LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

AGILE METHODS IN SMALL SOFTWARE PROJECTS

The subject of the thesis has been approved in the department council meeting of the Department of Information Technology in Lappeenranta University of technology on 13.12.2006.

Examiners: Professor Kari Smolander and M.Sc. Pasi Venäläinen
Instructor: M.Sc. Markus Runonen

Lappeenranta 2.12.2007

Lassi Romanainen
Tuulimyllynkatu 5 as. 3
53500 Lappeenranta
Lassi.Romanainen@quicknet.inet.fi

ABSTRACT

Author:	Romanainen, Lassi
Subject:	**Agile Methods in Small Software Projects**
Department:	Information technology
Year:	2007
Place:	Lappeenranta

Master’s Thesis. Lappeenranta University of Technology, 67 pages, 7 figures and 7 tables.

Supervisor: Professor Kari Smolander
Keywords: Agile methods, software development, small software projects

Agile software development methods are attempting to provide an answer to the software development industry’s need of lighter weight, more agile processes that offer the possibility to react to changes during the software development process.

The objective of this thesis is to analyze and experiment the possibility of using agile methods or practices also in small software projects, even in projects containing only one developer.

In the practical part of the thesis a small software project was executed with some agile methods and practices that in the theoretical part of the thesis were found possible to be applied to the project. In the project a Bluetooth proxy application that is run in the S60 smartphone platform and PC was developed further to contain some new features.

As a result it was found that certain agile practices can be useful even in the very small projects. The selection of the suitable practices depends on the project and the size of the project team.

TIIVISTELMÄ

Tekijä:	Romanainen, Lassi
Nimi:	**Ketterät menetelmät pienissä ohjelmistoprojekteissa**
Osasto:	Tietotekniikan osasto
Vuosi:	2007
Paikka:	Lappeenranta

Diplomityö. Lappeenrannan teknillinen yliopisto. 67 sivua, 7 kuvaa ja 7 taulukkoa.

Tarkastaja: Professori Kari Smolander
Hakusanat: Ketterät menetelmät, ohjelmistonkehitys, pienet ohjelmistoprojektit

Ketterät ohjelmistonkehitysmenetelmät yrittävät tarjota vastauksen ohjelmistotuotantoalan tarpeeseen saada kevyempiä ja ketterämpiä ohjelmistonkehitysmenetelmiä, jotka antavat mahdollisuuden reagoida muutoksiin kehitysprosessin aikana.

Tämä työ käsittelee ketterien ohjelmistonkehitysmenetelmien ja niiden määrittelemien käytäntöjen hyödyntämisen mahdollisuutta pienissä, jopa vain yhden ohjelmistokehittäjän projekteissa.

Työn käytännön osassa toteutettiin pieni ohjelmistoprojekti, jossa valittiin käytettäväksi tietyt teoriaosan analysoinneissa mahdollisiksi havaitut ketterien menetelmien käytännöt. Projektissa jatkokehitettiin S60 –älypuhelinalustalla sekä PC:llä ajettavaa Bluetooth proxy-ohjelmistoa.

Lopputuloksena havaittiin, että tietyt ketterien menetelmien käytännöt voivat toimia myös todella pienissä ohjelmistoprojekteissa. Sopivien käytäntöjen valinta riippuu projektista sekä projektin koosta.

FOREWORDS

This thesis is written in SYSOPENDIGIA Finland Ltd in Lappeenranta for the Lappeenranta University of Technology.

I want to thank the inspector of this thesis, Professor Kari Smolander for all the good comments and advices and for the interest towards this subject and thesis.

I would also like to thank my instructor, Markus Runonen and the second inspector Pasi Venäläinen from SYSOPENDIGIA Finland, for all the great pointers and words of advice about the Bluetooth proxy implementation and thesis work generally.

Last but not least, big thanks go to Noora for her patience and support during this quite long writing and graduation process, and to my family for always believing in my possibilities and supporting me with the choices I have made.

ALKUSANAT

Tämä työ on kirjoitettu Lappeenrannassa, SYSOPENDIGIA Finland Oy:ssä, Lappeenrannan teknillisen yliopiston tietotekniikan osastolle.

Kiitän diplomityöni tarkastajaa, professori Kari Smolanderia, hyvistä neuvoista ja kommenteista sekä mielenkiinnosta työtäni ja sen aihetta kohtaan.

Haluan myös kiittää työni ohjaajaa, Markus Runosta sekä toista tarkastajaa Pasi Venäläistä SYSOPENDIGIA Finland Oy:sta, asiantuntevista neuvoista sekä aihetta, että yleisesti diplomityötä koskevissa kysymyksissä.

Lopuksi haluaisin kiittää Nooraa hänen kärsivällisyydestään ja tuestaan tämän pitkähkön kirjoitus- sekä valmistumisprosessin aikana, sekä myös vanhempiani ja perhettäni siitä, että he ovat aina uskoneet mahdollisuuksiini ja tukeneet minua valinnoissani.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

DSDM	Dynamic Systems Development
FDD	Feature Driven Development
GPRS	General Packet Radio Service
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
KAELOC	Thousand (kilo) assembler-equivalent lines of code (Drobka et al., 2004)
KESLOC	Thousand (kilo) equivalent source lines of code computed using formulas that normalize reused and modified code in terms of new lines of code (Reifer, 2002b).
PC	Personal Computer
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
XP	Extreme Programming

TERMS

Localhost	A reference to the same machine where the software is running. IP address of the localhost is 127.0.0.1

1 INTRODUCTION

1.1 Objectives of the thesis

The purpose of this thesis is to study if we could benefit from the use of agile methods also in very small software projects. Another goal is to enhance an existing software product with new features.

In the case project, a proxy application running on a smartphone will be updated to support User Datagram Protocol (UDP) traffic and two-way Transmission Control Protocol (TCP) traffic. Smartphone is an advanced mobile phone with personal computer –like functionality.

The application is used in for example testing the Java environment implementation on the smartphone, because it enables running tests using Bluetooth connection instead of General Packet Radio Service (GPRS) via the Internet. This is more cost-efficient and is more secure, because GPRS traffic is expensive and slower than Bluetooth and also requires opening certain ports of the test machine to the internet.

A combination of agile practices will be used to test how they could benefit a small one-developer software project, or do they just cause problems or unpredictable behavior.

1.2 Structure of the thesis

First there will be a short brief about software development methods as a whole and how they are expected to work with small software projects; are there some special features regarding those projects.

After that, some of the most common agile methods and practices are introduced and they are analyzed if they could be applied also to small software projects.

The practical part of the thesis will contain a further development project of a Bluetooth proxy application for the Symbian smartphone operating system. According to the analysis in the previous chapters, some of the agile practices are chosen and taken into use in the case project.

After the execution of the case project, the results and findings regarding the agile methods or practices are analyzed and recommendations are given for the future appliers. Also the further research possibilities are documented.

In the end, a final conclusion of the use of agile methods in small projects according to this study will be given.

2 Software development methods and small projects

2.1 What is a method?

Quite a variety of different terms are in use in the field of software development. One of the most common terms is **method**. Many researchers and writers have written a definition for it. Fitzgerald et al. (2002) describes method as

"A coherent and systematic approach based on a particular philosophy of systems development, which will guide developers on what steps to take, how these steps should be performed and why these steps are important in the development of an information system."

2.2 Software development methods

Also the term of software development method is quite fuzzy; there are many similar terms like software development model, software life-cycle model, software process model et cetera. In practice, all of these terms refer to approximately same idea; defining the steps how to create software.

The earliest models of software evolution date back to the 1950s and 1960s. The purpose of these early software life-cycle models was to provide a conceptual scheme for rationally managing the development of software systems. Probably the most common example of these life-cycle models is the waterfall model, where software evolution proceeds through an orderly sequence of transitions from one phase to the next in order. (Marciniak, 2002)

In contrast to the life-cycle models, software process models are seen as a networked sequence of activities, objects, transformations and events that embody strategies for accomplishing software evolution. Such models can describe the software life-cycle activities in a more precise and formal way.

Software development methods and processes are nowadays a key element in successful software projects. For example Lockheed cut development costs 75%, shortened time-to-market for 40% and cut the amount of errors in their software 90% by improving their development processes during five years (McConnell, 1997).

Small projects set certain restrictions for the use of predefined software development methods. These issues are discussed in the chapter 2.6.

2.3 What is an agile method?

Agile software development methods have risen as a reaction against the “traditional”, heavyweight methods that are sometimes seen as strictly regulated, bureaucratic and slow. In this study the word “traditional” refers to the older, commonly used methods like the waterfall model. These traditional software development methods are also often criticized to be far from the real ways software engineers use to efficiently develop software.

In February 2001 a group of software consultants and practitioners gathered together and signed the Agile Software Development Manifesto, which started a whole new movement in the software development industry (Beck et al., 2001):

“Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan” (Beck et al., 2001)

The manifesto summarizes quite well the core elements of agile software development; the goal is to create working software, not to fulfill the predefined development process. Agile development should also make it easy to react to changes, which are very common in current project environments. The power of communication and collaboration is also highly valued asset within the agile methods.

Many of the agile methods also contain similar practices; constantly updated requirement lists, incremental and iterative development and small, frequent deliveries.

On one hand, agile methods are gaining support among software developers all over the world; the ability to create software without heavy, bureaucratic processes or thorough documentation easily draws attention within the ranks of the developers and project managers. On the other hand agile methods have raised much debate and criticism; is it really possible to create good quality software with such light processes?

Real life experience has shown that it is possible (see chapters 2.4.1 and 2.4.2) at least in some project environments. Success of an agile project depends on many factors, for example are the project team members and customers accepting the new ways of working, is the organization supporting the change and is the selected agile method suitable for the problem or the project environment.

2.4 Why agile methods?

Agile methods are relatively new software development methods, so there are relatively small amount of publications about the real world performance of agile methods, though the amount is increasing as the interest towards agile methods is continuing. Few of those released papers are introduced in this chapter.

2.4.1 Agile method survey

Donald J. Reifer executed a survey in 2002, in which 14 firms answered to a list of questions about their usage of agile techniques. In the Table 1 these firms are introduced by their industry. In the State of progress column Pilot means a first project to prove that agile method works, Pathfinder is used to determine how to integrate the agile method to the company processes and Production means an agile method is used in a normal production project. (Reifer, 2002b)

Table 1 Characteristics of the responded firms in the survey (Reifer, 2002b)

Industry	**Firms using agile methods**	**Projects**	**Year first tried**	**State of progress**	**Average size (KESLOC)**
Aerospace	1	1	2001	Pathfinder	23
Computer	2	3	2000	Pilot	32
Consultants	1	2	2000	Pilot	25
E-business	5	15	2000	Production	33
Researchers	1	1	2000	Pilot	12
Scientific	0	0	2000	Pilot	N/A
Software	2	4	2000	Production	25
Telecom	2	5	2000	Production	42
Total	**14**	**31**		**Average**	**31,8**

In the survey, seven of the 14 organizations had collected hard cost, productivity and quality data. Five of these organizations had benchmarks from the earlier projects, so they could compare the performance of the usage of agile methods to the traditional methods. According to this data: (Reifer, 2002b)

- Productivity was improved 15 to 20 percent, compared to published industry benchmarks (Reifer, 2002a).
- Costs were reduced by 5 to 7 percent on average, as compared to published industry benchmarks (Reifer, 2002a).
- Time-to-market was improved by 25 to 50 percent compared to previous projects in the participating firms.
- Quality of the software remained on par with their earlier projects, if compared by the defect rates.

Although these results were positive, it has to be taken into account that these were small, low-risk projects staffed by selected teams under controlled situations. The results might scale neither to larger projects nor higher-risk situations.

2.4.2 Piloting XP in four mission critical projects

In 2004, Jerry Drobka, David Noltz and Rekha Raghu from Motorola reported in the IEE Software magazine about piloting Extreme Programming (XP) on four mission-critical projects on 18-month period. (Drobka et al., 2004)

They used a slightly tailored XP process to fit their company needs, because few aspects of the process did not work with the type of the product they were developing. They also hired an experienced outside XP consultant as their coach to implement the XP process in the projects. (Drobka et al., 2004)

The results in this study were very positive. The productivity was measured using the formula (total KAELOC) / (total staff effort). KAELOC means thousand (kilo) assembler-equivalent lines of code.

The increase in productivity ranged from 162% to 385% when compared to the waterfall model. Also the work enjoyment seemed to increase, as 85% of the developers enjoyed using XP, 68% stated that using XP increased their job enjoyment and 79% would choose to work with XP again if possible. (Drobka et al., 2004)

The test coverage of the projects was also very high, unit test coverage ranged from 73% to 95%. Quality of the code was measured by formula: (total number of defects found in the system testing) / (total KAELOC). This kind of metric was used because similar data was also available for previous projects so comparison was possible. The increase in quality when using XP ranged from 51% to 74%. Majority of the developers (80%) were also more confident in the design and code they generated while pair programming than when they work alone. (Drobka et al., 2004)

Overall the XP pilot projects were seen as a positive experience though a few challenges with certain practices such as pair programming were encountered. The pilot program convinced the writers that it is possible to use agile process such as XP to develop complex mission-critical systems with long life cycles and that the productivity gains that XP provides make it an attractive development process for most object oriented projects. (Drobka et al., 2004)

2.5 Features of small software projects

Classifying a software project as a small project is not an easy task. It depends on many environmental factors, like the size of the organization; if the company has 10 000 employees, a project of “just” 50 people can appear small to them. Also the complexity of the project can be one criterion. A complex domain usually requires a more detailed project structure and experienced staff. (Russ and McGregor, 2000)

A detailed description for a small software project could not be found from the literature within the timeframe of this project, so here are listed some criteria for a small software project:

- Very few or just one developer
- Low amount of interaction between personnel
- Short time frame
- Low complexity
- Small amount of work to do

These characteristics, especially the small amount of developers, bring challenges when predefined development methods are used in a small project because process models are usually designed for a bigger project organization.

2.6 Methods and small projects

Small software projects have some restrictions when using software development methods. Most of the restrictions relate to the small amount of developers or project personnel. For example, it is not possible to implement a practice that involves a developer reviewing the work done by other developer, if there is only on developer in the project.

Also the time frame of the project may be restrictive; some practices may take so much time to implement that they do not provide enough value in short projects. Some practices can be also aimed to cope with the complexity of the domain or problem, but some small projects may include only low level of complexity.

In this thesis some of the most common agile methods are analyzed based on how they would fit in a small software project. The methods are inspected practice by practice, if they contain something that is not possible to implement in a small project, or if some practices are useless or too heavy in small projects.

3 AGILE SOFTWARE DEVELOPMENT METHODS

In this chapter there will be introduced some of the most common agile methods which are later analyzed if they could possibly be applied to even the smallest software projects.

3.1 Extreme Programming (XP)

Extreme Programming methodology arose from the problems caused by the long development cycles of the traditional development models (Beck, 1999). It started as “simply an opportunity to get the job done” (Haungs, 2001) with practices that had been found effective in software development in the earlier decades. After many successful trials in practice the XP methodology was “theorized” on the key principles and practices. The individual practices used in XP are not new as such, but they have been collected and lined up to function with each other in a new way, so that they can be seen to form a new methodology. The term “Extreme” comes from taking these common sense practices into extreme levels. (Abrahamsson, 2002)

3.1.1 XP Process

The Extreme Programming process consists of six separate phases, as illustrated in figure 1. Here the phases are introduced according to Abrahamsson (2002).

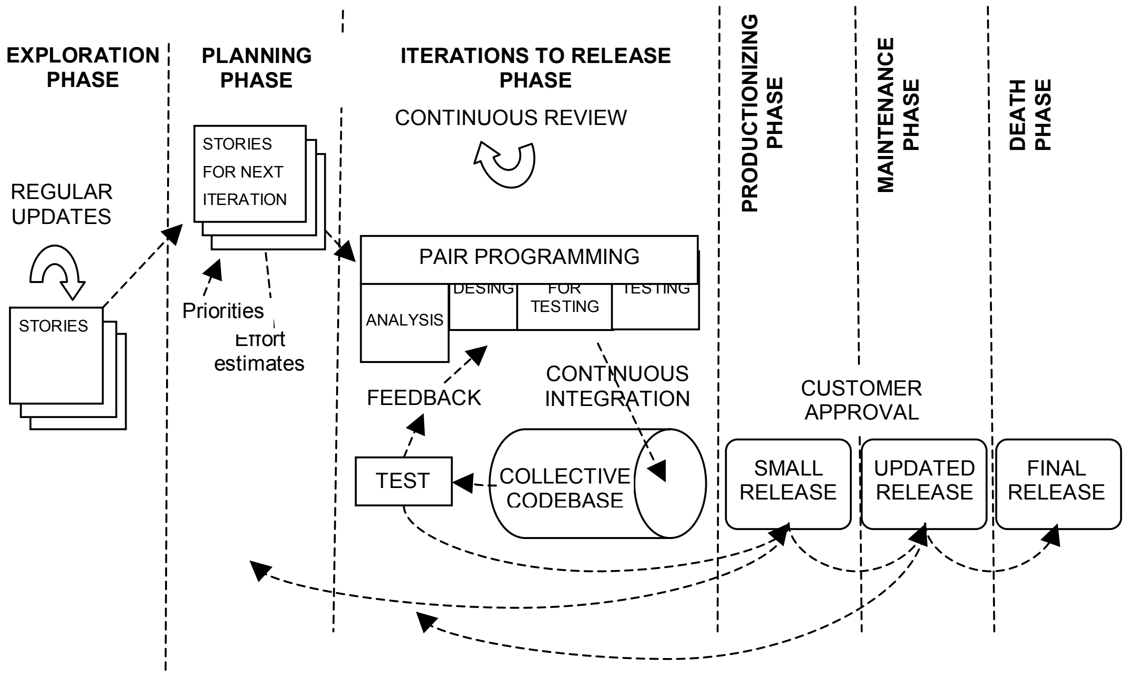

Figure 1 Extreme Programming process (Abrahamsson, 2002)

In the Exploration phase, so called User stories are created. The customers write out things they would like to see in the first release of the software on a story card. Each story card contains one feature. Simultaneously the project team familiarizes itself with the tools and technologies needed in the project. The Exploration phase takes from few weeks to few months, depending on how well the programmers know the technology.

In the Planning phase the User stories will be arranged to priority order and the contents of the first small release is agreed. Programmers make effort estimates for the stories and the schedule is agreed upon those estimates. The planning phase takes a couple of days and the first release usually takes no more than two months.

The ‘Iterations to release’ phase consists of several iterations of the system to create the first release. The first iteration creates the basic architecture of the system by implementing the user stories that enforce building the structure for the whole system. The customer decides the stories to be implemented in the iteration. The functional tests created by the customer are run at the end of every iteration cycle. After the last iteration, the system is ready for production.

In the Productionizing phase additional testing and checking is conducted before the system is released to the customer. New changes can still be found at this phase and it has to be decided if they are included in the current release. The implementation iterations for the changes may need to be shortened from three weeks to one week. If some changes are postponed, they are documented for later implementation e.g. in the maintenance phase.

In the Maintenance phase, after the first release is productionized and taken into use, the XP project has to keep the system running whilst implementing new features. This requires an effort for the customer support tasks also, which may decelerate the implementation pace of the new features. The Maintenance phase may require incorporating new people into the project team and changing the team structure.

The Death phase is reached when the customer does not have any stories to be implemented, i.e. the customer is satisfied with the system. In the Death phase the necessary documentation of the system is finally written as no more changes to the architecture, design or code are made. Death may also occur if the project is terminated for some reason; e.g. the system cannot deliver the desired outcome or it becomes too expensive for further development.

3.1.2 Practices

Extreme Programming is a collection of known and already existing practices. These are introduced in the following according to Abrahamsson (2002).

The Planning game includes close interaction between the customer and the programmers. Programmers make the effort estimates for the customer stories and the customer then decides the scope and timing of the releases.

XP also features **small and short releases**. A simple system is “productionized” rapidly; at least once in every 2 to 3 months. After the first release new versions are released even daily or at least monthly.

In XP, the **system is defined by a metaphor** or a set of metaphors, created together with the programmers and the customer. It guides the whole development by describing how the system works.

The emphasis in design is to get the **simplest possible solution** that is implementable at that moment. Any unnecessary complexity or extra code is removed immediately.

Software development is **test driven**; unit tests for the code are written before the code and are run continuously. Customer writes the functional tests.

The developed system gets often **refactored**; e.g. duplicate code is removed, communication is improved, code is simplified and made more flexible.

Pair programming is also distinctive for XP. In pair programming two programmers write code on one computer. They also analyze, design and write tests together (Beck, 2005).

In XP, the **codes are collectively owned**, i.e. anyone can change any part of the code at any time.

New pieces of code are integrated to the code-base as soon as they are ready. The system is built many times a day, and all tests are run. Tests have to be passed for the changes in the code to be accepted.

Working week is 40 hours in maximum. Two overtime weeks in a row is handled as a problem to be solved.

Extreme programming states that **customer has to be present in the same premises than the developers**. Customer has to be available full-time for the team.

Coding standards are in use and followed by the programmers. Communication through the code is encouraged.

The XP team has its **own set of rules that are followed**. The rules can also be changed any time, but the changes have to be agreed upon and their impacts assessed.

3.1.3 Using XP

Kent Beck suggests that when an organization has already been developing software with its own practices, XP should be applied gradually by adding XP practices that meet organizations goals and values. (Beck, 2005)

In the latest edition of the book (Beck, 2005), Beck also distinguishes primary XP practices and corollary practices. The adaptation to XP practices should be started with the primary practices such as weekly planning cycle, test first, pair programming etc. After the primary practices are in use properly and without any trouble, corollary practices like daily deployment and real customer involvement can be applied also. The reason for this is that for example deploying the system daily into customer use straight away, without lowering the defect rate first with pair programming, automated tests and daily integration, could cause a disaster. (Beck, 2005)

3.2 Scrum

The term ‘scrum’ originates from a term used in the game of rugby, where it means “getting an out-of-play ball back in to the game” with teamwork (Schwaber, 2002). Scrum has been developed for managing the systems development process; it does not define any specific software development techniques. It concentrates on how the team members should function to produce the system flexibly in a constantly changing environment. (Abrahamsson, 2002)

Scrum aims to enhance the used practices in the organization by using frequent management activities; for example short frequent meetings with the development team. The purpose of these activities is to find any deficiencies in the development process or practices as fast as possible. (Abrahamsson, 2002)

3.2.1 Scrum Process

The Scrum process is presented briefly, according to the definitions of Schwaber and Beedle (2002) and Abrahamsson et al. (2002).

The process contains of three phases: pre-game, development and post-game (see figure 2)

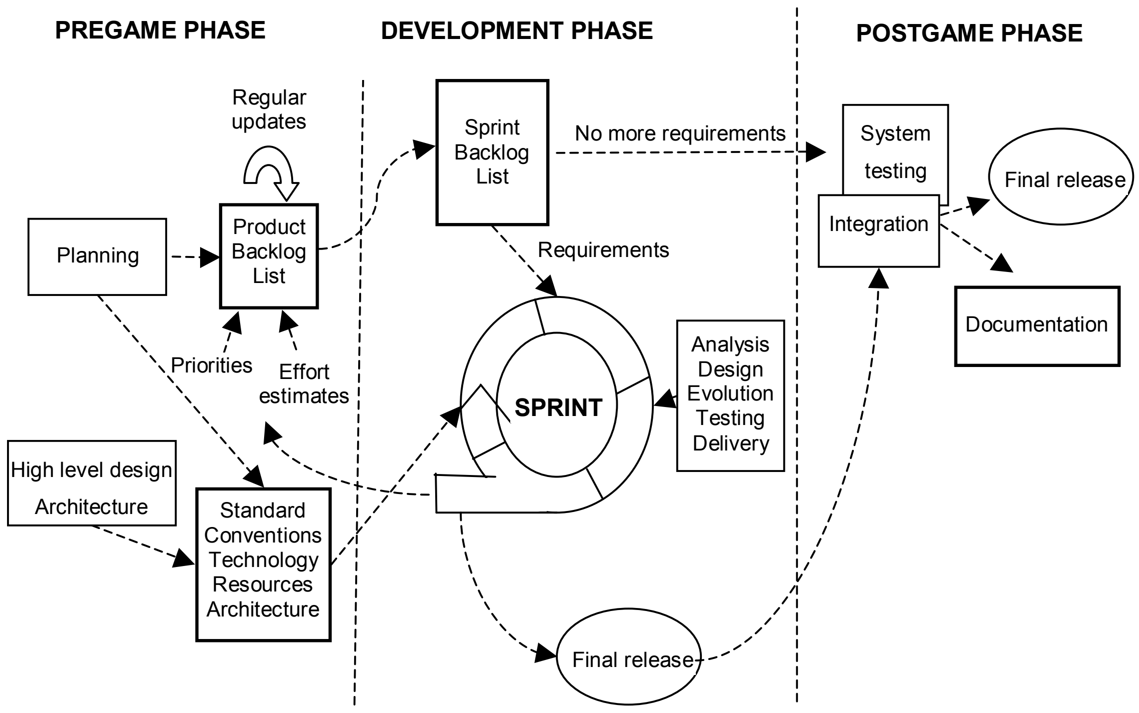

Figure 2 Scrum process diagram (Abrahamsson 2002)

The pre-game phase is a preliminary phase, which contains two sub-phases; planning and architecture / high level design.

In planning phase the system is defined and a Product Backlog list which contains all the currently known requirements is created. The requirements are prioritized and effort estimates are generated. The items in Backlog are constantly updated to be more accurate and new ones can be added. Planning also includes defining the project team, tools and other resources, risk assessment and management, training needs and verification management approval. The updated Backlog is reviewed by the Scrum Team(s) at every sprint phase to gain their commitment for the sprint.

In the architecture phase the high level design and architecture is done based on the current items in the Backlog list. After this, a design review meeting is held and decisions of the implementation are done on the basis of this review. Also preliminary plans for the contents of the releases are prepared.

The development phase is called the agile part of the Scrum process. It is treated as a “black box”, where unpredictable changes are expected. This means that all the environmental and technical variables (e.g. time frame, quality, requirements and resources) are identified, observed and controlled through Scrum practices during the Sprints. Usually these matters are taken into consideration only at the very beginning of the project, but Scrum aims to control them constantly to be able to flexibly adapt to these changes.

In the development phase the system is developed in Sprints. Sprints are iterative cycles which consist of the traditional phases of software development: requirements, analysis, design, evolution and delivery. Also the architecture and the design of the system evolve during the Sprints. One Sprint is targeted to last from one week to one month. One system development project can contain for example three to eight Sprints, before the system is ready for distribution.

The post-game phase contains the closure of the release. This phase is entered when it is agreed that all the environmental variables (for example requirements) have been completed. In this phase, no more items can be added or old ones modified. This phase also includes tasks like integration, system testing and documentation. The system is now ready for distribution.

Scrum identifies six different roles with different responsibilities. These roles are Scrum Master, Product Owner, Scrum Team, Customer, User and Management. The most important roles are presented in the table 2 according to Schwaber and Beedle (Schwaber, 2002).

Table 2 Roles and responsibilities

Role	**Responsibility**
Scrum Master	Takes care that the project is carried through according to the Scrum rules and practices. Is responsible for removing any impediments from the process.
Product Owner	Officially responsible for the project, managing, controlling and making visible the Product Backlog list. Is selected by the Scrum Master, the customer and the management. Makes all the final decisions related to the Product Backlog, participates in creating the effort estimates and turns the backlog items into features to implement.
Scrum Team	The project team, which has the authority to organize itself and make the necessary decisions to achieve the goals of each sprint. Is involved in the effort estimation, creating the Sprint Backlog, reviewing the Product Backlog list and suggesting the impediments that need to be removed from the project.
Customer	Participates in the tasks related to Product Backlog items.
Management	Responsible of the final decisions along with the charters, standards and conventions to be followed in the project. Participates also in setting the goals and requirements for the project, in gauging the progress, in selecting the Product owner and reducing the backlog with the Scrum Master.

3.2.2 Practices

Scrum does not require or provide any specific software development practices. Instead, it requires certain management practices and tools to be used to avoid the chaos caused by unpredictability and complexity. (Schwaber 1995) In this chapter the most important practices of Scrum are presented as described by

Abrahamson et al. (Abramson 2002) and Schwaber and Beedle (Schwaber 2002)

Product Backlog contains everything that is needed in the final product based on the current knowledge. It defines all the work that has to be done in the project. It is a prioritized and constantly updated list of requirements for the system. Product Backlog can contain items such as features, functions, bug fixes, defects, requested enhancements and technology upgrades. This practice includes all the tasks regarding the backlog from creating the Product Backlog list to updating and controlling it consistently. The Product Owner is responsible of maintaining the Product Backlog.

In Scrum, **effort estimation** is also an iterative process, where initial effort estimates are defined more accurately when more information is available. The Product Owner and the Scrum Team(s) are together responsible for the effort estimation.

Sprint is the procedure of adapting to the changing environmental variables (such as requirements, time frame, resources, knowledge or technology). The Scrum team organizes itself to produce a new executable product increment in a Sprint that takes time from one week to one month.

A **Sprint Planning meeting** is a two-phase meeting organized by the Scrum Master. In the first phase of a Sprint Planning meeting the customers, users, management, product owner and Scrum team decide the goals and the functionality of the next sprint. In the second phase the Scrum Master and the Scrum Team focus on how the product increment is implemented during the Sprint.

Sprint Backlog is a list of Product Backlog items that are selected to be implemented in the next sprint. The items are selected by the Scrum Team with the Scrum Master and the Product Owner in the Sprint Planning meeting, on the basis of the prioritized items and goals set for the Sprint. Unlike the Product Backlog, the Sprint Backlog is stable until the Sprint is completed. When all the items in the Sprint Backlog are completed, a new iteration of the system is delivered.

Daily Scrum meetings are held to keep track of the progress of the Scrum Team continuously and to solve any problems that have arisen during the Sprint. All the members of the Scrum team must attend to this meeting. Also other people can attend, for example to check the progress of the sprint, but they must remain silent; only members of the Scrum team and the Scrum Master are allowed to speak. Any deficiencies or impediments in the development process are looked for, identified and removed to improve the process. The meeting lasts approximately 15 minutes, and every member of the Scrum Team tells what he/she has done since the previous meeting, what problems he/she may have encountered and what he/she will do before the next scrum meeting. Scrum meetings are arranged by the Scrum Master.

Sprint Review meeting is held on the last day of the Sprint. There the results of the Sprint are presented to the management, customers, users and the Product Owner by the Scrum team and the Scrum Master. The participants evaluate the results and make decisions what to do next. The meeting can bring up new items to the Product Backlog and even change the direction of the system being built.

3.3 Feature Driven Development

Feature Driven Development (FDD) is an agile method for developing systems, but it does not cover the whole development process; it focuses on the design and implementation phases (Palmer and Felsing, 2002). However, it has been designed to work with the other activities of the process. FDD emphasizes quality aspects throughout the process, frequent and tangible deliveries and accurate monitoring of the progress of the project. FDD also claims to be suitable for delivering critical systems, unlike some other agile methods. (Abrahamsson, 2002)

3.3.1 Process

FDD process consists of five sequential processes; Develop an overall model, Build a feature list, Plan by feature, Design by feature and Build by feature (figure 3). The iterative design and build by feature part supports agile development by quickly adapting to late changes in requirements or business needs. (Abrahamsson, 2002)

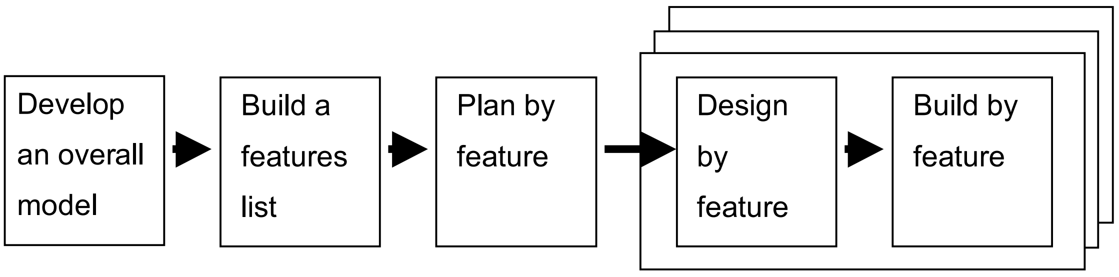

Figure 3 Sequential processes of FDD

When the **Development of an overall model** begins, the domain experts are already aware of the scope, context and requirements of the system to be built.

Also use cases and functional specifications are likely to exist at this phase. The domain experts present a so called “walkthrough” for the team members and the chief architect

The domain is then further divided into separate domain areas. A more detailed walkthrough is held for each of the domain areas by the domain members. After the walkthroughs, development teams in the domain areas work in small groups to create object models for the domain areas. Simultaneously, an overall model for the whole system is being developed.

In the **build a features list** process, the walkthroughs, object models and existing requirements form a good basis for building a features list for the system. The list consists of client valued functions that need to be included in the system. The list is divided into so called major feature sets, which include functions for a certain domain area. Major feature sets are also divided into feature sets, which represent features within that domain area. The features list is reviewed by the users and the sponsors of the system to assure its completeness and validity.

During the **plan by feature** process a high level plan for the system is created, where the feature sets are sequenced according to their priority and dependencies. Feature sets are also assigned to Chief programmers who are responsible of the smaller teams implementing those features. Furthermore the classes that were identified in the “develop an overall model” process are assigned to individual developers. Those developers become “class owners” for the classes. Also schedule and biggest milestones can be set for the project at this point.

Design by feature and **build by feature** are iterative processes, during which the features are designed and implemented. The length of iteration should be from few days to a maximum of two weeks. A small group of features is selected from the feature set(s) and feature teams are formed to develop the selected features. There can be multiple feature teams developing their features concurrently. The iterative process includes such tasks as design, design inspection, coding, unit testing, code inspection and integration. If the iteration is successful, the completed tasks are promoted to the main build and a new iteration begins with a new set of features taken from a feature set.

Roles and their responsibilities in FDD method are described in table 3.

Table 3 FDD Roles and responsibities

Role	**Responsibility**
Project Manager	Administrative and financial leader of the project. Protects the team from outside distractions and provides appropriate working conditions. Has the ultimate say on the scope, schedule and staffing of the project.
Chief Architect	Responsible of the overall design of the system. Runs also the workshop design sessions with the team. Makes also the final decisions on all design issues. This role can be divided into the roles of domain architect and technical architect if necessary.
Development Manager	Leads daily development activities and solves any conflicts that may rise within the team. Handles also the resourcing problems. Tasks of this role can be combined with the roles of the chief architect or project manager.
Chief Programmer	An experienced developer, who participates in the requirement analysis and design of the projects. Is responsible for leading small teams in the analysis, design and development processes of the new features. Selects also the features to be developed in the next iteration from the feature sets and identifies the classes and class owners that are needed in the feature team during that iteration.

Class Owner	Works under the guidance of the chief programmer, designing, coding, testing and documenting the new features. Is responsible for the development of the class that he has been assigned to own. Class owners form the feature teams.
Domain Expert	A user, a client, a sponsor, a business analyst or a mixture of these. Possesses the knowledge of the real world domain, e.g. how the software requirements should perform. They pass the knowledge to the developers to ensure that a competent system is delivered.
Domain Manager	Leader of the domain experts. Resolves the arguments that may rise within the ranks of the experts.
Release Manager	Controls the progress of the process by reviewing the progress reports from the chief programmers and by having short progress meetings with them. Reports the progress to the project manager.
Language Lawyer / Language Guru	A team member who possesses a thorough knowledge of a certain programming language or technology. Particularly important role when the development team has to work with some technology that is new to them.
Build Engineer	Responsible for setting up, maintaining and running the build process. Manages the version control system and publishes documentation for it.
Toolsmith	Builds tools for the development, test and data conversion teams in the project. May also be working with setting up and maintaining of the databases and Web sites for the project.
System Administrator	Configures, manages and troubleshoots the servers, workstations and development and testing environments that are needed in the project. May also take part in the productionization of the system being developed.

Tester	Verifies that the system will meet the requirements of the customer. Testers may be working in an independent team or as a part of the project team.
Deployer	Converts the existing to a format required by the new system. Participates also in deploying the system. May be working in an independent team or as a part of the project team.
Technical Writer	Prepares the user documentation. May form an independent team or work as part of the project team.

3.3.2 Practices

FDD includes a set of so called “best practices” which are not new as such but developers of the method claim that the specific mix of the practices makes them unique.
Domain object modeling is a technique to explore and explain the domain of the problem. The outcome is a framework where the features can be added.

Developing by feature is a practice where development and progress tracking is done with a list of small functionally decomposed and client-valued functions.

Individual Class Ownership means that each class in the system has a predefined owner who is responsible for the consistency, performance and conceptual integrity of the class.

Feature teams are small, dynamically formed implementation teams.

Inspections are used to catch the defects in for example designs or codes.

Regular builds are in use in the FDD project. The practice ensures that there is always a running, demonstrable system available. Regular builds form a baseline on top of which the new features can be added.

Configuration management is used to enable identification and historical tracking of the different versions of the source code and other files.

Progress reporting is done by reporting completed work to all necessary organizational levels.

FDD states that all the above practices have to be in use to comply with the FDD development rules, although the project team can adapt them according to their level of experience.

3.4 Dynamic Systems Development Method

Dynamic Systems Development Method (DSDM) was developed in the United Kingdom in the mid-1990s. It can be seen as an extension of rapid application development practices. The DSDM features the best-supported training and documentation of any Agile software development ecosystems, at least in Europe. (Highsmith, 2002)

DSDM is a framework based on the best practices and lessons learnt gathered by DSDM Consortium members since 1990. The DSDM Consortium is a non-profit, vendor independent organisation which owns and administers the framework. (DSDM, 2007).

DSDM states that more projects fail because of people issues than technology. The framework focuses to help people to work effectively together to achieve the business goals. It is also a technologically independent framework so it can be used in any business or technical environment without tying the users of the method to a particular vendor. (DSDM, 2007)

One fundamental assumption of DSDM is also that nothing is built perfectly first time, but that 80% of the complete solution can be produced in 20% of the time that it would take to build the complete solution. (DSDM, 2007)

DSDM also assumes that all previous steps can be revisited later on, because of the iterative nature of DSDM, so the current step need be completed only enough to move to the next step. It can be finished in a later iteration. The reasoning for this is, that the business requirements are likely to change anyway as the understanding increases, so the further work would have been wasted. (DSDM, 2007)

The framework is based on nine Underlying Principles that are said to enable projects to deliver what the organisation needs when it needs it. The principles are introduced in the chapter 3.4.3 (DSDM, 2007)

3.4.1 DSDM process

In this chapter, DSDM process is introduced according to DSDM Consortium (DSDM, 2007)
DSDM process consists of five phases: Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration and Implementation.

It is not mandatory to have the project lifecycle exactly as described in figure 4; actually it is not expected to meet the requirements of a particular project.

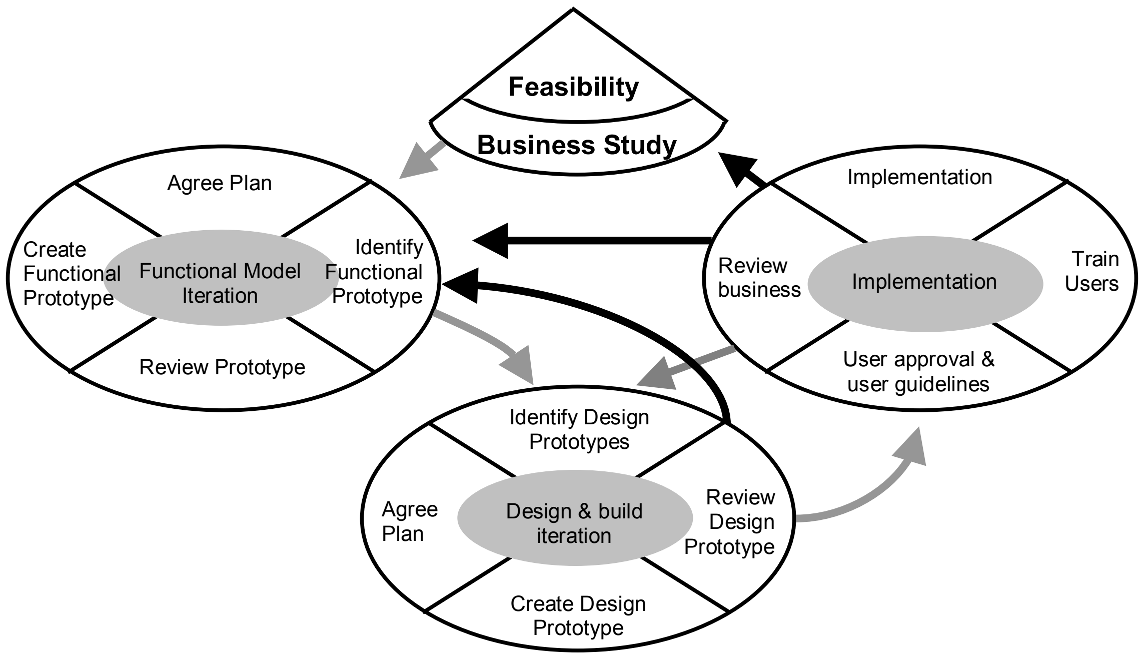

Figure 4 The lifecycle of a DSDM project (DSDM, 2007)

In the **Feasibility Study** phase it is first assessed if DSDM is at all the right approach for the project. If DSDM will be used, the problem is defined, the costs of the project evaluated and also the technical feasibility of delivering a system to solve the business problem. The duration of this phase should be relatively short.

Like the Feasibility Study, the **Business Study** phase it is as short as possible while achieving sufficient understanding of the business requirements and technical constraints to safely move forward.

Each of the requirements identified in the Feasibility or Business study phases has to be prioritized and recorded in the Prioritized Requirements List so that the requirements with the highest priority get implemented first.

During the **Functional Model Iteration** phase the business based high-level processing and information requirements identified during the Business Study are analyzed further and a Functional Model is created. Functional Model consist of software parts, such as functional prototypes which are later integrated to the system if possible, class models and data models and also supporting documentation for the prototypes and a textual description of some system aspects e.g. system start-up and closedown. Functional Model Iteration is the first iterative phase in the process. Continuous testing is also done during this phase.

The **Design and Build Iteration** is the phase where the system is engineered in iterations to a sufficient quality level to be handled to the users. The main output from this phase is a Tested System, which does not necessarily have to fulfil all the requirements, but the requirements agreed for the current increment. Testing is done throughout the phase, so it is not treated as a separate activity.

In the **Implementation** phase the system is transferred from the development environment to the real operational environment. This phase includes training the users, completing the user documentation and creating the Increment Review Document, which summarizes how well the project achieved the short term objectives and requirements.

The **Post-Project** phase includes maintaining the system, which in DSDM can be seen as continuing development. Maintenance can be handled with the same method as the project itself, e.g. starting again from the beginning and going quickly pass the Business Study phase.

3.4.2 DSDM Roles

DSDM specifies lots of different roles and responsibilities for the project. The main concept in DSDM is that a developer should always work with a user in pair. This helps creating strong user/developer partnership. The team can also include two users and one or two developers. (DSDM, 2007)

Other roles in a DSDM project include Executive sponsor, Visionary, Ambassador User, Advisor User, Project Manager, Technical Co-ordinator, Team Leader, Tester and a Scribe. More special roles are Facilitator and various specialist roles. (DSDM, 2007)

3.4.3 DSDM Principles

DSDM sets its foundations in the nine principles. These principles are explained in the following by the DSDM Consortium (DSDM, 2007):

1. Active user involvement is imperative. DSDM is a very user-centred method. If the real users are not closely involved in the development, delays will occur because the developers will make decisions without consulting the users, and the users may feel that the solution is imposed by the developers and/or their management. In DSDM the users are active participants of the development process.

2. DSDM teams must be empowered to make decisions. DSDM teams consist of both developers and users, and they must be empowered to make decisions as the requirements refine and get possibly changed. It must be realized that certain levels of functionality, usability etc. are acceptable without the frequent consultation of the higher management.

3. The focus is on frequent delivery of products. The work of a DSDM team focused on delivering products in an agreed period of time. These are not complete solutions, but just iterations towards the full product. This causes the team to select the best possible solution that can be achieved in the given timeframe. The periods of time are kept short so it is easy to decide which activities are needed to make the product.

4. Fitness for business purpose is the essential criterion for acceptance of deliverables. DSDM focuses to deliver the necessary functionality in the given time. The system can be more rigorously engineered later on, if agreed so. Traditionally the focus has been on fulfilling the documented requirements, even though the preset requirements are often inaccurate.

5. Iterative and incremental development is necessary to converge on an accurate business solution. DSDM allows systems to grow incrementally, so developers can fully use the feedback from the users. Also partial solutions can be delivered to satisfy immediate business needs.
When rework is not explicitly recognized in the development lifecycle, it may be difficult to return to a previous step because of the controlling procedures. Rework is built into the DSDM process so it is easy to go back to a previous step, which speeds up the implementation.

6. All changes during development are reversible. To control the evolution of all products (documents, software, test products, etc.), everything must be in a known state at all times. So the configuration management must be all-pervasive. Backtracking is a feature of DSDM. The ability to reverse changes is limited to current increment.

7. Requirements are baselined at a high level. This means “freezing” and agreeing the purpose and scope of the system at a level that allows investigating of what the requirements mean. More detailed requirement baselines can be introduced later in the development, but the scope should not change significantly.

8. Testing is integrated throughout the lifecycle. Testing is not treated as a separate activity, but is integrated to the development process. During the development the system is reviewed and tested by developers and users incrementally to validate that the development is going to the right direction, both technologically and business wise. In the early phases of DSDM the business needs and priorities are validated and later testing focus shifts towards verifying that the system functions correctly and efficiently.

9. A collaborative and co-operative approach between all stakeholders is essential. In DSDM the low level requirements are not necessarily fixed when the developers start their work. This requires that the short term direction for the project must be decided quickly without recourse, so all stakeholders have to have a collaborative and co-operative attitude.

4 AGILE METHODS IN SMALL PROJECTS

In the previous chapters some agile methods were presented that could possibly be applied to small software projects. In the following, these agile methods are analyzed in order to see how well they would fit into a project which consists of:

- a “black box” project with predefined requirements and time frame
- an offsite customer organization
- a project manager
- a developer

The major asset missing from this kind of project arrangement is the team dynamics and synergy of multiple developers, users and testers working in the same team, which is something most of the agile methods aim to harness.

Also the user participation is usually emphasized when using agile methods. In the case project of the thesis the customer is offsite and cannot participate so much to the development work. But the communication is possible using telephone and email and should help in the questions that will arise during the project. The project arrangement is not the most common for agile methods, so it will be interesting to see how the agile methods perform.

The following agile methods are analyzed in this chapter by finding practices or parts of the process that cannot and those that can be used in very small projects. All the possible practices will be listed in the end of the chapter.

4.1 Extreme Programming (XP)

4.1.1 Pros

The basic process structure (life cycle) of XP looks like it could adapt also to very small projects also. However, when having only one developer, pair programming and continuous review practices have to be dropped out.

Applicable practices include planning game, small/short releases, Metaphor, simple design, test driven development, refactoring, continuous integration, 40 hours week, coding standards and just rules.

4.1.2 Cons

Pair programming, that is very important practice in XP, cannot be applied to one-developer-projects. Customer collaboration is also not as strong as XP suggests when the customer is working offsite.

Collective ownership of code cannot be harnessed with one developer and also design/code reviews require an extra/external resource. Testing of the software is also done by the same person that is writing the code; all the possible problems may not be found because the developer knows how the application works.

4.2 Scrum

4.2.1 Pros

Basic process structure of Scrum could also fit small projects; creating small working releases with sprints. In the case project, the requirements are gathered; they could be prioritized to form the initial product backlog list

Daily scrum meetings with the scrum master (project manager) could be useful. 30 day release cycles would also fit in the case project

4.2.2 Cons

Customer is offsite and tight customer collaboration is not possible. Also improved team dynamics enabled by Scrum are not available in one-developer-project.

4.3 Feature Driven Development

4.3.1 Pros

Basic structure of the FDD process is applicable to small projects; plan by feature, design by feature and build by feature can be applied. Individual code ownership is also easy to arrange, when there is only one developer. Regular builds of the software are also possible.

4.3.2 Cons

Certain parts of the process seem to be targeted to the bigger projects; Domain object modeling is done with group of domain experts and presentations are held for the development teams. FDD describes also many different roles and responsibilities for project personnel. Design and code inspections require another developer or external reviewer.

4.4 Dynamic Systems Development Method

4.4.1 Pros

Although the DSDM process is a bit heavier than for example XP and Scrum with lots of pre-studies etc. all the phases of the process seem possible to be used even in very small projects with some tailoring. Case project is also time constrained as DSDM requires.

4.4.2 Cons

DSDM requires and is based on active user involvement which is not possible in the case project, and DSDM also defines many different roles and responsibilities for project team members.

4.5 Practices suitable for small projects

In the table 4 are listed all the agile practices that were found feasible in a small project described in the beginning of the chapter. Descriptions of the practices can be found from the chapter 3.

Table 4 Feasible agile practices

Method	Practice	Comments and limitations
XP	Planning game	Requires interaction with the customer.
	Small/short releases	No known limitations.
	Metaphor	Requires interaction with the customer.
	Simple design	No known limitations.
	Test driven development	Customer writes functional tests.
	Refactoring	Refactoring must be done by the same person that wrote the code; not so effective.
	Continuous integration	May require too much time to implement if time frame is limited, but otherwise a good practice.
	40 hours week	No known limitations.
	Coding standards	Will provide more value in a bigger team, but also useful in a one-man-project.
	Just rules	Will provide more value in a bigger team.
Scrum	Product Backlog	Can be used in the small projects also, as long there is a Product Owner managing the Product Backlog.
	Effort Estimation	No known limitations.
	Sprint	No known limitations.
	Sprint Planning meeting	Usually requires interaction with the customer.
	Sprint Backlog	No known limitations.

	Daily Scrum meeting	No known limitations.
	Sprint Review meeting	Can be used, as long there is a customer and users involved in the meetings.
FDD	Domain Object Modeling	The developer must act as a domain expert and build the overall model.
	Developing by Feature	No known limitations.
	Individual Class Ownership	Suits a small project very well.
	Regular Builds	Requires some effort to implement, but can be a valuable practice also with one developer. Can include running all module and unit tests and building all the components.
	Configuration Management	Enables tracking of changes made to the code or documents.
	Progress Reporting	No known limitations.
DSDM	The focus is on frequent delivery of products	Is used for getting accurate feedback from users, requires testing efforts from the customer.
	Fitness for business purpose is the essential criterion for acceptance of deliverables	No known limitations.
	Iterative and incremental development is necessary to converge on an accurate business solution	Makes possible to react to changing requirements.
	All changes during development are reversible	Requires configuration management.
	Requirements are baselined at high level	No known limitations.

	Testing is integrated throughout the lifecycle	No known limitations.
	A collaborative and co-operative approach between all stakeholders is essential	No known limitations.

5 CASE: FURTHER DEVELOPMENT PROJECT OF BLUETOOTH PROXY APPLICATION

5.1 Introduction

This project was taken as an example of a really small development project, consisting of only four stakeholders: client, project manager, developer and the part time technical aid person.

The aim of the thesis is to study how agile methods or certain parts of them can be used in small projects and do they give any advantages over traditional software development processes. Also the disadvantages or problems will be documented.

5.2 Symbian and S60 mobile operating systems

The software application in the project will be developed for the S60 platform which is briefly presented in this chapter. Symbian OS is an operating system designed for mobile devices, such as mobile phones. Symbian was formed from Psion Software in 1998 by Ericsson, Motorola and Nokia to provide a common standard and to enable mass marketing of the new generation of wireless devices.

From the very beginning, the goal of Symbian was to create an operating system and software platform for advanced mobile phones; so called smartphones (figure 5). EPOC operating system developed by Psion formed the foundations of Symbian OS. EPOC was a modular 32-bit operating system with multitasking capabilities and it was designed for mobile devices. (Digia, 2003)

Symbian develops and licenses the Symbian OS containing the base (microkernel and device drivers), middleware (system servers), a large set of communication protocols and a test user interface. Licensees develop their own user interfaces to suit their needs, and they can also license their user interface and application set on top of the Symbian OS to the other Symbian licensees – as Nokia has done with the S60 platform. (Digia, 2003)

Figure 5 Modern S60 mobile device, Nokia N95 (Nokia, 2007)

S60, formerly known as Series 60, is a smartphone platform that runs on Symbian OS. S60 can be seen as a user interface for Symbian OS. S60 is primarily developed by Nokia and licensed to other manufacturers, such as Lenovo, LG, Panasonic and Samsung. Symbian and S60 allows user to install new applications to the device, so the system can be expanded after the purchase of the device. (Digia, 2003)

5.3 Bluetooth Proxy for S60

The software which is to be developed further is the Bluetooth Proxy application, which can be used instead of GPRS, Universal Mobile Telecommunications System (UMTS) etc. over the air data transmission techniques to establish TCP connections to a computer. It is used in for example testing Java environment implementation in a S60 enabled mobile phone. The reasons to use Bluetooth Proxy instead of GPRS are that it does not use the mobile network, so it does not cost anything and it is also faster than GPRS or UMTS.

The Bluetooth Proxy currently transfers the normal TCP and Hypertext Transfer Protocol (HTTP) traffic over Bluetooth to PC-side proxy, which then relays the traffic to the application running on the same PC or to the internet, as illustrated in the figure 6.

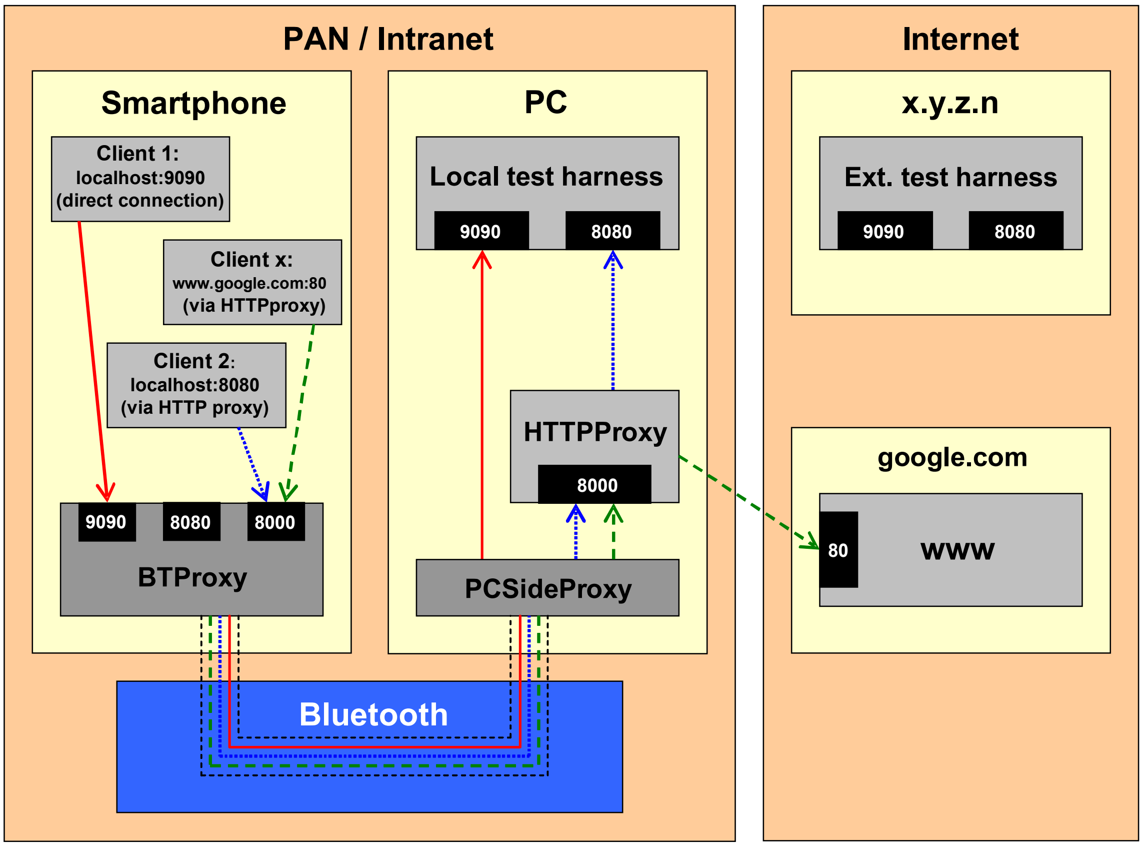

Figure 6 Functionality of the Bluetooth Proxy

The application consists of two main components, the phone side proxy application and the PC-side proxy application. PC-side proxy includes the router program that handles the normal TCP traffic and a separate freeware HTTP proxy, TinyHTTPProxy developed by Suzuki Hisao.

The phone side proxy application is written in Symbian C++, and the PC-side proxy in Python. Python was chosen for its simplicity and portability.

5.4 The project

5.4.1 Requirements included in the further development project

The project includes the following high level requirements, which are also represented in the figure 7.

- two way TCP-connection support; connections can be created from the PC-side also
- UDP-protocol support

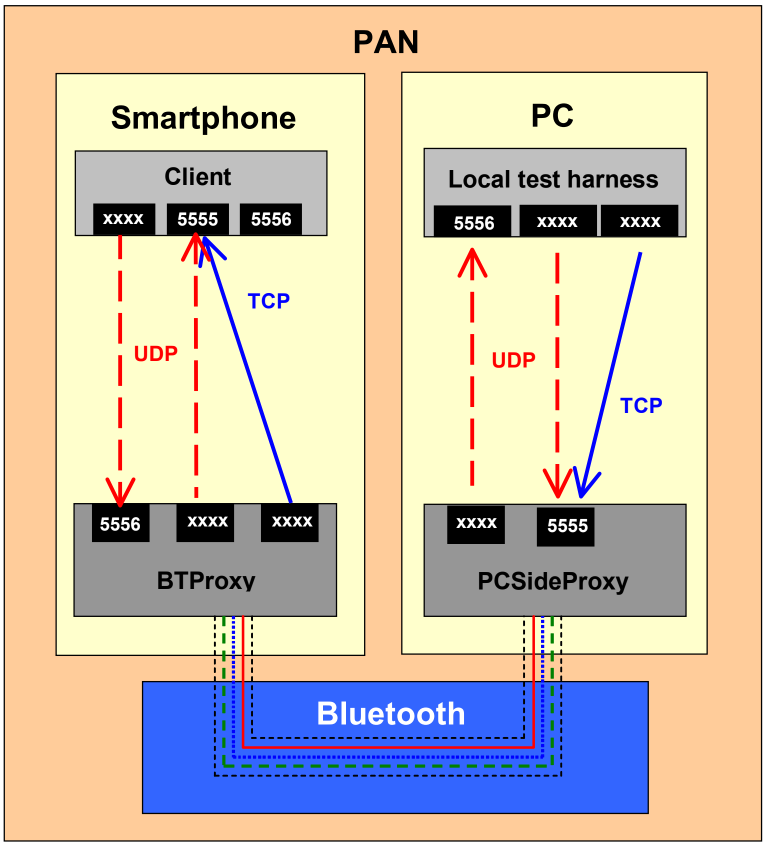

Figure 7 New features of the Bluetooth Proxy

Implementing the new features in the BT-proxy –protocol means also that the protocol handling has to be redesigned to support a new protocol (UDP) and new TCP commands like opening and closing TCP connections.

5.4.2 Stakeholders

Following roles and persons are involved in this project (table 5).

Table 5 Stakeholders in the project

Role	Responsibilities
Customer	Requirements and time frame
Project manager	Customer relations, requirement and time frame negotiations, project monitoring and decisions.
Developer	Time table, design, implementation, testing, documentation and deliveries.
Technical Aid	Provides technical support in design and development issues (previous developer of BT-proxy)

5.5 Agile practices for the project

All the agile methods described earlier define at least one team of developers by default, so some tailoring is required to fit them into a one-man-project. It would have been challenging to implement all the possible agile practices presented in the chapter 4.5, so some qualification was necessary.

Here are described the arguments why certain agile methods and practices were chosen for this project. Practices were analyzed and selected from the list of possible practices for small projects, described in table 6.

Scrum method was chosen to be the overall method for the project and tailored to fit in the project. Scrum was chosen because it contains so many practices that can be implemented even without a big project team and does not have any practices that are clearly designed for bigger projects.

Chosen practices of Scrum include:

- High level design and planning according to the predefined requirements
- Product backlog list
- Separate Sprint cycles for the separate features, so that implementation will be done feature by feature. When a feature is ready, it will be delivered to the customer immediately.
- Daily scrum meetings with the project manager

Also the test driven development practice from XP was chosen to be used. This involves writing the tests before the actual implementation and filling in the implementation so that the tests will pass. This would suit the distributed environment of the Bluetooth Proxy application, because debugging an application which communicates with its counterpart over Bluetooth can be a tedious task.

Extreme Programming in whole was considered also, but the lack of pair programming possibilities, collective code ownership and the fact that XP suggests very strong customer collaboration seemed bigger downsides than the missing team dynamics of the scrum method.

Feature driven development contains many practices that are targeted to bigger projects; domain object modeling that is suggested to be done by domain experts and many roles and responsibilities, so it was not chosen for this project. DSDM was not selected because it requires tight collaboration with the users, which was not possible in the case project and also specifies lots of different roles and responsibilities.

Table 6 Agile practices for the project

Method	Practice	Notes
Scrum	High level design and planning according to the predefined requirements	
	Product backlog list	
	Separate Sprint cycles for the separate features, so that implementation will be done feature by feature. When a feature is ready, it will be delivered to the customer immediately.	
	Daily scrum meetings	With project manager and developer
XP	Test driven development	Tests are written before the actual implementation.
	40 hours week	To keep working hours sensible and productivity high.
	Coding standards	Company has coding conventions for S60 programming.
FDD	Configuration Management	

5.6 Project execution

Project started with a high level design phase for the features. A preliminary product backlog list was created from the list of tasks that was compiled earlier by the previous developer and the customer. This phase included asking lots of questions from the previous developer who also reviewed some of the main design ideas. The initial schedule for the project which is presented in table 7 was agreed in the high level design phase also.

Table 7 Initial schedule for the project

Week	Task
wk 45:	Requirement gathering
	Set up the development environment
	Preliminary timetable and planning
wk 46	Refactoring of the protocol module
	o design
	o update unit tests
	o implementation
wk 47	Refactoring of the protocol module
	o implementation
	o run tests
	Two way TCP traffic
	o design
wk 48	Two way TCP traffic, phone side
	o design
	o unit tests
	o implementation

wk49	Two way TCP traffic, PC-Side
	o design
	o implementation
	o testing
	Testing with real customer environment
wk 50	UDP support
	o design
	UDP support, phone side
	o design
	o update unit tests
wk 51	UDP support, phone side
	o update unit tests
	o Implementation
	UDP-support, PC-side
	o design
	o update unit tests
	o Implementation
wk 52	UDP-support
	o Implementation
	Testing with real customer environment
	Update documentation
	- design docs
	- user guide

Daily scrum meetings were held regularly every day at 9 o'clock in the morning. Participants included the developer and the project manager. These meetings felt a bit stressing at first, but after realizing that they really help to solve possible problems right away, they started to feel like a good idea.

After the high level design phase was completed and a rough idea of the high level architecture was documented, a more detailed design was created for the first feature. The first of the two features was adding support for the two-way TCP connections.

Next phase was to implement the two-way TCP connections –feature. The implementation started with writing unit and module test cases for the new features, and adding the real implementation to make these tests pass. This proved to be a very effective practice in this project, because the project included communication over Bluetooth and following a predefined protocol in this communication. Some problems were encountered when certain classes seemed impossible to test, and the implementation was done without testing it first. This caused some errors to slip in to the implementation and the errors were corrected with time consuming debugging between the phone and the PC counterpart. Later on it turned out that testing those classes would have been possible. The debugging caused the project to slip from the initial schedule for about a week, and the next tasks were moved ahead accordingly.

After the first feature was finalized the first delivery was made to the customer. The customer started testing the new feature right away and some errors were corrected. The communication with the customer was conducted via email and telephone.

While the customer started experimenting with the new feature, the design and implementation of the second feature, UDP protocol support, was initiated. A more detailed design was done for the feature and the implementation was done the same way as with the first feature; by writing the test cases first and then filling in the implementation.

During the second implementation phase, some problems were encountered when testing the implementation with the customer test environment. The configuration of the test environment was difficult because there was no prior experience on that particular case within the company.

Finally the delivery of the second and last feature of the project was made, about two weeks behind the initial schedule.

5.7 Project success

The main objective, the additional functionality for the BT Proxy software was accomplished and the software was taken into use by the customer. The implementation itself did not go as smoothly as it was thought in the beginning and caused the project to slip a few weeks from the predefined timeframe. This was a setback, and the reasons for the delay are analyzed in the following.

New features were first tried to test with customers test suites. This left too much room for configuration errors with the test suites. Also the insufficient amount of knowledge about the functionality of the test cases was a problem. These problems combined caused some delays in the schedule of the project. It would have been a better choice to make particular test applications that test two way TCP connection and UDP connections. They were actually done after the trials with the test suites.

Not enough design before implementation in certain cases. This caused for example a situation, where in the middle of UDP implementation it was realized that it is not possible to listen to a UDP port and send UDP packets to the same port; doing so causes a packet to loop within the proxy application.

“Test first” practice was not followed at all times. In a few cases, wrong assumptions were made that it is not possible to test the certain functionality with unit tests. Also the test applications for the connection testing which were implemented on the later part of the project would have helped a lot if they had been available during the implementation of the UDP and two way TCP features.

However, these problems can be seen more of as normal software development mishaps rather than caused by the use of agile practices.

5.8 Agile practices in action

Here are described how each of the applied agile practices succeeded.

5.8.1 High level design and planning

In this case project the high level planning was not so difficult, because the requirements had been gathered before the project started. The requirements were added to the product backlog list, and the list was prioritized to see what should be done first. Also high level architecture for the new features was drawn. Some of the design ideas were also reviewed by the previous developer of the Bluetooth Proxy application.

The high level design and planning phase itself worked fine in the project and proved to be effective. After the project had been finished, it became obvious that more time spent in the design and planning saves time in the implementation phase. Good examples of this were the problems in the UDP implementation phase, described in the chapter 5.7.

5.8.2 Product backlog list

All the predefined and new requirements were gathered to the product backlog list during the project and were also updated when new information became available. This appeared as a worthy practice, because the changes in the requirements were immediately visible to the developer and the project manager. No problems were encountered with this practice.

5.8.3 Separate Sprint cycles for the separate features

This practice worked well in many aspects; the development was consistent, customer was happy to receive a working product relatively fast when the first feature was ready and the customer could also start testing the new feature right away. It was also easier to focus on the implementation, when the goal for the sprint was fixed.

One thought that came up after the project was that could shorter sprint cycles have helped when designing and implementing a feature? The implementation of a feature could have been divided into two separate sprint cycles. However, with that approach it would have been hard to deliver an executable product increment after the sprint execution.

5.8.4 Daily scrum meetings

Daily Scrum meetings were held with the developer and project manager every morning at 9 o'clock. They kept the project manager on track with the project and the problems that arose during the sprints could be solved quickly.

The only downside was that the developer felt the constant monitoring a bit uncomfortable at first but it eased up when he realized that it really helped in solving the problems. Overall, it was found to be a very effective way to enhance communication in the project.

5.8.5 Test driven development

The practice of writing the module and unit tests before the actual implementation proved to be really effective especially in the distributed environment of the case project. A set of tests could be run for the application after altering the code to assure that the change had not broken something in the application.

As reported in the chapter 5.7, the more accurate following of this practice would have saved time during the implementation. Overall, this was seen as a very good practice in the case project, and did not cause any trouble.

5.8.6 40 hours week

The working hours for the developer were kept strictly within the limit of 40 hour per week, no overtime was allowed. A few times, when the initial effort estimates were exceeded, there was an urge to do longer days. This would have most likely caused decrease in productivity on the next day.

This seems to be a valuable practice, although it was not tested what would have happened if the developer had done for example 60 hours per week.

5.8.7 Coding standards

Coding standards and company coding conventions were in use during the whole project. This made it easy for the developer to write code that other people in the company can read easily and also to free the developer from wondering how to write comments, how to indent blocks and so on, so it actually saves time from the implementation phase.

5.8.8 Configuration Management

The initial version of the software that was to be developed further in the case project, was stored in a configuration management system already, so the same system was used in the case project.

Because there was only one developer, the benefits of the configuration management included keeping the source code safe and backed up, change history for the files and the possibility to label different configurations of the source code, for example releases.

If there would have been more developers involved in the project, one benefit of the configuration management would have been also preventing two or more people changing same files and overwriting each others changes.

6 DISCUSSION

After the agile practices for the project were chosen and the implementation part of the project began, some questions and improvement ideas emerged. These issues, recommendations for the future and further research possibilities are discussed in this chapter.

First thing that came up was that it would have been possible to follow almost the whole Scrum process in the project, although it would have required more effort to break away from the usual project practices.

The length of the sprints was also one issue which was considered; shorter sprints would maybe made it easier to plan tasks for the sprint, but on the downside, it would not been possible to deliver an executable product increment after each sprint. The use of shorter sprints could be tested in the future projects.

A tool for following the project progress and the state of the product backlog, for example free of charge ScrumWorks Basic (Danube, 2007), would have been a good addition to the project. It helps also when updating the effort estimates and planning the Sprints.

The results of this study were positive and the developer found the use of the agile method and practices enjoyable, which corresponds to the results published in the agile method surveys introduced in the chapter 2.4. To get broader scale results of the use of agile methods and practices in small projects, further research work, as presented in the chapter 6.2, has to be conducted.

6.1 Recommendations for the future

On the basis of this study, it can be stated that agile practices can be useful also in the small, and even “black box”-projects. Because this study consisted of only one project, it is not possible to say agile methods and practices provide good results in every small project, but it was a good start.

Although the study was made with just one project, the results were very promising and agile methods and practices should be tested and taken into use in other small projects also to get some broader scale results. Performance data should be gathered from the projects, also from the non-agile ones, to make further analysis and comparison with other methods.

SYSOPENDIGIA Finland Ltd has a dedicated software development process, which can be used as a base for the development process in new projects. The improvement of the company’s software development process is a continuous task and the results of this study should be used also as material for this development work.

6.2 Further research possibilities

Comparing the performance of the software development methods is always hard, because the projects are unique. Every time a project is executed, there are some variables that differ from the previous projects. However, there are some values that could be measured.

To obtain more information about the performance of the agile methods and practices in small projects, some statistical data about for example productivity and quality aspects should be gathered and the sample size of the projects should be significant.

Performance data should be gathered also from the projects that are using traditional software development methods to be able to compare them with the agile projects.

Also using other possible agile methods in the small projects could be experimented. It would be possible to try all the methods described in chapter 4, with some customization and tailoring.

7 CONCLUSIONS

The aim of this thesis was to test if agile methods and practices would bring value even to the smallest software projects, which consist of only one developer. According to the findings in the case project, agile practices were useful in the one developer project. However, because the study contained only one development project, broader scale results cannot be stated yet.

It was found that the benefits of agile methods in small projects depend on the project environment, on the selection of agile practices and their suitability to the problem at hand and also on the attitude of the team members and customer towards agile methods. But with appropriate tailoring agile practices can be useful. Some very useful agile practices were found that can be applied to a project, regardless of the size of the project. These practices were listed and documented in the table 4.

In this test run there was no obvious possibility to test how a traditional software development method like the waterfall model would have performed against the agile method and practices, but the goal was to test how the agile practices affect the work in a project like this, not so much to compare them with the traditional methods.

According to this study also agile methods and practices should be taken into the consideration when choosing the right software development method or practices for a small, even one-developer project.

REFERENCES

(Abrahamsson, 2002)	Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. 2002. Agile software development methods. VTT Publications 478. ISBN 951-38-6010-8
(Beck et al., 2001)	Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R., Mellor, S., Schwaber, K., Sutherland, J. and Thomas, D. 2001. Manifesto for Agile Software Development. http://agilemanifesto.org, checked 6.5.2007.
(Beck, 1999)	Beck, K. 1999. Embracing Change With Extreme Programming. IEEE Computer, volume 32, issue 10, Oct. 1999: pages 70-77.
(Beck, 2005)	Beck, K., Andres, C. 2005 Extreme Programming Explained: Embrace Change – Second Edition. Pearson Education, Addison-Wesley. ISBN 0-321-27865-8
(Danube, 2007)	Danube Technologies, Inc., 2007. ScrumWorks Basic. http://danube.com/scrumworks, checked 8.5.2007.
(Digia, 2003)	Digia Inc., 2003. Programming for the Series 60 platform and Symbian OS. John Wiley & Sons. ISBN 0-470-84948-7.
(Drobka et al., 2004)	Drobka, J., Noltz, D., Raghu, R. 2004. Piloting XP on Four Mission-Critical Projects. IEEE Software, December 2004. Pages 70-75. 0740-7459/04.

(DSDM, 2007) DSDM Consortium, 2002-2007. DSDM Public Version 4.2. www.dsdm.org, checked 14.4.2007.

(Fitzgerald et al., 2002) Fitzgerald, B., Russo, N. L., Stolterman, E. 2002. Information systems development: Methods in action. MC Graw-Hill Education, ISBN: 007709836 6.

(Highsmith, 2002) Highsmith, J., 2002. What Is Agile Software Development? STSC Crosstalk, The Journal of Defense Software Engineering, October 2002 issue.

(Haungs, 2001) Haungs, J., 2001. Pair programming on the C3 project. Computer, Vol. 34, Issue 2. IEEE Computer Society. Pages 118-119.

(Marciniak, 2002) Marciniak, J. J..,2002. Encyclopedia of software engineering, Vol 2. Wiley, New York.

(McConnell, 1997) McConnell, S. 1997. Software Project Survival Guide, Microsoft Press. ISBN: 1-57231-621-7.

(Nokia, 2007) Nokia, 2007. Press photo archive, www.nokia.com, checked 30.4.2007.

(Palmer and Felsing, 2002) Palmer, S. R., Felsing, J. M. 2002. A Practical Guide to Feature-Driven Development, Upper Saddle River, NJ, Prentice-Hall.

(Reifer, 2002a) Reifer, D. J. 2002. Let the numbers do the talking. STSC Crosstalk, March 2002. Pages 4-8. 0740-7459/02.

(Reifer, 2002b) Reifer, D. J. 2002. How Good Are Agile Methods? IEEE Software, July/August 2002. Pages 16-18. 0740-7459/02.

(Russ and McGregor, 2000)
Russ, M. L., McGregor, J. D. 2000. A Software Development Process for Small Projects. IEEE Software, September/October 2000. Pages 96-101. 0740-7459/00.

(Schwaber, 2002) Schwaber, K., Beedle, M. 2002. Agile Software Development With Scrum. Upper Saddle River, NJ, Prentice-Hall.