Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 79 (2006) 1295–1311
The Journal of Systems and Software
www.elsevier.com/locate/jss

How agile are industrial software development practices?

Christina Hansson [a,*], Yvonne Dittrich [a,b], Björn Gustafsson [a], Stefan Zarnak [a]

[a] Blekinge Institute of Technology, Department of Technology, Box 520, 372 25 Ronneby, Sweden
[b] IT University of Copenhagen, Design and Use of IT Rued Langgaards Vej 7, DK-2300 København S, Denmark

Received 31 October 2005; received in revised form 23 November 2005; accepted 12 December 2005
Available online 7 February 2006

Abstract

Representatives from the agile development movement claim that agile ways of developing software are more fitting to what is actually needed in industrial software development. If this is so, successful industrial software development should already exhibit agile characteristics. This article therefore aims to examine whether that is the case. It presents an analysis of interviews with software developers from five different companies. We asked about concrete projects, both about the project models and the methods used, but also about the real situation in their daily work. Based on the interviews, we describe and then analyze their development practices. The analysis shows that the software providers we interviewed have more agile practices than they might themselves be aware of. However, plans and more formal development models also are well established. The conclusions answer the question posed in the title: It all depends! It depends on which of the different principles you take to judge agility. And it depends on the characteristics not only of the company but also of the individual project.

Keywords: Agile software development; Industrial software companies; Qualitative research methods

1. Introduction

Many representatives of the agile development movement claim that agile ways of developing software are more fitting to what is actually needed in industrial software development (Beck, 2000; Cockburn, 2002). If that is the case, then successful industrial software development ought to exhibit agile characteristics. This research hypothesis was the starting point for the empirical study that is analyzed and evaluated in this article. But how can we 'measure' the agility of development processes, when the practitioners themselves are barely familiar with the term?

Software engineering research and education has for the last 30 years emphasized control over the development process: communication between developers and users should stop when the requirements specification was signed; detailed control over the individual phases should ensure high quality software, independent of the individual developer. Lead-time and budget accuracy seemed to be the most important criteria for software quality. One of the most extreme and, according to the software engineering community, most influential, and award winning articles in this tradition is Osterweil's 'Software processes are Software too' (1987), although other researchers acknowledged that the ideal was impossible to reach (Clement and Parnas, 1986). This has lead to a situation where practices that did not conform to the ideal were not documented, even though they were considered to be an important contribution to the success of the software development. Use-oriented software development especially, requiring the deferring of design decisions until users can be provided with hands-on experience of the software under development, had problems to argue for more flexibility to take in user feedback late in the process (Nørbjerg and Kraft, 2002). Perhaps the most important contribution of the agile

* Corresponding author. Tel.: +46 457 385862; fax: +46 457 27125.
E-mail addresses: christina.hansson@bth.se (C. Hansson), ydi@itu.dk (Y. Dittrich), bjorn@eventus.se (B. Gustafsson), zarnak@tiscali.se (S. Zarnak).

0164-1212/$ - see front matter © 2006 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2005.12.020

community is the introduction of terms to describe software development practices that do not fit in with the above described ideal of software development, in ways that acknowledge the contribution of these practices rather than criticizing them. The agile movement built a new set of distinctions and vocabularies to describe software development (Cockburn, 2002).

With the term 'practice' we describe a common way of acting, acknowledged by a community as the correct way to do things. It can be taught to newcomers by letting them take part in this practice as an apprentice (Wenger, 1998). A community maintains the common practice through more or less formal 'articulation work' (Gerson and Star, 1986) which is also the means to handle exceptional situations. Ad-hoc behavior—always necessary to handle exceptions and to maintain the 'normal' (Suchman, 1983)—is as such only perceivable by its deviation from both the formalized rules and the established practice. The plural 'practices' is thus used to relate to the different well established ways that different development companies have cultivated to cope with requirements posed by customers, business environment, market section, and software products.

In order to capture concrete practices, rather than a polished record, we focused on both the methods used and on concrete projects.

We interviewed representatives of five companies. Analysis of the interviews shows that the companies deploy different process models depending on the size of the company and the project, who the customer is, and the type of software to be developed. Co-operation and frequent face-to-face communication seem to have a central role, especially within the smaller development teams. Companies working both with major redevelopment and with regular updates adapt their one project model to the different circumstances. Projects for regular updates, in particular, develop agile characteristics. All companies are open for changes in requirements even late in the project, if they are considered to be important enough. The majority of them have long-term relationships with their customers and collect customer feedback for use in update and redevelopment projects. The level of documentation differs, but in general documentation is seen as an important though time-consuming activity. All in all, the software providers we interviewed have more agile practices than they might themselves be aware of. However, plans and more formal development models are also well established in today's industrial practice. The software providers we interviewed had good reasons for their design and adaptation of the respective process model, and made use of functions such as the help desk and customer support to provide input to their update and development projects.

The focus placed on projects by software engineering research may have blinded the research community, both for connections between different projects and for activities that provide input to updating and redevelopment projects. The main conclusion to draw from the research reported here is the importance of taking industrial practice seriously. The interviewed companies quite expertly combine agile and traditional practices and adjust their practices according to the situation at hand. Through more empirical research we might learn when and how to apply more plan driven or more agile development methods.

2. Related research in the area of agile software development and flexible development practices

The term 'Agile development' was coined in 2001, at a meeting in a ski resort in North America, when a group of people involved in finding, testing and defining new software development methods came up with an 'Agile Manifesto' (Agile Manifesto, 2001): The agile manifesto was developed in response to the emphasis placed by mainstream software development research on planning, control, and efficiency. A number of methods or approaches are classified under the category of agile methods. They implement evolutionary and flexible software processes that allow reaction to changes in customer requirements and rely on communication and cooperation, rather than written documents, to communicate requirements and design ideas.

The promoters of agile software development often advertise their approaches as a practitioners' answer to the methods promoted by software engineering research. Many of the agile methods have been developed based on practical experience and have been used and consolidated in industrial contexts. The agile movement claims to formulate what is common sense among software development practitioners, though it is not backed up by broad research. Contributions to conferences on extreme programming and agile development mainly summarize experiences from introducing agile development or extreme programming practices in different contexts. According to Abrahamsson et al. (2003) more empirical research is needed, for example to examine the ease of use or possible negative implications of different methods and practices for different sizes and kind of companies.

Exceptions are a group of publications addressing conditions regarding the development of Internet applications (e.g., Baskerville et al., 2003). Based on 10 case studies the authors analyze the development practices in companies developing Internet applications. Competition and short innovation cycles force these companies to apply iterative development, to begin development based on vague requirements and to change the requirements due to e.g. competitors' releases. They conclude that the observed practices implement the principles of agile development although tempered by the observation of what the authors call more traditional principles, like design for reuse, low coupling of components or continuous improvement (Baskerville et al., 2003, p. 76).

As part of our previous research, we have reported examples that support our hypothesis that industrial practice is often more flexible than is admitted by the traditional software engineering research, and that for good reasons. For a small provider of booking systems for

municipalities and sports facilities, developing the ability to react to customer and user feedback became a business advantage (Hansson et al., 2004). Software development at a telecommunication provider in Sweden deployed a software project model that in practice left enough flexibility to accommodate tight user/developer co-operation and evolutionary development. A steering group for each project helped to deal with situations when the dynamics threatened to extend the project's scope, lead-time or budget (Dittrich and Lindeberg, 2004).

Based on the above-cited publications one can claim that software development in industrial practice is probably more diverse than being either strictly agile or traditional. This is supported by an article written by Barry Boehm, where different software development practices and methods are placed on a 'planning' spectrum, where 'hacking' and 'inch pebble ironbound contract' occupy the extremes (Boehm, 2002). The article argues that the suitability of more or less agility or planning depends on the context of development as well as on the kind and size of the software to be developed. However, though certainly based on extensive experience of different software development practices, this argumentation is not founded in empirical research.

With our study we aim to contribute to a more differentiated picture of the agility respectively the control orientation of industrial practices. We give a detailed view of different development practices, sometimes even of different methods used within the same company. Though none of the companies had a customer present under development, most of them had developed contact with key customers that continued beyond single projects. After the analysis of our interviews we will discuss the challenges that our findings provide for software engineering research.

3. Research methods

3.1. Data collection

The article contains analysis of interviews with five companies. Four of them were randomly chosen from the telephone book of two cities in southern Sweden. The fifth we had come in contact with through a parallel research project on e-government and citizen participation. We do not know how representative this convenience sample is, regarding either the Swedish or the global software industry. Economically, Southern Sweden can be seen as representative of many Western European and North American regions. The Southern region especially is highly developed, with a well-educated workforce. IT and software development play an important role in the local economy and this is backed up by higher education and research institutes in the area. Besides a number of small and medium sized enterprises, even national and international IT companies are represented in the region. Our selection provided us with a sample covering a wide spectrum of size (the interviewed companies had between seven and 900 employees), kind of software (the companies developed back office and production systems for telecommunication providers, web based and other applications for municipalities, and financial systems) and contract model (the interviews concerned projects that ranged from custom development, regular updates, maintenance, and tailoring based on a set of modules). See Table 1 for a summary of the company and project characteristics. This table also categorizes the projects, according to the project characteristics that Glass (2001) proposes as indicators for whether or not agile methods are suitable. Innovativeness can be understood in two different ways. We understood it in terms of innovative uses of technology rather than in terms of technological innovation. Criticality was mainly evaluated based on the application domain.

We chose to use semi-structured interviews (Robson, 2002, p. 278) addressing the companies' software development methods and project models as well as the practices in and around concrete projects our interview partners have been involved in. These semi-structured interviews helped us to capture the companies' practices and their degree of agility, without specifically asking about them. The interview guidelines were developed on the basis of literature research on agile development and contained questions on how the companies actually work in specific projects instead of how they should work according to possible documented software processes. Most of the companies were interviewed twice. The first round of interviews (documented in Saarnak and Gustafsson, 2003) resulted in a set of additional questions regarding e.g. different kinds of project within the same company or the different channels for collecting customer and user feedback and improvement proposals. The clarification of these points was the subject of a second round of interviews.

Table 1
Company and project characteristics

	Xodus	Mandus	Kuling	Cellex	Yampus
Application domain	Accounting	Small systems often web-based	Telecom. production and back office	Standard and custom business systems	Web applications
Project size (developers)	9 over 36 months medium	4–5 over 4 months low	Ca. 100 over 4 months high	Ca. 30 for 4 months medium	3–5 over 6 months low
Criticality	Medium	Low	Medium	Medium	Low
Innovativeness	Low	Medium	Medium	Low	Medium

At Xodus the interviewee was a project coordinator. The interviewee at Mandus had the role of project manager. He acted as an intermediary between the project group (customer) and the company. He also worked extensively with development. At Kuling the interviewee worked as a project manager. She was also responsible for gathering new requirements from the customer and estimating time and costs for these. Besides this, the interviewee handled all contacts and communication between the customer and the company. At Cellex the interviewee had the role of product responsible, and was responsible for the development of a real estate product. She also played the customer role within the company and coordinated the project. The interviewee at Yampus was project leader but he had also the initial and continuous contact with customers.The interviewees at all companies had a management or coordination function. This ensured that they had an overview of the development process.

All interviews were conducted by two of the authors at each of the companies' premises. The interviews lasted about 1 h each, and were recorded, transcribed, and categorized based on the interview guidelines. These categories were then divided into subcategories in order enable a more detailed analysis of the field material. Data was processed by all of the authors in order to counter individual biases. The results from the interviews were validated by referring to and cross checking with the interviewees. Multi-perspective analysis, member checking and categorization are means of triangulation to assure the validity of qualitative research results (Robson, 2002).

The detailed presentation of the interview results given below is in order to allow the reader to follow or contest our analysis and the conclusions we draw from our research.

3.2. Analysis scheme

The agile manifesto affirms four values for software development. 'We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value: (1) Individuals and interactions over processes and tools. (2) Working software over comprehensive documentation. (3) Customer collaboration over contract negotiation. (4) Responding to change over following a plan.' (Agile Manifesto) These values are based on experience and reflection on that experience. The four values are followed by twelve principles that characterize an agile process.

We see compliance with these four values as one way to evaluate/analyze whether industrial software processes can be seen as agile and if so, to what extent. Our analysis is therefore organized around the values. To categorize our qualitative findings we used a five level scale: the company's practice more or less implements the principle (++), it implements the principle rather than implementing the traditional counterpart (+), it is neutral in this respect (O), it rather implements traditional values (−), it implements the traditional counterpart of this value fully (−−).

3.3. Individuals and interaction over processes and tools

Communication and cooperation within the development team are considered to be fundamental and necessary in agile software development. Cockburn and Highsmith (2001) states that good communication and interaction within a development team signify that the employees can operate at a higher level than when using their individual skills. Therefore it is worth enhancing both individual competencies and collaborative skills. Furthermore, Cockburn (2000) argues that people sitting near each other with frequent, easy contact will develop software more easily. Boehm (1981) states that the quality of the programmers and the team is one of the most influential factors in constructing successful software and regards formal processes and tools as less important. One of the basic arguments for agile methods is the increased efficiency of people who work together. Frequent interaction between individuals compensates for less documentation (Highsmith, 2002). Discussion of new ideas based on flaws in previous projects improves future projects. A good process will not repair a bad team, but a closely united team can work with a bad process. It is not enough to be a talented programmer if the person cannot cooperate and communicate (Martin, 2002). A traditional document-driven development process places emphasis instead on the process. Extensive planning, codified processes and rigorous reuse should make development an efficient and predictable activity (Boehm, 2002). The waterfall model (Royce, 1970) emphasizes development through sequential phases with predefined documents, milestones and reviews after each phase, and is thus a prescriptive model.

The emphasis given to and organization of communication was used as an indicator of the agility of the reported practices.

3.4. Working software over comprehensive documentation

The agile manifesto challenges the traditional school of software development that emphasizes the production of different documents and takes the finalizing of documents as the main measure of progress during the waterfall-like development. According to the agile development idea, the working system is the best way to judge how much the team has achieved. If documents are not up to date, they are useless, so the agile recommendation is to produce no documents unless the need for them is immediate and significant. The team can rely on the tacit knowledge of its members, rather than writing the knowledge down in necessarily incomplete documents and plans (Boehm, 2002). This can however be discussed; Glass (2001) argues that more emphasis must be put on the maintenance documentation, especially in highly evolutionary development.

There, emphasizing working software over documentation would counteract iterative and evolutionary development.

Another way to transfer knowledge to, for example, new project members is to work together with them and make them part of the team through training and discussions instead of reading documents (Cockburn, 2002). Here Cockburn refers to, for example, Peter Naur's article on 'Programming as Theory Building' (Naur, 1985).

We used the companies' attitude towards documentation and its purpose as the second indicator.

3.5. Customer collaboration over contract negotiation

The agile manifesto emphasizes that software cannot be developed successfully without involving customers and/or users. The customer, or even more likely the user, knows the needs and requirements for the new software and should therefore take an active part in the development process. In order to develop proper software it is therefore important to merge different experiences and expertise (Highsmith, 2002). Feedback must be given on a regular and frequent basis. A contract should describe and stipulate how the development team and the customer/user will work together. A contract that only specifies the technical requirements, time-line and costs is flawed (Martin, 2002). In opposition to that, Glass (2001) argues that having customers around at the beginning and the end of a project is the most efficient way to involve customers. He states that having customers on site through all stages in a project is a waste of the customer's time.

In a traditional software engineering paradigm the requirements are part of the contract and each update requires a re-estimation and renegotiation of the contract.

Our interviews provide an even more differentiated picture of how users and customers are involved in development and can contribute to the software under development: The support function, for example, provides an important channel for user and customer involvement, especially when developing and updating standard systems. After recognizing the importance of the support function we asked more explicit questions concerning these activities that are normally not considered part of software development. So we included a wider scope of involving users in software development when evaluating the companies' practices according to this category.

3.6. Responding to change over following a plan

According to the agile movement, plans must be flexible, allowing response to changes in business and in technology. Building a detailed plan for the next few weeks is useful; having rough plans for the next few months and only vague ideas for the future is a good strategy. The customer will probably alter the requirements once they test the system (Williams and Cockburn, 2003). New requirements will be uncovered during the development process or previous requirements will be found to be unnecessary. The plan should therefore be renegotiated and reprioritized after each time-slot, to keep it up to date. An outdated plan is not useful. Many developers who favor the traditional way of developing software criticize the way of for example over-responding to changes and also criticize the lack of documentation and planning in agile methods. However, agile methods emphasize a fair amount of planning when compared to unplanned and undisciplined developing. Although the focus is more on the planning process than on the resulting documentation, agile methods often appear less plan-oriented than they actually are (Boehm, 2002).

Also Glass (2001, p. 14) states that 'For too many years, the traditionalists have put so much emphasis on planning software development activities that they produced plans that were rigid and inflexible'. The question is therefore; how do we better handle inevitable changes throughout a project's lifecycle and respond to environmental changes?

The last category we used for the analysis is therefore the way in which the companies react to changes.

4. Five different companies and their development practices

In this section we describe the development practices in the five companies based on the analysis and analyze them according the scheme developed in the previous section. Table 2 summarizes this analysis.

4.1. Xodus

Xodus was founded in the beginning of the 1990s and has today roughly 100 employees. The main focus lies on developing financial systems in Windows environments. The customers that the systems are developed for are small, medium sized and large companies.

The project covered in this interview was an update project and lasted roughly three years. Nine persons were involved in this project, eight of whom were programmers. The final delivery was delayed by six months. The project was an update of their financial system for large companies, which is a standard product aimed at a particular market section. Large updates like this one are performed

Table 2
Summary of the result

	Xodus	Mandus	Kuling	Cellex	Yampus
Individuals over processes	–	+	–	+	++
Working software over documentation	––	O	–	+	++
Customer over contract	O	+	O	O	++
Change over plan	–	+	–	+	++

approximately every second year. Smaller updates, such as adding specific features, are released in between. Specific adaptations for specific customers' needs are implemented by the company's consultancy department.

The product manager took part in the project, and was responsible for validating content and functionality. He was also responsible for prioritizing emergent requirements. The project coordinator ensures that different parallel ongoing development projects are coordinated. One project member was responsible for time planning. Time planning was based on the phases. During the development phase additional milestones were planned after each finished module, to enable closer control. One project member was responsible for developing the reports. One employee is responsible for integrating the modules developed by 'partner companies'. That person, however, supports all projects. Since the consultancy department maintains contact with the customers, they acted as a customer in the project that was discussed in the interviews. Two members of the consultancy department and the support department acted as testers. The consultancy department was also responsible for the acceptance test.

4.1.1. Lifecycle

Xodus does not have any formal software model, but in practice projects more or less follow a waterfall model: requirements, development, and test phase.

The basic requirements were taken from the previous version of the system. Only the new requirements were analyzed and designed, and therefore the *requirements phase* lasted only three months. The original time schedule was however changed, since new requirements continued to arrive. This was one of the reasons for the project's delay.

The *development phase* in this project included the development of all functionality, component tests and integration tests. No coding standards were applied. A graphical interface standard and a standard for component documentation were used. This phase lasted approximately two years. As no tests besides component tests and integration tests were carried out before the test phase, many problems only appeared during the test phase. This was the other main reason for the project's delay.

The *testing phase* in this project was divided into two parts. In the first part the functionality in the delivered system was still open to changes. Free tests and tests of the new requirements were made. This phase lasted six months, which was longer than planned. The later part of the test phase was divided into three sub phases. One sub-phase focused on the main functions of the system. In the second sub-phase more detailed tests were performed, and finally the graphical user interface was checked. The system was delivered as one final delivery approximately six months later than planned. The progress of the project was measured by how many percent of each activity was finished in relation to the time schedule.

When the second interview took place, a new project leader had been employed. Xodus has started to develop product specifications based on proposals from the consultants. Three update projects have been implemented during the six months since the above-described project was finished. The lifecycle of these projects has been much more iterative; a prototype was constructed first, and then the consultants tested the prototype. Based on the feedback, the prototype was further developed, and so on. There was a meeting with the customers every week and the product manager was responsible for maintaining customer contact and ensuring that the results of the meetings were included in the development process. The later update projects have been divided internally into several smaller projects. Everyone agrees that the development process works better when a large project is split into smaller projects. 'The motivation increases, you can see the goal.'

4.1.2. Documentation

The documents produced can differ somewhat from project to project but are very similar. The project definition, requirements specification, component descriptions, object diagrams, sequence diagrams, test cases, help files and meeting minutes are often documented.

The programmers write the technical documentation and the consultancy department writes the user documentation. Sometimes the project manager writes the non-technical documentation, if the consultancy department runs out of time.

4.1.3. Communication and coordination within the projects

In this project, communication with the consultancy department started during the requirements phase, with roughly two meetings for each of the five modules. The meetings were held to see if the requirements were understood correctly and to explain those that were not. Contact with the consultancy department almost stopped during the development phase and was not restarted until the test phase began.

Most of the problems were dealt with within the development team during the morning meeting, but the team could also discuss freely during the course of the day, since they were seated very close to one another.

The members of this project could work freely and were able to affect the outcome of the project. This freedom resulted in, amongst other things, changes that were not discussed with other project members. This in turn led to things having to be redone.

Each developer was responsible for at least one module and 'owned' the code. No other project member was allowed to make changes in other developers' code. This made the project very vulnerable when someone became ill or left the company.

The development speed was dramatically reduced towards the end of the project, mainly owing to reduced motivation among the project members. The developers did not have a common understanding of the goal of the project. Delays within the project also led to reduced motivation.

4.1.4. Communication with users/customers

It is the consultancy department that handles most of the communication with users and customers; they mediate all new requirements. Besides the consultancy department a support service is available. Two economists are responsible for telephone support. Technical problems are passed on either to the person responsible for integration or to someone in the consultancy department. The economists in support deal with the other problems and also take notes of feedback. All problems and feedback are entered into a case handling system and are later discussed and prioritized by the project members and the consultants. Many of the additional requirements must be implemented as customer adaptations, since this system is a standard product. The consultancy department implements all customer adaptations.

A user meeting is arranged once every year. All users are invited and seminars on different subjects are arranged. Tips, tricks and new features in the system are demonstrated and the users can also present feedback regarding the system. Quite a large number of users make use of this possibility. The meeting lasts for three days and also includes some social activities.

4.1.5. Evaluation according to agile values

Individuals and interaction over processes and tools. Communication between the consultancy department and the system developers at Xodus is sporadic during the whole process. Communication was frequent within the development team. The members of the project had extensive freedom and were able to affect the outcome of the project. The code was owned by the individual developer, which meant that no one else had access to it. However, the communication did not result in changes in the project. The project plan was not changed, even when the development stagnated. We judge that on the whole, Xodus relied on plans rather than on individuals (−).

Working software over comprehensive documentation. As mentioned earlier, Xodus documents extensively. The programmers write the technical documentation and the consultancy department usually writes the user documentation. Progress was expressed as a percentage of how far the development had proceeded in relation to the time schedule of the waterfall-like project plan. Xodus did not release any software until the whole system was finished. In this respect Xodus implemented a very traditional way of developing software (−).

Customer collaboration over contract negotiation. Xodus uses their support service among other things to collect feedback from customers. The consultancy department regularly keeps in touch with users and customers during the rest of the year. It is also possible for customers to propose new functions.

The consultancy department performs all testing before the system is delivered to the customers. In this way, even use-related issues are detected. The consultancy department acts as a user or customer representative.

On the one hand Xodus as a whole keeps in contact with their customers and users, and uses different feedback channels to improve the software. On the other hand, the consultancy department acts as a filter between the users and customers and the developers. So neither traditional nor agile practices are implemented (O).

Responding to change over following a plan. Xodus used a sequential lifecycle with almost no iterations. New requirements dropped in during the requirements- and development phase, delaying the project. The development phase lasted two years and after the test phase the system was finally delivered. The development was not organized in a way to cope with changes originating from user and customer requests (−).

4.2. Mandus

Mandus was founded in the late 1990s and has roughly 10 employees. The customers that usually turn to Mandus are local municipalities, county councils, and small and medium sized companies. A majority of their projects cover web-based products. Lately they have also started to develop standard products with Microsoft.NET™. The standard products are usually plug-in applications to widespread previously existing systems developed by other companies.

In our interviews we covered one project for a completely new web-mail application. The web-mail project involved about four or five people. It was planned to last approximately four months. The product developed was a web-based email system including specific functionality such as an import function to gather data from some other systems. Since Mandus is quite a small company, all their project teams normally consist of three to four persons. The web-mail project consisted of four project members and one 'helicopter' project manager who participated in project meetings, which lasted roughly 30 min once every week. The 'helicopter' project manager was in charge of the time planning and progress monitoring and the discussions during these project meetings, but he had no contact with the customer. The project co-operated with a customer-side project group. The project manager on the customer side handled the communication with Mandus.

4.2.1. Lifecycle

Mandus, who mainly develop customer specific software, has developed a software model that is followed in all their projects, both when updating products and when developing completely new products. The software process is in practice a list of documents that should be created during projects. There is usually a very rough time plan made with only a few milestones; completing the requirements specification, completing the development and a pilot installation. It is the requirements phase, development phase, and the test phase that are the most important phases according to the interviewee. The lifecycle of all their projects is sequential so the identified phases are per-

formed consecutively. It is always the project leader who is responsible for planning the projects. Generally, the problem description presented by the customer serves as the foundation for the project.

In the *requirements phase* Mandus performs the analysis and design, and of course identifies the requirements. In this specific project, the customer was knowledgeable about how the web mail system would be used and greatly influenced the requirements specification. Implementation had already begun before the development phase started. The system was split into three parts with defined interfaces. Some of the component tests were already implemented during the development phase. All other tests were left to the *test phase.* The tests were performed in a development environment that was set up to simulate the operation environment. Component, integration, and system tests were performed.

The pilot installation took place before final delivery, to allow the customer to test the system. The *pilot phase* lasted two weeks and ended with an evaluation meeting together with the customer. Most of the viewpoints brought up by the customer resulted in changes to the system. Thereafter followed a new two-week test period, and another evaluation meeting was held. At the end of this phase a full installation was performed. Before the customer accepted the delivery, the system was required to run for a month without any problems. The acceptance agreement was signed in consultation with the customer.

One aspect that usually differs, depending on whether a new product is developed or an old one updated, is the complexity of the requirements specification, which is much more extensive when developing a new product. Mandus uses prototypes as much as possible to ensure that they really are developing what the customer wants. When the customer sees and tests the prototypes he can give feedback on possible changes. All deliveries are via Internet, where the customers can download the system.

They tried to establish a very detailed timetable where different parts of the project were described. They followed this timetable as closely as possible.

4.2.2. Documentation

Mandus produces a number of documents depending on the project. This documentation includes different kinds of project-management documents, such as the product specification, test specification and system specification, the user manual, and progress reports. New and more documents are required when developing new products. These can include an installation manual and administrator manual, and maintenance documentation.

Mandus finds producing documents very time consuming when new releases are to be delivered. Besides these documents, all developers also document most of the source code. The developers write the documentation for their parts of the system and the project manager writes all remaining project documents. Almost all documentation is digital and can be found on Mandus' website, from where it can be downloaded.

4.2.3. Communication and coordination within the projects

Most of the communication within the project group is verbal and face-to-face, but they also use e-mail. The difference between an update project and a project for a new product is that communication is much more frequent in the latter. Mandus often make use of design consultants concerning user interfaces and other graphical tasks.

In almost all projects, all developers participate in creating the requirements specification, but the design and architecture are created by only one of the programmers. Each developer is responsible for the part of the system that he/she develops. The parts that each developer has developed should only be changed by him or her alone. The developers are given positions in accordance with their knowledge and they have extensive freedom concerning how technical problems should be solved. To motivate their personnel, Mandus has a bonus system, and extra gratuities, such as, for example, weekend trips to celebrate a successful project.

4.2.4. Communication with users/customers

Mandus usually only has contact with the customer, not the end-users. The customer conveys the users' perspective and as Mandus states "it is the customer who pays". In their projects, a project meeting usually takes place once a week, to discuss project progress together with the customer.

For larger projects, Mandus generally signs support agreements with their customers. The type of questions that they answer varies, depending on the type of the supported system. Even customers without a support agreement call frequently about problems that are often not due to Mandus' systems. Since Mandus even provides some support for customers without an agreement, this can lead to too much unpaid support and thus becomes a problem. Additional support activities that Mandus offers are product training when the product is delivered. The customers also can email Mandus about problems or ideas for improvements. For the web-mail system, a website was set up, where end users could fill in a feedback form and send it by e-mail to Mandus.

4.2.5. Evaluation according to agile values

Individuals and interaction over processes and tools. At Mandus, projects usually have formal meetings once a week during the whole process. At these meetings it is mainly the progress of the project that is discussed. In between they have frequent communication face-to-face or by e-mail. Communication is more frequent when developing new products. The developers are free to solve technical problems, and they own their own code. A bonus system is used to motivate the personnel. Mandus can be judged to implement this agile value to some extent (+).

Working software over comprehensive documentation. Mandus produces an increasing number of documents; all staff is responsible for documenting their parts of the product. New products especially require more documenta-

tion. Documentation is regarded as time consuming. The emphasis on documentation can be seen as an attempt to develop a more structured way of developing software. Almost all documentation is digital and can be downloaded from the company's website.

On the one hand, projects try to follow the time plan as accurately as possible. On the other hand, Mandus delivers prototypes fairly often to ensure that development addresses the customer's need. On the whole, they implement a mixture of traditional plan-oriented and agile prototype-based practices (O).

Customer collaboration over contract negotiation. Mandus generally signs support agreements with their customers, which include free telephone support. Although end-users do not usually have any contact or communication with Mandus, in the project discussed here, a website for end-user feedback was established. It was possible for the customer to test a pilot version before the final system was delivered. Since Mandus frequently uses prototypes, is it possible to obtain feedback early in the project. The customer also took an active part in the requirements phase and participated in project meetings once a week. Especially the latter lets Mandus' practices tend more towards the agile side (+).

Responding to change over following a plan. Mandus has a waterfall-like process model that is not followed very strictly, e.g. in the project discussed here development started during the requirements phase. There is often only a very rough time plan with few milestones. Prototypes are used as much as possible. During the pilot phase requirements are added or changed. This means that Mandus takes new requirements into consideration even late in the process. These practices take change into account and provide space for their projects to react to it (+).

4.3. Kuling

Kuling was founded in the late 1990s and has roughly 900 employees spread over five cities. They are a consulting company focusing on telecommunication. Their largest customer is a Swedish telecom company. Ninety to 95% of all projects are directed towards this single customer.

In our interviews, we have covered two projects, one update project and one developing a completely new product. At least 100 persons were involved in each of these projects. The update project was planned to last four months and covered an additional application to a web-based sales promotion system. The project X for the new product was in collaboration with two other large companies and the final goal was a large standard business solution.

4.3.1. Lifecycle

Kuling uses different software models for their projects. The software model for update projects is sequential and consists of three main phases; the analysis, development and test phases.

To be able to allocate the right resources for the update project discussed during the first interview, dates were established early on for when the different phases should be finished. These and other important events were used as milestones for the project.

The project started with the *analysis phase*, for which the customer contact delivered a first draft of a requirements specification. A product specification was developed where the customer requirements were clarified and further defined. The intention is to find and analyze all requirements during this phase, but in this case, new requirements also appeared during the development phase. Time estimates were based on experiences from earlier projects. The interviewee performed these estimates together with the developers. A very rough design was also developed in this first phase.

The rough design was then further developed in the *development phase*. The plan was to develop all functionality during this phase. However, due to errors and some misunderstandings concerning the requirements specification, some development took place during the test phase. During the development phase, developers performed component tests to ensure that their components were behaving correctly in isolation.

In the *test phase* the testers carried out system tests. Before the final product was delivered, it was handed over to Kuling's management. Since the company is responsible for the operation of the entire system, additional tests were performed to ensure that the new product worked correctly together with existing systems.

To check the project's progress, the project manager followed up the time estimates in order to compare them with the time it actually took to perform the activities. The completed activities were then compared against the timetable to check that the project was not falling behind the planned schedule.

Kuling always performs the three above-mentioned phases, both within update projects such as the one described above and for completely new development. When a new product is developed all phases are more extensive and the requirements specification is also more complex.

Their main customer has demanded that Kuling become more efficient in the future, regarding their working procedures and costs. Due to these demands and also because they themselves believe a new software model is needed, they have started to introduce RUP (Kruchten, 2000). The intention is to have an iterative software model. Kuling has also developed a very simple 'working model' of their model, which is more or less a checklist of what to do in really small projects. On some occasions, Kuling has even adapted their way of working to fit the way the customer works. However, this only happens if the customer requires them to do so.

4.3.2. Documentation

Kuling produces a number of documents in all projects. This documentation includes different kinds of project management documents, product specification, test specifi-

cations, system specifications, user manuals, progress reports, and also a final report.

The developers document all source code. The testers are responsible for producing the test specifications. The project leader writes the management documents, the progress reports and also the final report. The product specification steers the developers. Kuling notes increasing demands from their customers for documentation. Existing documentation of old systems is incomplete. Stricter requirements for system documentation are therefore needed to fulfill new requirements for higher quality documentation.

Kuling uses a change handler tool to trace changes that have been made in the design structure, components, source code, interfaces between components etc.

4.3.3. Communication and coordination within the projects

Communication within the project group is handled via discussions and e-mail. They usually have daily discussions throughout the entire project. This is possible since they are located very close to each other in the same building. In smaller projects, project meetings with all project members are usually held once a week.

In project X, they had quite a deep personnel hierarchy, since the project members were distributed over different departments. To address the problem of lost information they let each department supervisor, group leader etc. inform their own staff.

The project members have considerable freedom to design technical solutions, as long as this does not negatively affect the budget or the timetable. All developers have the right to make changes in each other's source code without always informing their leader. The different parts of the product are assigned so that the developers handle the parts related to their area of expertise. To prevent loss of knowledge due to a person leaving the company, less experienced members of staff sometimes handle a certain part of a product in order to become acquainted with it. This is not implemented in all their projects due to time shortage. To motivate their personnel Kuling arrange social activities such as laser games, dinners etc.

4.3.4. Communication with users/customers

When Kuling presents a prototype or delivers a system to their customers, both buyers and end-users are usually present. It is important to use prototypes, especially when working with new products, to ensure customer satisfaction at an early stage.

Other communication with the customer is handled via discussions, telephone and e-mail. Kuling provides telephone support for their customers in case product problems occur. Calls come from both end-users and buyers, depending on the type of product and type of problem. The questions can vary a lot, from a small user problem to complex installation procedures. Kuling has a special system where all incoming ideas and customer suggestions are entered. This information can later be used to improve a specific product or as a starting point for a completely new development. Apart from telephone support, Kuling also offers systems training, both via Internet and at their offices.

4.3.5. Evaluation according to agile values

Individuals and interaction over processes and tools. During development projects, the development group at Kuling has frequent communication face to face and by e-mail, besides the weekly project meetings. However, communication does not override the project plan. In the large project that was the subject of the second interview, communication was organized by letting each department's supervisor inform the staff under them in a hierarchical way. The developers are free to solve technical problems in their own and in others' code. Kuling's practices implement the traditional emphasis on process rather than relying on individuals and their interaction (−).

Working software over comprehensive documentation. Kuling produces a number of different kinds of documents. The product specification steers the developers. The company experiences increasing demands from their customers regarding quality and documentation.

Progress is measured on the basis of performed activities rather than running code, though prototypes are used to validate requirements.

To communicate the technical design and extend their knowledge, less experienced developers work with parts of a system that they are not yet familiar with. This way the design is—in line with the agile values—communicated via co-operation and communication rather than documentation. Altogether, Kuling implements traditional values rather than agile values regarding this aspect (−).

Customer collaboration over contract negotiation. Kuling provides telephone support, for both end-users and customers. All viewpoints and proposals are stored. In the update project discussed in the first interview, the customer—a telecommunication provider with in-house IT-expertise—delivered a draft requirements specification, which Kuling further defined. Contacts with the customer occur occasionally during the analysis and development phase. When working with new products and prototypes, customer contact is more frequent. Kuling also offers systems training.

The customer sometimes even requires Kuling to follow a certain development process. Due to demands made by their main customer, the company is in the process of redefining their basic process model.

Kuling co-operates closely with their customers. However, customer contact does not usually alter the overall project plan during the development process (O).

Responding to change over following a plan. Kuling uses sequential software processes for their projects. They have started to introduce a more iterative process model. For new development, the phases are more extensive than in an update project. In very small projects a very simple working model is used, which is more like a checklist of

what to do. Kuling is also able to adapt their way of working to fit the customers' processes if necessary. Optimally, all requirements are defined in the analysis phase but new requirements almost always appear during the development phase. Some developing even took place in the test phase, because of errors and misunderstandings in earlier phases. However this is avoided if possible (−).

4.4. Cellex

Cellex was founded in the beginning of the 1980s and has today roughly 80 employees who work mainly with software development, marketing, and sales of standard systems. The company is active in a vast number of different business areas, but their main focus lies on developing applications for banking, real estate, and insurance. To keep up to date, Cellex have experts in the above-mentioned areas who keep track of what is happening within the market.

In our interviews, we covered one update project and how Cellex generally work in a project for developing a completely new product. Most projects involve about 25–30 persons. Cellex have a service agreement with about 95% of their customers. The service agreement for their large systems states that Cellex is obliged to perform three larger updates per year, besides bug fixing and possibly some smaller updates etc. The delivery dates for the larger updates are agreed upon in advance, so all involved personnel, as well as their customers, are well aware of the deadlines.

4.4.1. Lifecycle

Cellex has two different software models that they have defined in-house. One is for updates of standard software and one is for projects where they develop completely new products. The software model used for update projects consists of a sequential lifecycle with three distinct phases. These phases are the requirements, the development, and the test phase. All analysis and design is performed during the requirements phase.

The product-responsible, customers, and also developers all contributed to the requirements for this specific update. Some of these requirements were changes to earlier updates and some were completely new functionality. The product-responsible had the final say on what should or should not be included. Customer-specific requirements can be included in an update, if the project members consider it a useful feature for all customers. Otherwise, customers can pay extra to have specific features developed.

The requirements were prioritized, and the product-responsible, in co-operation with the developers, estimated the time required for the implementation. A deadline was set after which no additional requirements should be accepted for the current release. However, this was not strictly followed. Requirements that were seen as very important or that improved the system were added even after this date. This led to reprioritization and even removal of existing functions in order to keep the deadline. During the *development phase*, the developers also performed different kinds of basic tests of the parts they had responsibility for.

During the *test phase*, the developers continued to test their product parts and components, while the testers focused on integration- and system testing. Cellex usually writes the acceptance test that is performed when the systems are delivered. The customers then perform their own user tests. Some even perform more technical tests to ensure that the system corresponds to the agreed requirements. Feedback from these tests can sometimes be used for improving the next update.

Project progress was measured by examining what proportion of the requirements was developed according to the time plan. If they were behind schedule, requirements were reprioritized or even removed after informing the customers.

Cellex rarely use prototypes. Prototypes are only used if the systems are really large, to increase the likelihood of customer satisfaction.

4.4.2. Documentation

Our interviewee thought that documentation should be handled more efficiently. More documents should be produced during the development process. The time scheduled for writing the documents is often too short. Today, documentation is usually written at the very end of the project. Changes in the requirements specification should be documented in a supplementary document which should be accessible through a shared folder. This procedure is not followed. There is no tracing between the requirements and the actual source code.

The person who is responsible for a component also produces the documentation for that component, often in the source code. The product-responsible produces all documents related to project management and also user manuals and requirements specifications.

4.4.3. Communication and coordination within the projects

Communication between project members is very frequent in all projects, especially during the development of a new product. Discussion is unproblematic, since they are all located very close to one other. Mail, telephones and chat are also used frequently. Another way of communicating is the use of a shared folder. Everything concerning the current project, such as the requirements specification, error reports, time plans etc. is kept in this folder. Meetings between the product-responsible and key developers are usually held once a week during the requirements phase and once every two weeks during the development phase.

The project members have extensive freedom and are able to affect the outcome of the projects. This freedom also implies responsibility towards the rest of the project members and towards the company. If someone comes up with a superior technical solution, nothing stops this

solution from being implemented as long as it is in line with the company standards.

There is always someone who is responsible for a specific part of the code, but all project members have the right to change others' code if necessary. For larger components, the responsibility can be shared between several developers. Cellex always tries to have at least two or three people that have deep knowledge of each component, to prevent problems if someone leaves the company. If possible, less experienced personnel take responsibility for some specific parts to broaden their knowledge. This is a motivation for the employees, who feel that they develop, whilst the company gets better-trained personnel. The project members have quite extensive experience. Many of them have been working for the company or in similar businesses for more than ten years.

To motivate their employees Cellex includes all personnel at parties, kick-offs etc.

4.4.4. Communication with users/customers

The support division at Cellex consists of eight persons. Besides providing support for the customer they perform a number of different tasks. Throughout the project lifecycles they feed projects with ideas for improvements and changes. They perform some tests, and they also help to write some manuals and help texts for the products. A large diversity of questions is handled during support. Both end users and buyers call the support-line. Support usually handles the technical questions, and questions about use are passed on to the product-responsible. All incoming ideas from the customers are logged so that nothing is missed e.g. for a new update. Communication with customers is handled via telephone, mail and meetings. Apart from telephone support, Cellex also offers product training via Internet, and they carry out service at the customers' site.

Cellex maintains a frequent dialog with their customers, especially during the requirements phase. Differences in opinions between the customers and Cellex are solved by discussion, but in the end it is money and time that decide what should be included in an update. When it is time to deliver an updated system, support personnel inform their customers and other companies that the updated system is now on the market. Cellex always sends out a newsletter to all their customers, which describes the features of the new update. Both buyers and end users are normally present when a finished product is presented for a customer.

4.4.5. Evaluation according to agile values

Individuals and interaction over processes and tools. Communication within the development team at Cellex is very frequent, especially during the development of new products. For communication, they also use a shared folder. All developers at Cellex have access to each other's code and are allowed to make changes if necessary. Developers are encouraged to share responsibility for single modules. By joining or taking over responsibility for a new module they also have the possibility to broaden their competence. This kind of in-service training also motivates the employees. The deadline for each release is fixed, despite only having a rough project model. If features cannot be implemented, they are postponed to the next update. The conscious deployment of cooperation as a means of knowledge transfer can be seen as an implementation of agile values, though the overall process organization is not questioned (+).

Working software over comprehensive documentation. On the one hand, Cellex itself is not satisfied with the time that is allocated for writing documents. Today, almost all documentation is produced after a product is finished. More than one person has knowledge about each component.

The majority of their projects consist of regular updates of standard systems. This series of projects can be regarded as an iterative process where a running version always exists. In case of a project delay the delivered functionality is diminished, rather than postponing the deadline. This is done in consultation with the customer. Altogether their practices—though not their ambition—emphasize working software rather than comprehensive documentation (+).

Customer collaboration over contract negotiation. Cellex' support division logs all incoming ideas from the customers, which are later used in the development process.

Customers are able to propose new functionalities when Cellex updates or develops new products, though it is the product manager who decides what to implement. Though customer contact is close during the requirements and testing phases, and feedback is taken care of, contact is facilitated via a product manager (O).

Responding to change over following a plan. Cellex has two different software processes, one for update projects and one for new development. Both processes are sequential and consist of three phases; the requirements, the development, and the test phase. In projects developing new software, the only difference is that the requirements phase is more extensive than in update projects.

A deadline is set up after which it is not permitted to add more requirements. However, important requirements can be added later in the process. In the development phase, requirements are reprioritized, added or removed, in order to conform to the delivery date. Customers perform user tests, and feedback from these is sometimes used in the larger update.

Cellex rarely uses prototypes except when developing very large systems. However, as most of the projects are update projects, the versions currently in use provide the basis for feedback and further development. Most of the development can thus be seen as an ongoing change to existing software (+).

4.5. Yampus

Yampus is a small company employing seven people. Yampus mainly delivers services and products that help companies, municipalities and organizations to use the Internet as a communication channel. Yampus has an

e-democracy focus. Their e-democracy portal consists of several linked modules.

The people in the company have worked together for a number years and are very close to each other. The only change in the project group since it started is that a new designer has been hired, as the old one left the company.

The project covered in this interview lasted approximately six months, and the system has been updated several times since then. The system enabled the user to fill in an application form for adult education and send it via Internet instead of by mail. It also allowed the applicants to perform online tests. The project team consisted of three members: one project manager, one programmer, and one designer.

The users of the system can be divided into two groups: administrators and end-users. Only the administrators were represented at the first meetings. The end-users became involved as test subjects at a later stage of the process.

4.5.1. Lifecycle

Yampus, the young company developing web applications, does not use any predefined software model. Based on the study of different models, the most appropriate parts had been selected. This model is quite informal and is adapted and improved to fit the way of working and the needs at hand.

Some deadlines can be found in the development process. One is when the requirements specification, sometimes in form of a demo, is finished and another is at the end of an approval stage. Planning of the development is done in a sequential way but it allows for requirements changes during the entire development of a system.

The first version of the requirements specification in the project discussed took a couple of months to produce. It was produced together with two contact persons from the customer. Thereafter, a couple of meetings took place with the administrators and a project manager from the customer side. From Yampus, the project manager and a second project member usually attended these meetings. Changes to requirements were documented towards the beginning of the project. All changes at the end of a project were implemented directly in the system.

Our interviewee describes their process as a continuous one. Development begins before the requirements phase ends, and continues almost until a project is finished. The customer in this case was very active during the development phase and examined the system almost every day.

Every function in the program is tested by the programmer and another member of the company. In this case the customers arranged their own tests when the system was ready. The customer tested not only the administrative interface but also the user interfaces, and collected feedback from the end-users.

Yampus handles the running of the system and each module in the system was delivered as soon as it was finished, which means that deliveries took place often. A module was considered to be finished when the customer was satisfied with it. There were no formal delivery contracts. This has never been necessary since they have always reached agreement with their customers.

Systems maintenance occurs approximately twice a year. One of the larger updates altered the whole front of the system, implementing, among other things, a new design and a new search interface.

The progress of the project was not measured in any special way. When a module, or a part of a module, was delivered and the customer was satisfied, that part of the project was regarded as completed.

4.5.2. Documentation

In all projects, most of the documentation is found in the source code and in general only the functions are documented. The source code itself documents the system. Each module has a log file where all changes are recorded; these log files can be seen as a kind of historical change-documentation of the system. The database and file structure are self-explanatory since all tables and folders have names that explain their functionalities. According to the project leader, no other technical documentation is necessary.

User manuals are available on line for most of the modules. Some modules do not need any user manual or documentation since they are small and self-explanatory. Focus is upon developing usable products that are easy to understand.

4.5.3. Communication and coordination within the project

When a new project starts, all those involved in the project meet to discuss how the project should be organized. This meeting often takes place at Yampus' premises. Sometimes the meeting takes place in the customer's office. In-house communication is mostly face-to-face, since everybody is working in the same office area and can easily reach the others. Communication between developers and between the project leader and developers is frequent throughout the whole project.

All developers have complete control and power over their coding. They decide how to solve technical problems. Functionality and user interfaces are discussed together with the project leader and the customer. The customer has extensive influence regarding these aspects. No one person owns the code, which should be documented in a way that allows other developers to understand it and make changes if necessary.

It is difficult to find time to take courses to learn new techniques; so instead, one person learns a new technique and then teaches the others. To motivate the employees Yampus often arrange different kinds of activities such as trips or games.

4.5.4. Communication with users/customers

Yampus has a support service on their website. Customers and users are encouraged to fill in their questions via a form to ensure more efficient support. Questions are

answered via e-mail or a phone call. This support service is used irregularly and periodically. For example, the support service is used fairly often when a new version is released, but at other times the service is hardly used. It is also possible to contact support via telephone, although Yampus prefers the use of the support service on the website. All questions and proposals, etc. are stored in a database. The contents of the database provide new ideas for further development. Customers' questionnaires are also studied before new modules are developed or developed further, to make sure that the right things are developed.

During a project, most customer contact takes place directly with the developer or with the project leader, via E-mail or telephone. Communication is most frequent during the initial and the final parts of a project. The customer also has access to the website during the whole development process, which make it possible for the customer to follow the development and to give feedback continuously.

4.5.5. Evaluation according to agile values

Individuals and interaction over processes and tools. The development team at Yampus works very close together, and since all developers are working in the same open-plan office it is easy to communicate frequently face-to-face. It is possible for the team members to decide independently how technical problems should be solved. The code is documented in a way that makes it possible for all developers to understand and change it. As there is only a rough schedule, the work organization and the deadlines are subject to negotiation within the team and with the customer. Yampus clearly relies on individuals and their interaction rather than on formal processes and tools (++).

Working software over comprehensive documentation. Nearly all documentation is part of the source code. There is a log file connected to each module where all changes are documented. The database and file structure are regarded as self-explanatory. User manuals are available on line when necessary. Yampus does not measure project progress in a special way; they deliver the product. The project is regarded as completed when the customer is satisfied. To improve the staff's qualifications, each employee takes courses/read books and teaches the others. Yampus implements the agile values in their practices (++).

Customer collaboration over contract negotiation. Yampus has a support service on their website. Customers and users also call the developers directly. All questions and proposals are stored in a database. This information generates ideas for new development. Questionnaires are sometimes used before new development takes place.

Customers take an active part in the establishment of the requirements specification. During the development process, customers have access to the website that is under construction. This means that the customers can follow the process and give feedback continuously. Yampus' practices in this respect are clearly on the agile side (++).

Responding to change over following a plan. Yampus does not use a predefined software process model. They work in an informal way and continuously try to improve their way of working. Development is still planned in a sequential way but they allow changes or new requirements throughout the development of a system. The requirements specification and the approval stage are important deadlines. Completion of the phases is floating, which means for example that the development phase can start before the requirements phase is ended, and continues almost until the system is finished. When a system module is finished and the customer is satisfied, it is delivered. Here Yampus is once again at the agile end of our sample (++).

4.6. Summary of the empirical part

Comparing the analysis of the different interviews we see a variety of practices ranging from very rigorous, plan driven development to agile practices. The following list summarizes the findings to be discussed further in the next section.

- Co-operation and face-to-face communication are relied on as long as the project team is small enough, and are complemented by more formal lines of communication when the groups are larger. The question of code ownership is also treated in a variety of ways but most often the developers are allowed to make changes in each other's code.
- Documentation is in many cases a topic that leads to the discussion of the short-comings of today's practices. Analyzing our interviews, we found very different ways of handling documentation; a small company providing web-based systems uses only a requirements specification. They document the code and have developed naming standards. Other companies have a more traditional way of handling documentation. In some companies the project organization is used to distribute knowledge about single components, so that they are not dependent on individual developers.
- All of the companies are more open for customer co-operation than anticipated. However, that co-operation might not take place in exactly the way the agile community or the participatory design discourse (Shuler and Namioka, 1993) proposes. One important communication channel to the customer is the support line. It is often used to collect feedback and proposals for further development. The providers of standard software in particular use this channel to inform the requirements process for their, in some cases, regular updates. Companies developing custom software often have an even closer contact with their customers and even co-operate with users.

The majority of the companies have more than one project model that is adapted according to the size of the project, depending on whether it is a development from scratch or an update project. Late requirements in the process are common; these requirements are in most cases taken into

account even late in the process. In practice, prototypes are often used when requirements are unclear. Versions are provided for customer and user testing, and the development contract sometimes even includes major updates based on feedback, which are followed by a new test period. Some companies even publish early versions on the web and encourage the customer to explore them and give feedback. The smallest company has the most iterative and informal way of developing software.

5. Discussion

In this section we would like to bring forward some of the aspects we observed when analyzing the interviews that we think take the discussion around agile development one step further. They concern the for us surprising variability of the software development process even within the same company, how the agility of the observed practices relates to the project characteristics, and the way user and customer feedback was gathered and used in the development.

Finally, we discuss the reliability of our findings.

5.1. Adapting the development process

Industrial software companies are more inclined to adapt their predefined software models to specific circumstances than they themselves actually believe. Most of the companies in the study claimed that they follow a linear software model with predefined phases. The requirements, which are the core of the process, should be stipulated in the requirements phase. However, all interviewed companies accept new or changed requirements even late in the process. Adding requirements late in the process means that the predefined process is interrupted and is no longer linear. This adds a kind of flexibility into the process and makes it non-linear and in a way iterative.

A second issue is the question of iterative or evolutionary development. According to Highsmith and Cockburn (2001) a team is not agile if the feedback loop with customers is as long as six months. Instead, short iterations within the range of two-to-six-weeks are recommended. On the one hand, only one of the examined companies implements this recommendation for all of their projects. On the other hand, the majority of the interviewed companies sell products that include more or less regular updates.

A final issue concerns the organization of maintenance and updating of standard software, which is not much discussed in the literature. Even when organized in a waterfall-like way, the single projects can hardly be discussed as independent development projects. Each of them can be seen as a cycle in an ongoing evolutionary project.

As a fourth point, it is worth mentioning that most of the companies had different process models, or at least different variations of their process model, which they used depending on the size and complexity of the development project. In one case they even told us that they might use the customer's process model, if it would benefit the project. The situation-specific adaptation of process models and methods, and the reflected deployment of more formal or more agile organization that Glass (2001) proposes in his debate article already seems to be implemented in some companies.

Even though one company ran into problems with a waterfall-like process, the practices in general relate to the requirements of the customers or the market sector the company addresses. They cannot be described as fire-fighting or ad-hoc reaction to changing contingencies. The practices can be identified as such, and deviations or variations are motivated by project specific circumstances.

5.2. Which project characteristics relate positively to agile development practices?

The companies do not consciously decide to implement agile practices. Instead, they follow what they perceive as common sense. Our interview study seems however to indicate that the presence of more agile practices might depend on the project characteristics as they are discussed in the literature regarding the suitability of agile methods. In the above, we categorized the projects that were subject to the interviews according the project criteria proposed in (Glass, 2001). Table 3 relates this categorization to the level of fulfillment of the four agile principles. Regarding the projects that were subject to the interviews, we have compared conformance of the practices with the four principles of agile development to the set of project characteristics from (Glass, 2001). This seems to indicate a positive relation between project size and criticality and traditional development. There is no clear relation between innovativeness and traditional software engineering. However, further research is required to provide statistically significant relations.

5.3. Cooperation with customers and users

Cooperation with users and customers is little discussed in software engineering research literature, though in the companies that were the subject of our study, this cooperation provides important input to development and correlates with how the development takes place. All of the companies in our study use their support service to collect customer and user feedback on the software, besides providing help for problems related to the use of their products. The support service is also used for collecting proposals for new functionalities. Apart from the support service, some direct contact with customers/users is common, for example through user meetings, through consultants, or by involving the customers and users in requirements specification and testing. The latter, though, is restricted to the development of custom software. Most of the companies have a long-term relationship with their customers—partly because they sell a standard product that is updated continuously.

Table 3
Relating the level of conformance with agile principles to the project characteristics from Glass (2001)

	Innovativeness			Criticality			Size		
	H	M	L	H	M	L	H	M	L
Individuals over processes		**Yampus** **Mandus** ***Kuling***	**Cellex** *Xodus*		**Cellex** *Xodus* *Kuling*	**Mandus** **Yampus**	***Kuling***	**Cellex** *Xodus*	**Mandus** **Yampus**
Working software over documentation		**Yampus** Mandus ***Kuling***	**Cellex** *Xodus*		**Cellex** *Xodus* ***Kuling***	Mandus **Yampus**	***Kuling***	**Cellex** *Xodus*	Mandus **Yampus**
Customer over contract		**Yampus** **Mandus** Kuling	Cellex Xodus		Cellex Xodus Kuling	**Mandus** **Yampus**	Kuling	Cellex Xodus	**Mandus** **Yampus**
Change over plan		**Yampus** **Mandus** *Kuling*	**Cellex** *Xodus*		**Cellex** *Xodus* *Kuling*	**Mandus** **Yampus**	*Kuling*	**Cellex** *Xodus*	**Mandus** **Yampus**

Bold = ++; **Bold** = +; Plain = O; *Italic* = −; *Italic* = −−.

These ways of maintaining the customer and user relationships and deploying them in development is not considered in either the traditional school of software engineering or in the agile development publications. Even the discourse around participatory design (Shuler and Namioka, 1993) does not consider this way of involving users (Henfridsson and Holmström, 2002; Hansson et al., 2004 are exceptions to this reflection). From an agile perspective, the support and consultancy departments might act as 'gardeners' for user feedback and use stories, as proposed in Rittenbruch et al. (2002). Here new research opportunities open up that would relate to both the software engineering and the participatory design discourse.

5.4. How reliable are our findings

As Sweden is representative for the first world countries, similar samples in Western Europe of North America can be expected to yield a similar spectrum. We did not address outsourcing or distributed development. Otherwise a variety of business models for software development were represented. The main result does not consist of recommendations for practitioners. It provides a challenge for further research: Instead of advocating one set of methods or 'best practices' over another, research might better address the suitability of specific methods and practices for specific circumstances.

6. Conclusions

Concluding the article we have to answer the question our title asks: How agile are industrial software processes? The answer is: It all depends! It depends on which of the different principles you take to judge agility. It also depends on the characteristics, not only of the company, but also the single project. The companies quite expertly combine agile and traditional practices and adjust their practices according to the situation at hand. Here, more empirical research regarding the combination of agile and traditional methods and practices would contribute to the understanding of the situated adaptation of both kinds of methods.

Relating the project characteristics proposed in Glass (2001) to the level of conformance between the reported practices and the four agile principles seems, on the one hand, to confirm the principles as relevant differentiations. On the other hand, it can be seen as confirming the adequacy of the observed practices. Our research does not allow the claiming of any statistical significance. Here also, more research is necessary to understand better the relationship between project characteristics and which methods may be suitable.

A third interesting area of research that our results open up for is the improvement of industrially applicable ways of including users as well as customers into the software development. Here, neither the software engineering nor the participatory design discourse addresses the deployment of the support line or user groups as a way of bringing use to design.

References

Abrahamsson, P., Warsta, J., Siponen, M.T., Ronkainen, J., 2003. New directions on agile methods: a comparative analysis. In: Proceedings of the 25th International Conference on Software Engineering (ICSE'03).

Agile Manifesto, 2001. <http://www.agilemanifesto.org>, 041210.

Baskerville, R., Balasubramaniam, R., Levine, L., Pries-Heje, J., Slaughter, S., 2003. Is Internet-speed software development different? IEEE Software 20 (6), 70–77.

Beck, K., 2000. eXtreme Programming Explained: Embrace Change. Addison-Wesley, San Francisco.

Boehm, B., 1981. Software Engineering Economics. Prentice-Hall, UK.

Boehm, B., 2002. Get ready for agile methods with care. Computer 35, 64–69.

Clement, D.L., Parnas, P.C., 1986. A rational design process: how and why to fake it. IEEE Transactions on Software Engineering SE-12 (2), 251–257.

Cockburn, A., 2000. Selecting a project's methodology. IEEE Software 17 (4), 64–71.

Cockburn, A., 2002. Agile Software Development. Addison-Wesley, UK.

Cockburn, A., Highsmith, J., 2001. Agile software development: the people factor. Computer 34, 131–133.

Dittrich, Y., Lindeberg, O., 2004. How use-orientated development can take place. Information and Software Technology (46), 603–617.

Gerson, E.M., Star, S.L., 1986. Analyzing due process in the workplace. ACM Transactions on Office Information Systems 4 (3), 257–270.

Glass, R.L., 2001. Agile versus traditional: make love not war. Cutter IT Journal 14 (12), 12–18.

Hansson, C., Dittrich, Y., Randall, D., 2004. Agile processes enhancing user participation for small providers of off-the-shelf software. In: Eckstein, J., Baumeister, H. (Eds.), Extreme Programming and Agile Processes in Software Engineering, Lecture Notes in Computer Science, vol. 3092. Springer-Verlag, Heidelberg, pp. 175–183.

Henfridsson, O., Holmström, H., 2002. Developing E-commerce in internetworked organizations: a case of customer involvement throughout the computer gaming value chain. Database for Advances, Information Systems 33 (4), 38–50.

Highsmith, J., 2002. What is agile software development? Crosstalk. The Journal of Defense Software Engineering, 4–9.

Highsmith, J., Cockburn, A., 2001. Agile software development: the business of innovation. Computer 34, 120–122.

Kruchten, P., 2000. The Rational Unified Process: An Introduction. Addison-Wesley, UK.

Martin, R.C., 2002. Agile processes. In: Martin, R.C. (Ed.), Agile Development: Principles, Patterns and Practices. Prentice Hall.

Naur, P., 1985. Programming as theory building. Microprocessing and Microprogramming (15), 253–261.

Nørbjerg, J., Kraft, P., 2002. Software practice is social practice. In: Dittrich, Y., Floyd, C., Klichewski, R. (Eds.), Social Thinking—Software Practice. MIT Press, USA.

Osterweil, L., 1987. Software processes are software too. In: Proceedings of the 9th International Conference of Software Engineering, March 1987, Monterey, CA, USA, pp. 2–13.

Rittenbruch, M., McEvan, G., Ward, N., Mansfiels T., Bartenstein, D., 2002. Extreme participation—moving extreme programming towards participatory design. In: Binder, T., Gregory, J., Wagner, I. (Eds.), Proceedings of the PDC 2002, Malmö, Sweden.

Robson, C., 2002. Real World Research, second ed. Blackwell Publishing.

Royce, W., 1970. Managing the development of large software systems. In: Proceedings of IEEE WESCON 1970. IEEE, pp. 1–9.

Saarnak, S., Gustafsson, B., 2003. A comparison of lifecycles—agile software processes vs. projects in non-Agile software companies. Master Thesis no.: MSE-2003-12, Blekinge Institute of Technology.

Shuler, D., Namioka, A. (Eds.), 1993. Participatory Design: Principles and Practices. Lawrence Erlbaum Associates, Hillsdale, NJ.

Suchman, L.A., 1983. Office procedure as practical action: models of work and system design. ACM Transactions on Office Information Systems 1 (4).

Wenger, E., 1998. Communities of Practice. Learning, Meaning, and Identity. Cambridge University Press.

Williams, L., Cockburn, A., 2003. Agile software development: it's about feedback and change. Computer 36, 39–43.