional individuals and teams, which is fine. However, these people can exhibit high productivity in almost any development setting, which is not overly constrained by bureaucracy. The real test of XP is then, whether mere mortals or “normal” developers can employ it as successfully.

In the future, we would like to see how XP can be used in larger scale settings with external customers, either consumers or users in other units within the same company, possibly located in other countries. These would put XP in test with more complex requirements gathering/elicitation phases and maintenance of systems through release versions. It would also be interesting to study
if XP or some other agile method would be easy enough to be adopted in more traditionally organized IS departments. XP might also be a useful method for organizations with only few IS specialists in managing their ISD projects with external consultants and vendors.

ACKNOWLEDGMENT

This research was supported in part by the Academy of Finland (project 674917), the Jenny and Antti Wihuri Foundation, and the Foundation for Economic Education. We wish to thank the contact persons and interviewees, Tapani Isotalo, Sami Relander, Markku Vihervä and Juha Virkki, in the case companies for cooperation. We also thank the anonymous referees for their valuable comments.

REFERENCES


Hilkka Merisalo-Rantanen is a researcher and PhD student of Information Systems Science at the Helsinki School of Economics. Her research interests include information systems development methods, stakeholder and end-user participation throughout the information system lifecycle, and multi-customer – multi-vendor information system development projects. She has worked over 20 years in various tasks of information systems development, consultancy and project management in leading Finnish companies. She has published in the IEEE Transactions on Professional Communication (forthcoming).

Tuure Tuunanen, M.Sc. (econ) (Helsinki, 2001), is finalizing his PhD degree in information systems science at the Helsinki School of Economics. His doctoral dissertation focuses on requirements elicitation methods to support idea generation and software development in wide audience end-user environments, such as JAVA applications for mobile phones and digital TV terminals. He is currently a research fellow at Graduate School of Electronic Business and Software Industry. His research has been published in Journal of Management Information Systems, Information & Management, and Journal of Information Technology Theory and Application. In addition, his work has appeared in conference proceedings, such as eCOMO, ISD, HICSS, RE and WITS.

Matti Rossi received his Ph.D. degree in Business Administration from the University of Jyväskylä in 1998. He is an acting professor of information systems and director of the electronic business program for professionals (Muuntokoulutus) at Helsinki School of Economics. He has worked as research fellow at Erasmus University Rotterdam and as a visiting assistant professor at Georgia State University, Atlanta. He has been the principal investigator in several major research projects funded by the technological development center of Finland and Academy of Finland. His research papers have appeared in journals such as Journal of AIS, Information and Organization, Information and Management and Information Systems, and over twenty of them have appeared in conferences such as ICIS, HICSS and CAiSE.
### APPENDIX 1

**Development environment of CASE Factory system**

| **Users** | The factory system is used in factories as well as in sales offices and agencies of the high barrier division throughout Europe. Practically all, total 500, employees are end-users of the system. Actually every user or user group has an own tailored system. One user profile consists of at most 3-4 users/shift. Users receive their work tasks from the system automatically at the sign-in. Management has read-only rights into the system. |
| **Tools** | The system is developed using an application development tool AdWISE (Western Systems Oy, [http://www.western.fi/](http://www.western.fi/), 13.5.2003) and MDBS IV database (Micro Data Base Systems Inc., [http://www.mdbs.com/](http://www.mdbs.com/), 13.5.2003). AdWISE is a three tier (client, application server, database server), modular architecture consisting of a fourth generation application description language W, W compiler and W interpreter. AdWISE supports prototyping and end-user programming, and makes systems efficient, scalable and platform independent. LAN/WAN is used only for data traffic. MDBS IV is an efficient, reliable, fault-tolerant navigational database system used in mission-critical real-time applications. With these efficient tools, a standard portable computer or PC is sufficient for developing and running the system and production database. The execution environment is usually small enough to enable the use of for instance diskless workstations, mobile phones and PDA’s as clients. In addition, Cognos PowerPlay software package (Cognos Inc., [http://www.cognos.com/](http://www.cognos.com/), 13.5.2003) is integrated to the system for online analytical processing (OLAP), multidimensional analysis and reporting. |
| **Team** | The development team consists of 6 persons. The key developers have been in the organization since 1990. The number of developers has gradually increased in 1995 and 2000 the total number now being 4. All developers have worked earlier in other units of the group and in different jobs, so they have a wide experience and total view of the activities in the group and the division. It takes about 6-12 months from a new employee to become acquainted with the business. In addition to developers there are two persons in the team who are in charge of the user help desk, training and testing. The business development manager and the IT manager responsible for online analytical processing (OLAP), multidimensional analysis and reporting also participate actively into the development work. Every developer is familiar with the entire factory system and all the code is mutual. In addition, developers are specialized. One developer is in charge of the sales and statistics applications, one of the production applications and one of the maintenance and procurement applications. One designer is responsible for the database, the system structure and the working methods with one of the three other designers participating actively into these tasks. |
### APPENDIX 2

Development environment of CASE Communications application portfolio

| **Users** | The four programmers use the applications as tools in developing customer software. The applications are used in about 50 customer implementations the total amount of end-users being about 300-350. About 75% of the customers use the extranet, over 50% use the content management, and only a few customers use the other two applications. Per customer, the extranet has 10-50, the content management 1-10, and the other two applications 1-5 end-users. |
| **Tools** | The application portfolio is built in a LAMP-environment around a common application server platform Midgard (Midgard Project, [http://midgard-project.org/](http://midgard-project.org/), 13.5.2003), which is open source framework for information management solutions. LAMP originates from Linux operating system, Apache (web server), MySQL (database management system) and PHP (programming and scripting language) components. Some additional code is made using Perl, C++ and Java. |
| **Team** | The team consists of 4 persons. Three software engineers, familiar with each other from University and still in the middle of their occupational studies, started in 2001. All programmers have profound technological knowledge, but little experience of the business. One programmer has left the company and has been replaced with another, also an acquaintance from school with around 2 months of apprenticeship. A senior consultant makes up the remainder of the team. The unit manager also participates actively into the development work. Management control over the unit is virtually non-existent. The unit functions like a miniature open-source software development community with main reward system being acknowledgement and approval from peers. All code is mutual, but there is one specialist for every application. Everyone is responsible for customer support and other activities of the team. All programmers are located in the same room sitting by the same table, which makes the communication continuous, informal and easy. Therefore, the team is very much inline with the overall philosophy of the case company, which is to be a relatively small, nimble and efficient that can quickly adjust to changes. |
APPENDIX 3

Cross-comparison of the cases

<table>
<thead>
<tr>
<th>Topic</th>
<th>Case 1</th>
<th>Case 2</th>
</tr>
</thead>
</table>
| Case company   | - A manufacturing division of an international group, founded about 30 years ago  
                  - 500 employees in Europe, 360 in Finland  
                  - A separate IT unit since 1990  
                  - IT part of business development unit since 1993, 12 employees  
                  - The operational systems team near users both organizationally and physically | - A PR agency belonging to an international network of agencies, founded in 1986  
                  - 50 employees  
                  - Digital communications unit founded in 2001, 12 employees  
                  - The technology team near technology and other team members |
| System         | - The operational system called as the factory system is made in-house  
                  - Strategic and critical, 24 hours a day 7 days a week | - The application portfolio is developed in-house  
                  - Strategic, not critical |
| Change         | - Continuous and rapid, internal and external in business, system, process, working habits, standards, ownership | - Stable, technology |
| Driver         | - Business driven, not only customer driven approach  
                  - Bottom up=user driven, not only management driven approach | - Business driven approach  
                  - Technology as enabler of new business possibilities |
| Methods        | - XP, evolutionary prototyping  
                  - No other ISD methods  
                  - No project work | - XP, waterfall, end-user programming  
                  - Customer implementations as projects |
| Users          | - 500 internal end-users | - 4 internal users: 3 internal programmers and a consultant  
                  - 300-350 external end-users |
| Tools          | - adWISE application development tool  
                  - MDBS IV navigational database  
                  - Cognos PowerPlay for online analytical processing (OLAP), multidimensional analysis and reporting | - Open source and LAMP  
                  - Midgard |
| Team           | - 6 persons, experienced both in business and in technology and methods  
                  - Specific roles and responsibilities | - 4 persons, experienced in technology  
                  - No specific roles and responsibilities, but one specialist for each application |
| Requirements   | - Business, users, system administration | - Business, technology, customers |
| Decision making| - Individual developers daily and independently on errors and small changes  
                  - Managers (3 persons) together on larger development needs | - Individual programmers daily and independently on errors and small changes, no clear responsibilities  
                  - Manager and/or consultant consulted on larger needs and on decisions on customer or development project |
| Process        | - Iterative short cycle process like XP  
                  - Resembles XP, but was started in 1990  
                  - Resembles also evolutionary prototyping  
                  - No pair programming  
                  - Like 1960s – 1970s | - Iterative short cycle process like XP  
                  - Resembles XP in some parts, but more like end-user programming or streamlined waterfall  
                  - No pair programming |