TAMPERE
POLYTECHNIC

PROFESSIONAL MASTER THESIS

FINAL THESIS

EVALUATING AGILE METHODS AND THEIR IMPLEMENTATIONS

Minna Väänänen

Information System Competence
May 2008
Supervisor: Paula Hietala

TAMPERE 2008

TAMPEREEN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

PROFESSIONAL MASTER'S DEGREE

Author: Minna Väänänen

Degree Programme: Master Degree Programme in Information System Competence

Thesis title: Evaluating Agile methods and their implementations

Month and year: May 2008

Supervisor: Paula Hietala **Pages:** 81

ABSTRACT

All kind of Agile software development methods has been found favour with software companies. Clear paradigm shift is happening at the moment in the software development world. The old and proverbial waterfall method has been noticed to contain big defects and that is why many new software development methods have been developed to solve the known problems.

This thesis has been made to the software development team which is going to switch from traditional method to an agile method. Earlier research made to the team recommended that an agile method would solve best the difficulties of the current software development process. The purpose of this thesis is to study which agile method would suit best to the projects of the team and how this method could be adopted.

In theory part of the thesis different agile methods are explained and methods are compared with each other. In the second part five persons using agile has been interviewed. The purpose of the interview part is to figure out what agile method has been used and why, what has been achieved by using agile and how the method has been adopted. In the third part of the thesis the best agile method for the team and the projects is proposed. The agile best practices for future software development process are suggested also. In addition the way how agile method should be taken in use has been gone through and what is going to change for example in the testing of the software.

The result of the thesis is concrete proposal for the suitable agile method for the team. It contains proposal of different practices, an example how the agile method could be take in to use by pilot project, and how the results and learning from the pilot project can be exploit in the real agile projects.

Keywords: Agile Software development Program

T A M P E R E E N
AMMATTIKORKEAKOULU

Y L E M P I A M K - T U T K I N T O

Tekijä: Minna Väänänen

Koulutusohjelma: Tietojärjestelmäosaaminen

Opinnäytetyön nimi: Ketterien menetelmien arviointi ja niiden käyttöönotto

Työn valmistumis-kuukausi ja -vuosi: Toukokuu 2008

Työn ohjaaja: Paula Hietala

Sivumäärä: 81

TIIVISTELMÄ

Erilaiset ketterät ohjelmistokehitysmenetelmät ovat saavuttaneet viime vuosina suuren suosion ohjelmistoyrityksissä. Selvä paradigman muutos on siis tapahtumassa ohjelmistokehitysmaailmassa. Vanhassa ja yleisesti käytössä olevassa vesiputousmallissa on todettu olevan suuria puutteita ja tämän vuoksi viime vuosina on kehitetty lukuisia uusia menetelmiä ratkaisemaan vanhojen mallien ongelmat.

Tämä opinnäyteyö tehdään ohjelmistokehitystiimille, jonka tarkoituksena on siirtyä ketterän menetelmän käyttöön lähitulevaisuudessa. Työn taustalla on tutkimus, jossa todettiin ketterien menetelmien ratkaisevan parhaiten tällä hetkellä tiimin ohjelmistokehityksessä olevat puutteet. Työn tarkoituksena on tutkia, mikä ketterä menetelmä sopisi parhaiten tiimin ohjelmistoprojekteihin ja miten menetelmä saataisiin parhaiten käyttöön.

Teoriaosuudessa tutustutaan erilaisiin ketteriin menetelmiin sekä verrataan niitä toisiinsa. Haastatteluosuuden tarkoitus oli selvittää viideltä ketterää menetelmää käyttävältä henkilöltä, mikä ketterä menetelmä oli valittu käyttöön ja miksi, mitä ketterän menetelmän käytöllä on saavutettu ja miten se on otettu käyttöön. Kolmannessa osuudessa ehdotetaan perusteluineen tiimille sopivin menetelmä sekä mitä ketteriä käytäntöjä kannattaa ottaa mukaan tulevaan ohjelmistokehitysprosessiin. Lisäksi käydään läpi tapa, jolla ketterä menetelmä kannattaisi ottaa käyttöön ja mitä sen käyttöönotto tulee muuttamaan esimerkiksi testauksessa.

Työn tuloksena on konkreettinen ehdotus tiimille sopivasta ketterästä menetelmästä erilaisine käytäntöineen sekä ehdotus siitä, kuinka ketterä menetelmä voidaan pilotoimalla ottaa käyttöön ja kuinka pilotista saatuja tuloksia ja oppimisia voidaan hyödyntää varsinaisissa ensimmäisissä ketterissä projekteissa.

Avainsanat ohjelmisto ohjelmistokehitys ketterät menetelmät

ABBREVIATIONS

Agile software development	Framework for software developing. In agile, software is developed through the iterations.
ASD	Adaptive Software development. Agile method.
Crystal	Family of agile methodologies.
Daily meeting	Scrum meeting, daily stand-up meeting. Project status meeting, which is arranged every day.
DSDM	Dynamic Systems Development Method. Agile method.
FDD	Feature Driven Development. Agile Method.
IID	Iterative and Increment Development
Iteration	In agile, the software is developed in small cycles (iterations).
Product backlog	Prioritized list of all the requirements that the system should include and address (functionality, features and technology).
Product owner	Product owner represents the voice of the customer. Manages and owns the project. Gathers up the requirements (Product backlog) and establishes and updates the schedule.
RUP	Rational Unified Process. Agile method.
Scrum	Agile method.

Scrum Master	Helps the Product Owner and teams to notice and remove the problems to deliver the sprint goal.
Sprint	One time boxed iteration in the Scrum method.
Sprint backlog	Detailed document about what and how the team is going to do in the upcoming sprint.
TDD	Test-Drived Development.
XP	Extreme Programming. Agile method.

1 INTRODUCTION

This thesis is executed to software research and development team, which has developed a worldwide favour reached software.

Organization has been grown fast and heavily and the growth of the software's sale still continues. The biggest problems have been noticed to be nowadays the lack of resources, processing of the change requirements in the middle of the projects, controlling the quality (no explicit indicators of quality in use), defective documentation and orientation of the new employees.

The goal of this thesis is to study how agile software development can be introduced and made to work in forthcoming further development projects of the software. Agile software development method is rather new way of developing software, emphasis of the agile development is on doing workable software instead of documentation and on direct communication and short iterations.

Along with the agile software development method for example nature of the testing of the software will change totally. Nowadays there is a massive system testing phase in the end of the project, but in agile method, testing happens continuously in every iteration and there is only a regression testing phase in the end of the project. Also the collaboration with the customers is going to grow. One big change concerns the composition of the groups inside the development team. Roles of the employees are going to change dramatically. Many things are going to change, so a careful plan how the introduction of the method will be performed is needed.

In chapter two the team to whom the thesis is written is introduced. In this chapter the current situation of the team is explained, what software

development method the team is using for the meantime and what are the biggest problems of the current method.

Chapter three introduces the concept of the Iterative Software development, what it is and what features it includes. The agile method is based on the Iterative Software development and therefore it is important to understand what it means to produce workable software iteratively, by using continual small development cycles. Also the term time box is introduced in this chapter, because it is an essential part of the Iterative Software development.

Chapter four covers the main topics of the agile methodology. The main values of the agile software development and the principles behind the agile development concept are explained to get the general view of the method. In this chapter all most famous agile methods are described with their principles and practices and also those agile methods are compared with each other.

The interview research is handled in the chapter five. Five persons were interviewed to gain information on how the agile method works in the practice and what it has brought along to the teams of the interviewees. Intention of these interviews was together with the theory part to produce the opinion about what agile method could be suitable for the team and how the methods should be taken in use.

Agile method recommendation is given in the chapter six. This recommendation is based on the agile method introductions in the chapter three and the interview part in the chapter four. In this chapter possible effects on the organization structure and working methods are considered. Specified agile rules and practices which the team should adopt are introduced.

The way to transit to the agile is handled in the chapter seven. In this chapter the adopting process itself and things what must have been done before the first agile project to get the best profit is considered. Also those things which are good to be ready before the first project are recommended. The pilot project is also introduced in this chapter. In this thesis the pilot project is recommended to get training and experience in agile method before the first actual agile project.

2 CURRENT SITUATION

At the moment, software development process of the team is based on the corporate's own Unified Process method. This UP method is an incremental program development model, which is used in many software development projects in the corporate.

Recently the own UP method has been modified so, that after one component is ready for the testing, the component verification phase takes place. The component verification is done separately for every component. Finally when all the components included into the release are tested, it is time for the release verification. In that phase all the components are merged and the verification covers a regression round for the entire software to ensure that all the new and the changed functionalities, and the whole system after the merge, work fine. After the release verification phase and a minor regression round, software is ready for the customer deliveries or the pilots.

According to the research (Improving the Software Development Process of a Research and Development Team) made inside the team, this software development model does not work anymore. Because of the growth of the software development team (nowadays teams in Finland and in Bangalore, India) and amount of program's features, the development model is out of date and ineffective. There are needs for better control of the development work and improvement of the quality. Most of the quality problems of the software development projects have been noticed at the end of the project in the system testing phase. Despite of these problems in the phase of development, the team has been able to deliver the product with reasonable quality to the customers and the future of the software is promising. The quality of the development process has to be improved before it starts to influence to

the quality of the published program. Until now the problems have been coped with by lengthening the schedule.

Agile software development method (for example Scrum) was recommended in the research. With agile method team can respond quickly to the changing requirements. Method also helps team to develop software more effective way than nowadays. Goal of this thesis is to study what agile method would be suitable for the team and how the method can be introduced.

3 ITERATIVE SOFTWARE DEVELOPMENT

The concept of growing the system via iterations is called iterative and incremental development (IID), but usually it is called simply ‘iterative development’. All agile methods, including most common methods Scrum and Extreme Programming (XP) are based on IID. (Larman 2004: 10-11). Different agile methods are explained later on in this thesis.

In traditional waterfall method the software is developed in several phases, which each follows each other, without opportunity to go back to the previous phases. The planning phase is always in the beginning of the project, all plannings and designs are made before the implementation itself can be started. Testing phase takes place when implementation work is totally finished. That is the reason why errors are found late in the project and the schedule of the project might stretch because of the correction time of the bugs. Also it is hard to accept change requirements, because all plannings are already made in very early phase of the project.

Iterative software development (see figure 1) means that software is build in several iterations in sequence. Each iteration is so called mini-project, which every one is composed of different activities such as requirement analysis, design, programming and testing. (Larman 2004: 10.)

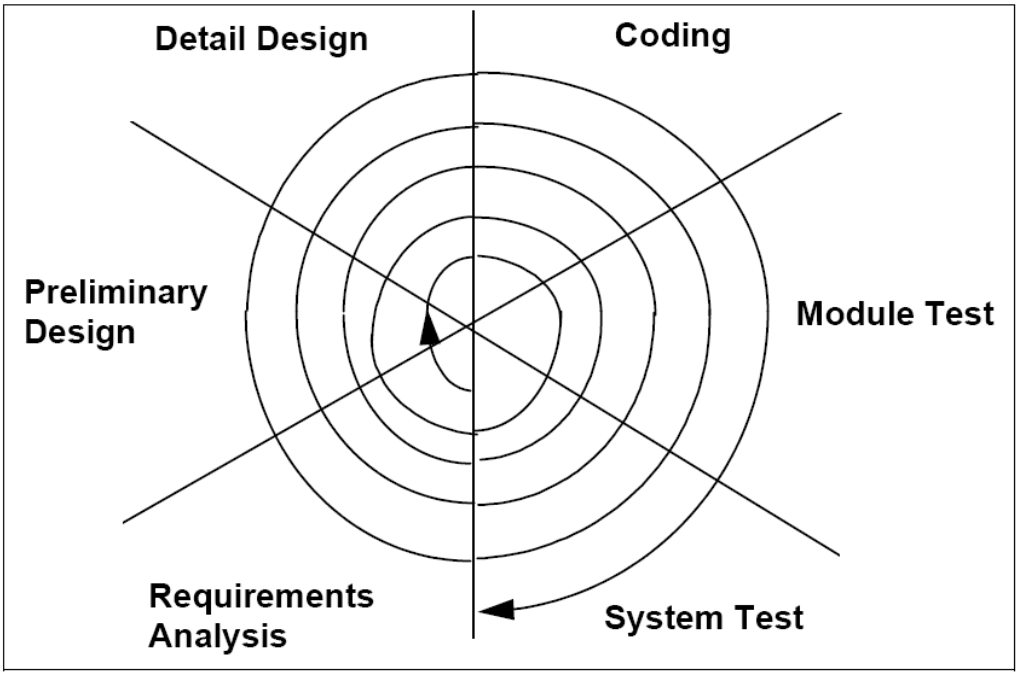

Figure 1: Iterative Methodology (Schwaber: 7)

Every iteration round generates an iteration release, a stable, integrated, and tested, partially complete system. Iteration release is not usually released externally, only the final iteration release, the complete product, is released to the market or the clients. The outcome from the iteration is not a prototype or proof of concept, but a subset of the final system. (Larman 2004: 10-11.)

In principle the output of the every iteration is independent, workable product, which can be delivered to the customer. But in practice, the output of the iteration is so called demo, workable product, which is demonstrated to the customer. That way the customer can see what has been done in the previous iteration and what kind of product he is going to get in the future. Also customer can examine that the product is accordant with customer's needs and he also might want some changes which can be taken into account in the next iteration.

Schedule must be planned carefully in every iteration. Iteration time boxing is the practice of fixing the iteration end date. It is not allowed to change this end date. If it appears that the iteration is behind the

schedule, resolution is reducing the scope (lower priority requests are not carried out) instead of slipping the iteration end date. Time box of iterations do not need to be equal in length. The first iteration can be four weeks, the second three weeks and so on. (Larman 2004: 13.)

Almost every agile method recommends the length of the iteration time boxes. For example the Scrum method recommends that iteration time box is one to six weeks, usually always 30 days. The most important thing is not to follow faithfully different methods' recommendations but clarify what is the most suitable iteration length for the own project and schedule the timetable of the project accordingly. And keep that time.

4 AGILE

The Manifesto of Agile Software Development established four core values (Manifesto for… 2001):

> We are uncovering better ways of developing [products] by doing it and helping others do it. Through this work we have come to value:
>
> *Individuals and interactions over processes and tools*
>
> *Working [products] over comprehensive documentation*
>
> *Customer collaboration over contract negotiation*
>
> *Responding to change over following a plan.*

‘Individuals and interactions over processes and tools’ means that processes provide guidance and support and tools make effectiveness better. However all the processes and tools are worthless and won’t produce results without people having suitable technical and behavioral skills. (Highsmith 2004: 13.)

‘Working software over comprehensive documentation’ means that primary goal of the software development is to create software instead of the document. Documentation however has its place; it is a valuable guide for customer to understand how and why a system is build and how to work with the system. (Ambler 2002: 7.)

‘Customer collaboration over contract negotiation’ means that successful developers work closely with their customer, because only customers can tell to developers what they want. (Ambler 2002: 7.)

‘Responding to change over following a plan’ means that developer’s software process must reflect to changes. Every project has to balance planning and changing. There must be possibility to edit project plan

when situation changes, otherwise project plan becomes irrelevant. (Ambler 2002: 7.)

4.1 The Principles for Agile Software Development

The members of Agile Alliance defined their manifesto into a collection of twelve principles that agile software development methodologies should follow to. These principles which can be found from Agile manifesto pages are as follows (Principles behind… 2001):

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

7. Working software is the primary measure of progress.

8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity - the art of maximizing the amount of work not done -is essential.

11. The best architectures, requirements, and designs emerge from self-organizing teams.

12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

4.2 Agile development

Agile development varies a lot compared with old waterfall method. Agile software development stands for time boxed iterative and evolutionary software development and it includes planning with continuous changing. Agile development uses evolutionary delivery, which means that the product is develop gradually, with small pieces at the time. Flexible attitude towards changes is essential part of the agile software development.

Agile methods cannot be exactly defined, only the foundation of the methods is the same. Specific practices vary in every agile method. Every method however shares few same basic practices; short time boxed iterations are used in every method. Also adaptive, evolutionary development of the plans and goals are common to all methods. In addition following same agile principles and practices are emphasized in every agile method: simplicity of the code, lightness of the processes, face to face communication, self-directed teams, programming and workable product over the documentation etc. (Larman 2004: 25-26.)

Agile methods have many common themes. Documentation is subsidiary in comparison with working functionality (Schuh 2005: 18). The customer

appreciates more workable product than comprehensive documentation, if the product is incomplete because of using time for writing the documentation. Of course the documentation is a required part of the product but instead of investing time in writing documents which are not necessary, agile accentuates the working product.

Time boxing is used to ensure that tough decisions are not delayed, and that the most important tasks are prioritized. Time boxing also makes possibility to adopt changes in the middle of the project.

The whole agile team participates in planning and estimating together with the project manager (Schuh 2005: 18). Teams are self-organized, what means that team organizes itself, team members decide how much work they can perform within the iteration, and they decide how they carry out the workload.

Feedback should be aggregated regularly so that team can make adjustments during the project (Schuh 2005: 18). Naturally, the feedback loop is born when after every iteration so called debriefing is held among the teams and team members to get information on how the iteration succeeded.

Projects also must be able to adjust their direction as the result of the internal feedback and the external events (Schuh 2005: 18.)

4.3 Agile methods

Characteristic of most common agile methods are explained in the following chapters.

4.3.1 Scrum

Scrum is the most common and used agile method in the word. It has been created by Jeff Sutherland already in the beginning of the 90’s. Scrum has been used widely and successfully in different kind of projects

and several educational material and researches of the Scrum can be found.

Scrum's differs from the other agile methods, that it emphasizes self-directed cross-functional teams, which gather every day to the stand up meetings. Also strict prescriptive processes are missing. (Larman 2004: 109.)

Scrum's key practices are (Larman 2004: 109):

- Self-directed and self-organizing team. Teams are responsible for delivery of successful outcome at each iteration (sprint). Teams are amongst themselves decided which features can be produce in this iteration. Also the combination of the roles inside the team can be decided the team itself.
- No external addition of work to iteration. In other words work within a sprint is fixed. In next iteration, new requirements and changes can be taken into account, but in current iteration, no changes are allowed.
- Daily stand-up meeting with specified questions. Every team member participates every day in meetings, which take only about 15 minutes.
- 30 days iterations (sprints). Length of the iterations might also vary, but the recommendation is that the spring is always 30 days length.
- Approximately three 30 days sprints per release. This also may change, depending on the size of the product and the project.
- Demo to external stakeholders at the end of the iteration. The product must be workable; otherwise the demo cannot be shown.
- Client-driven adaptive planning in each iteration.

Figure 2 illustrates the Scrum process and the most important practices of the method.

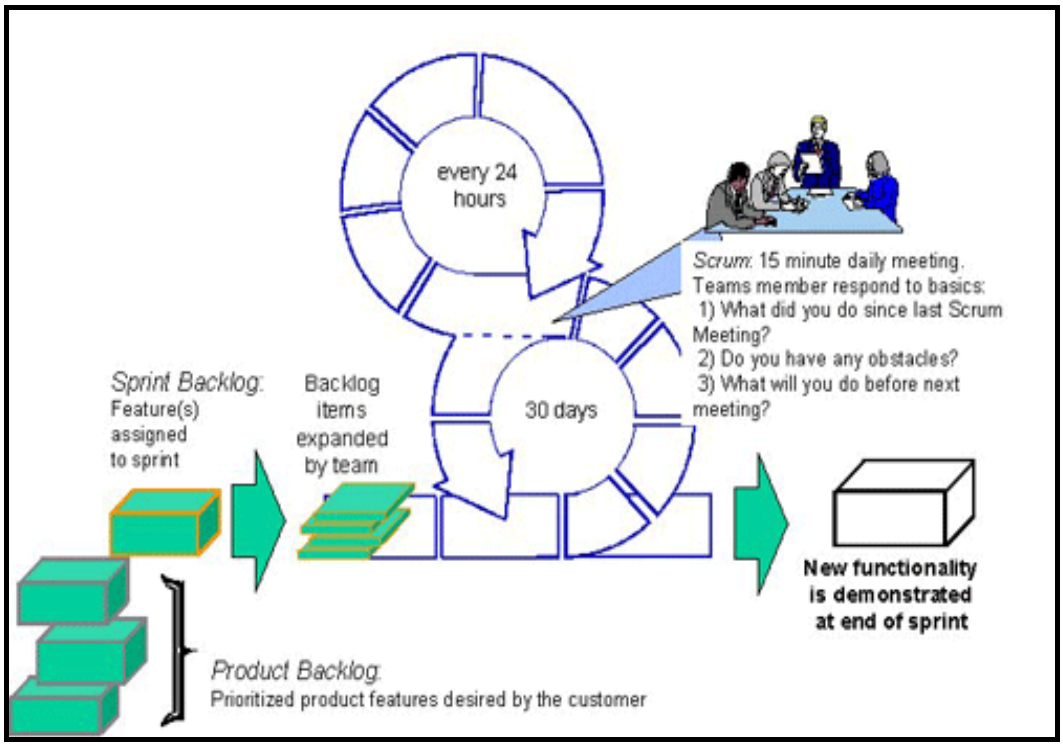

Figure 2: Scrum process (What is Scrum?... 2008)

Scrum is light framework for dynamic, continuous changing environment. Several variables are taken into account in Scrum when developing software and planning the releases. Customer requirements change continuously and new requirements must be taken into consideration when planning iterations, time frame must be thought out carefully in the beginning of the project, resources and backup must be considered etc. (Schwaber n.d.: 3).

There should be less than eight members in Scrum team but multiple teams may form a project and build the increment. Scrum has been used on both, small projects and big ones with hundreds of developers. Scrum practices include working in a common project room, where small teams work together and hold daily stand-up meetings, and representatives from each team meet also daily. (Larman 2004: 111.)

It is however possible to work in separate rooms, but according to the agile main principles, free communication is essential part of the agile method. Thus the open-plan office is the optimal choice for working space. Daily stand-up meetings can be held in corridor, if other common room is not available

The Scrum lifecycle is made up of three main phases: pre-game (includes two sub-phases: planning and architecture), development and post-game/release).

Purpose of the planning phase is to establish the vision. Also expectations are set. In planning phase vision is written, budget and initial product backlog (contains all prioritized requirements that are currently known) are made and items are estimated. Also exploratory and prototypes are made. (Larman 2004: 113.)

In the architecture phase the high level design of the system is planned according to the items in the product backlog list. Preliminary plans for the content of the releases are made also. (Coram and Bohner 2005.)

In Development phase (also called the game phase) implementation of the system is ready for release in a series of 30 day iterations (Sprints). There might be for example three to eight Sprints in one process before the system is ready for distribution. Sprint planning meeting are held in every iteration and also daily Scrum meetings take place. (Larman 2004: 113.)

Each sprint includes following normal software development phases: requirements, analysis, design, evolution and delivery.

Purpose of the post-game phase (release phase) is the closure of the release. No requirements are in the backlog list anymore and the system is ready for the release. (Coram and Bohner 2005.) Training and marketing & sales belong to the release phase also (Larman 2004: 113).

Stakeholders, business users and upper management select the system's features and get a regular view into the activities of the team; the programmers perform the day-to-day management needed to build the system. (Schuh 2005: 23.)

4.3.2 Extreme Programming (XP)

Extreme Programming is well-known agile method. It is widely in use just like the Scrum, but contrary to the Scrum's framework for project management, XP offers practices for the development work.

XP is founded on four values: communication, simplicity, feedback, and courage. Its focuses are on collaboration, quick and early software creation and skillful development practices. (Larman 2004:137.)

XP has twelve core practices are follows (Jeffries 2001):

- Whole team: All the members of one team sit together. Team must include also a customer, who provides the requirements, sets the priorities and steers the project. Other possible team members are programmers and testers. Analysts may help customer to define the requirements. Also there might be a coach, who helps the team keep on track and facilitates the process, and the manager, who provides resources, handles external communication and coordinates activities. The best teams have no specialists, only general contributors with special skills.
- Planning game: XP planning addresses two key questions in software development: foretelling what will be performed by the due date, and determining what to do next. Planning games means a set of rules and moves that may be used to simplify the release planning process.
- Customer Tests: Customer defines one or more automated acceptance test to show that the feature is working. The team

builds the tests and uses them to prove to themselves and to the customers, that the feature is implemented correctly.

- Small Releases: Team releases running and tested software on every iteration. Software is visible, and given to the customer, at the end of every iteration. Everything is open and concrete.
- Simple Design: Teams build software based on a simple design. Programmers design and code a system that works here and now, not something, that may be needed in the future.
- Pair Programming: Two programmers are sitting side by side at the same machine and build together software. This way all code is reviewed beforehand and quality of the design, testing and code is better.
- Test-Driven Development: Every time any programmer releases any code to the repository (twice a day or more) every programmer tests are run correctly. This way programmer gets immediate feedback.
- Design Improvement: XP uses a process of continuous design improvement called ‘Refactoring’. In refactoring process, all the duplications in the code are removed and ‘cohesion’ of the code is increased. Refactoring is supported by comprehensive testing to be sure that nothing is broken.
- Continuous Integration: In XP, teams keep the system fully integrated throughout the development. This way the system is never far from a production state.
- Collective Code Ownership: On a XP project, any programmer pair can improve any code at any time. It increases quality of the code and reduces faults.

- Coding Standard: Teams follow a common coding standard. All the code in the system looks coherent and harmonious.
- Metaphor: ‘Metaphor’ is teams’ developed common vision (simple description) of how the program works.
- Sustainable Pace: Sustainable pace means that the team members work hard at a pace that they can go along with for the time being.

One essential idea of the XP is that different practices should be tailored to suit the needs of individual projects. Project does not need to adopt all XP practices but select suitable and needed practices.

An XP project is run in one- to four –week iterations. XP is a very handy tool for rapid prototyping and systems intended for rapidly changing business environments. At the end of the each iteration, fully programmed, tested and production worthy version of the system is delivered. Usually a specific number of iterations are grouped into a release and new software is delivered into production less frequently. (Schuh 2005: 22.)

XP is suited for small and medium sized teams, only three to twenty project members in the team. Also the physical distance between the team members can not be big because the communication and coordination between project members should be possible all the times.

Following five phases constitute the life cycle of XP (see figure 3): Exploration, Planning, Iteration to Release, Productionizing, Maintenance, and Death. (Larman 2004: 142.)

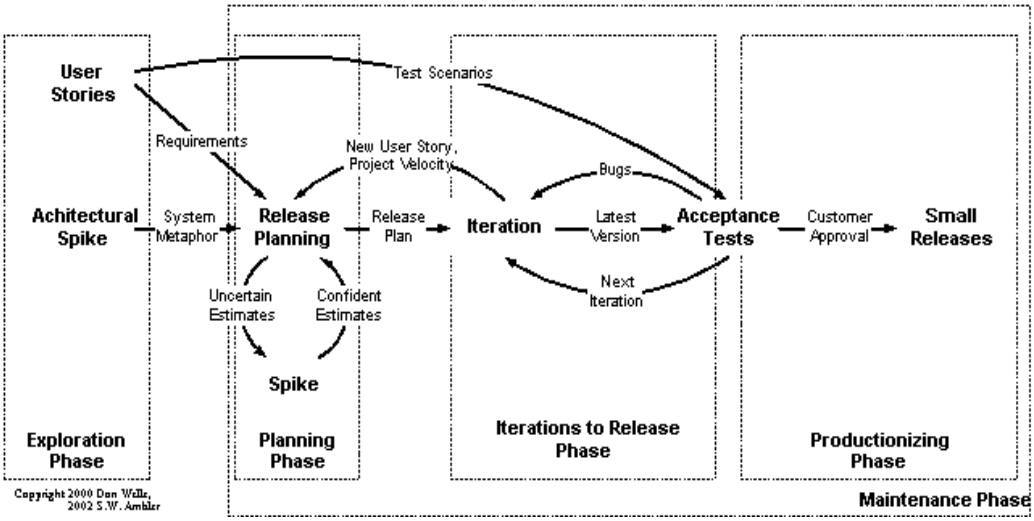

Figure 3: XP life cycle (Ambler 2007b)

Customer writes the story cards (features) in the Exploration phase, in which they specify what is wanted to be included in the first release. Also in the Exploration phase, project team familiarize themselves for example with the technology and the practices they will use in the project. Also the prototype of the system can be built. (Larman 2004: 142.)

In the Planning phase the stories are set to the priority order. Also an agreement of the content and the schedule of the first release are made. (Larman 2004: 142.)

The Iterations to release phase includes several iteration before the first release. Each iteration takes one to four weeks to implement. In the first iteration a system with the architecture of the whole system is created. Customer's created functional tests are run at the end of the every iteration. After the last iteration the system is ready for the production. (Coram and Bohner 2005.)

In the Productionizing phase more tests and performance checks are made before the system can be released. New changes may be found and they might still be included in the current release. The postponed

ideas and suggestions are documented for the later implementation. (Coram and Bohner 2005.)

The XP project must keep the system in the production while also producing new iterations. In the Maintenance phase support for customers is given. (Coram and Bohner 2005.)

The Death phase takes place when the customer does not have any more stories to be implemented. The necessary documentation of the system is written and no more changes to the architecture, design or code are made. The death phase can be executed also if the system is not delivering the demanded outcomes, or if it becomes too expensive to develop anymore. (Coram and Bohner 2005.)

4.3.3 Crystal

Alistair Cockburn developed Crystal methods. In Crystal method ‘peopleware’ issues (such as communication and education) are prioritized over process.

Crystal is a collect of software development methodologies, which are people focused, communication-centric, ultra light and highly tolerant. Following three properties are central to every Crystal methodology: frequent delivery, close communication and reflective improvement. (Schuh 2005: 30.)

Crystal methodologies are cataloged by project size and criticality. Project size (people on the team) is marked by color (the darker the color the heavier the methodology) and criticality (measures the severity of damages) by letter (C: Comfort, D: Discretionary money, E: Essential money and L: Life). According to the Crystal, as the team gets larger, a heavier methodology is required. (Schuh 2005: 31.)

Projects use incremental development cycles, which lengths are not more than four months, recommendation is between one and three months. Crystal methodologies do not define which development

practices, tools or work products have to be used. For example practices of XP or Scrum can be adopted. (Schuh 2005:33.)

Following three main Crystal methodologies have been constructed: Crystal Clear, Crystal Orange and Crystal Orange Web. Only Crystal Clear and Crystal Orange have been constructed and used in practice. (Schuh 2005: 32.)

Crystal Clear is meant for very small project up to eight people working on one team and same area. Crystal Orange is designed for medium sized projects, with 10 to 40 members in the project. Duration of the project is max two years. (Schuh 2005: 32-33.)

Both Crystal Clear and Crystal Orange use following policy standards (Abrahamsson, Salo, Ronkainen, Warsta 2002: 39[1]):

- *Incremental delivery on a regular basis*
- *Progress tracking by milestones based on software deliveries and important decisions rather than written documents*
- *Direct user involvement*
- *Automated regression testing of functionality*
- *Two user viewing per release*
- *Workshops for product- and methodology-tuning at the beginning and in the middle of each increment.*

According to the Crystal Clear, incremental delivery happens within a two to three month time frames. In Crystal Orange, the duration of the increments can be max four months. (Schuh 2005: 32-34.)

[1] Original source of information: Cockburn, Alistair 2002. Agile Software Development. Boston: Addison-Wesley.

4.3.4 Feature driven development (FDD)

Feature Driven Development (FDD) is an agile and adaptive approach for developing systems. FDD focus is on the design and building phases, not to the entire software development process. It does not require any specific process model to be used and it has been designed to work with the other activities of a software development project. (Abrahamsson, Salo, Ronkainen, Warsta 2002: 47.[2])

FDD is based on eight main practices, from which the agile team can adopt one or more. However the best profit can be obtained only when all eight practices are taken in use. Those practices are (Schuh 2005:26):

- Domain object modeling: overall roadmap of the system to be built. It is composed of high-level diagrams that describe the relationships between classes and sequence diagrams that demonstrate behavior.
- Develop by feature: Common foundation to all agile methodologies.
- Class ownership: Each class within a system is assigned to a specific programmer. Opposite of XP's collective ownership.
- Feature teams: Since features usually involve more than one class, feature teams are the common approach to design and development in FDD.
- Inspections: Focus on the identification of defects. Improve the transfer of the knowledge and conformity of coding and design standards.

[2] Original source of information: Palmer, S.R. and Felsing, J.M 2002. A Practical Guide to Feature-Driven Development. Upper Saddle River, NJ, Prentice-Hall.

- Regular build schedule: Complete system is built at regular intervals.
- Configuration management: The code, analysis, design and testing artifacts need to be stored and versioned throughout the lifetime of the project.
- Reporting/visibility of results: Regular and easy-to-understand updates of status.

FDD consists of five sequential processes (see figure 4). During these processes the system is designed and built completely. Typically an iteration (that includes both designing and building) of a feature takes one to three week period of work for the team.

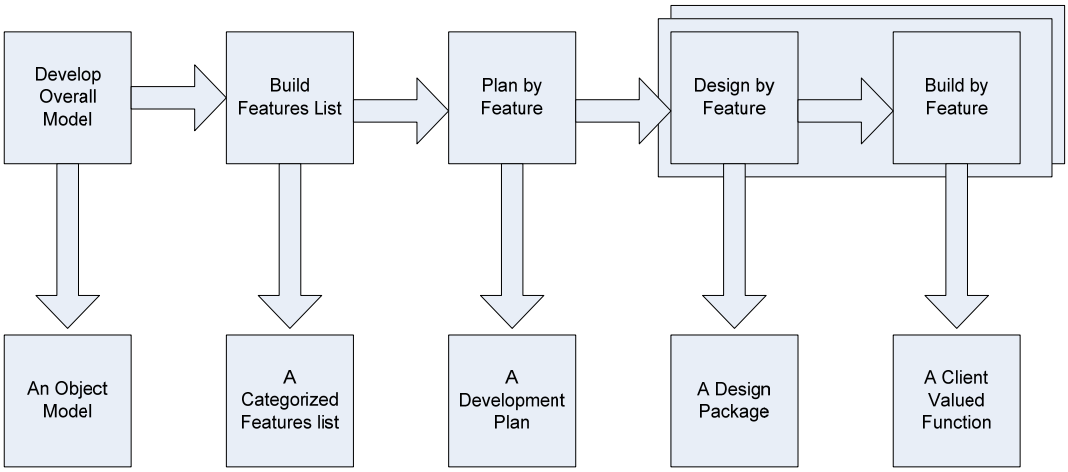

Figure 4: FDD Workflow (Nelson n.d.(b))

Five processes of the FDD method are Develop an Overall Model, Build a Features List, Plan by Feature, Design by Feature and Build by Feature (Nelson n.d.(b)).

- Develop an Overall Model: The project beginnings with a so called high level walkthrough, where the chief architect and team members are informed of the high level description of the system. The overall domain is then divided into different domain areas and more detailed walkthrough is held for each of them with the

domain members. After that a development team works in small groups and produces object models for the specific domain area. Then development team decides the suitable object models for each of the domain areas. Finally domain area models are merged into an overall model.

- Build a Features List: In the Feature list, the development team states each oh the client valued functions included in the system. Features should no take more than two weeks to complete, otherwise they should be divided into smaller pieces. Users and sponsors of the system review the feature list for insuring the validity and the completeness.

- Plan by Feature: Subsequently it is time to produce the development plan. High level plan is created, in which the feature sets are ordered according to the priority and dependencies and also assigned to Chief Programmers.

- Design by Feature and Build by Feature: A group of features is selected from the feature sets and feature teams needed for developing the selected features are formed. Selected features are produced by iterations; one iteration should take two weeks in maximum. Iterative process includes following tasks: design inspection, coding, unit testing, integration, and code inspection. After that the completed feature is promoted to the main build.

4.3.5 Adaptive software development (ASD)

Adaptive Software Development (ADS) emphasis is on the problems in developing complex, large systems. In ADS, the static Plan-Design-Build lifecycle is replaced with a dynamic Speculate-Collaborate-Learn lifecycle. In other words ADS project is carried out in cycles; every cycle consists of three phases as shown in figure 5: speculate, collaborate and learn. (Schuh 2005: 36.)

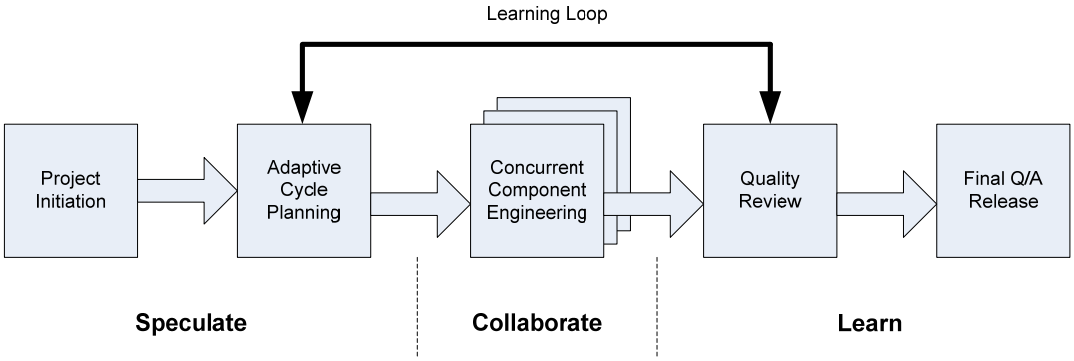

Figure 5: The ASD lifecycle phases (Highsmith 2000: 26)

Speculate phase includes two phases (Project initiation and Adaptive cycle planning) and seven different steps Highsmith 2000: 26):

- Conduct the project initiation phase: setting the mission and objectives of the project, understanding and documenting constraints, establishing and outlining requirements, making initial size and scope estimates, and identifying key project risks.
- Determine the project time box: setting the time box for entire project.
- Determine the optimal number of cycles and the time box for each: for a small or medium sized application, cycles usually takes from four to eight weeks.
- Write and objective statement for each cycle: developing a theme or objective for each of the cycles.
- Assign the primary components to the cycles: see next step.
- Assign the technology and the support components to cycles: Every cycle must deliver a visible, tangible result to an end user.
- Develop a project task list: Each component can be a target of a task. Also additional tasks, which are not directly component

related but necessary for project complication, can be added to the task list.

Purpose of the Collaborate phase is to deliver working components. Several components may be under coincident development. Actual programming activity occurs in Collaborate phase. Contrary to the Scrum, ADS does not define how the programming should be done or how programmers should go about performing technical activities. (Schuh 2005:36.)

In ADS, focus is on collaboration across the project team instead of focusing on design, build and testing. ADS does not recommend any specific procedure for fostering collaboration within a project. Collaborate practices can be adopted for example from Extreme Programming (XP). For example pair programming and collective code ownership are suitable practices for small closely-spaced teams. (Highsmith 2000:27.)

Giving feedback is the main purpose of the phase Learn. Each iteration ends with a quality review. In review, following issues are gone through: result quality from the customers' and technical point of view, the functioning of the delivery team and the practices they are utilizing, and the status of the project. (Highsmit 2000:27.)

Focus of the ADS is on the results and the quality of the results instead of tasks or the process used for producing the result. Characteristics of the adaptive development cycles are as follows (Abrahamsson, Salo, Ronkainen, Warsta 2002: 71[3]):

- Mission-Driven: All activities in each development cycle must be in accordance with the overall project mission. The mission must be checked, as the development proceeds.

[3] Original source of information: Highsmith, Jim 2002. Agile software development ecosystems. Boston: MA., Pearson Education.

- Component-Based: Development activities are not task-oriented. Focus is on developing working software by building the system a small piece at a time.
- Iterative: Focus of the development is on redoing. Components develop over several iterative cycles according as customers' feedback.
- Time-Boxed: Regular deadlines forces a project team continuously re-evaluate the validity of the mission and make hard trade-offs early in the project.
- Change-Tolerant: Developers must constantly evaluate whether the components they are developed are probably to change.
- Risk-Driven: The development of the high-risk items should be begun as early as possible.

ADS has only few practices to daily development work: iterative development, feature-based (component-based) planning and customer focus group reviews.

4.3.6 Dynamic systems development method (DSDM)

Dynamic Systems Development Method is most famous framework for rapid application development (RAD) in the UK. The basic idea of the DSDM is to fix time and resources, and then adjust the amount of functionality accordingly. (Nelson n.d.(a).)

Focus of the DSDM is on building systems in quick and small increments. Compromises between more valuable and less valuable features are needed to be made. DSDN has nine main principles (Schuh 2005: 38):

- Necessary active user involvement.
- Authority for the team to make decisions.

- Frequent delivery of products.
- Propriety for business purpose is criterion for acceptance of deliverables.
- Iterative and incremental development.
- Changes during development are reversible.
- Requirements are baseline at a high level.
- Continuous testing integration.
- Collaboration and cooperation between stakeholders.

Size of the development team should be composed of two to six members. Several teams can work together within one project.

DSDM is composed of five phases (see figure 6): feasibility study, business study, functional model iterations, design and build iteration, and implementation. Feasibility and business studies are done only once one after the other. Three last phases, according to the agile principles, are iterative and incremental. Time boxes of the DSDM takes form two to six weeks. Each of these time boxes can contain multiple cycles of each of the iteration phases. (Nelson n.d.(a).)

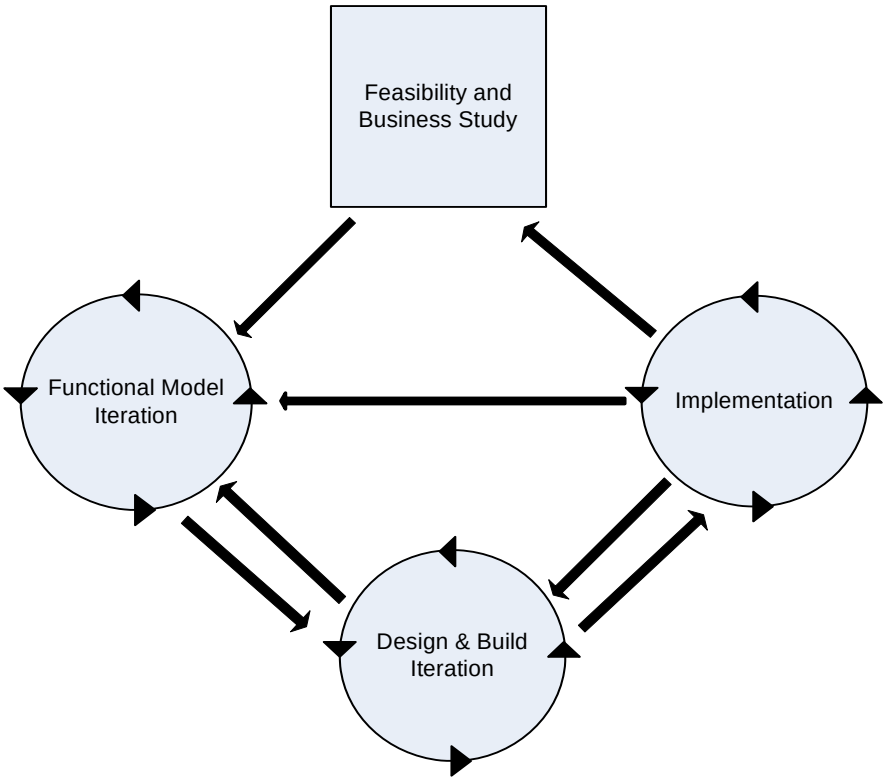

Figure 6: DSDM project flow (Nelson n.d.(a))

In Feasibility Study phase, it is decided if DSDM is suitable method for the project by answering given questions provided by the DSDM consortium. Also the technical feasibility of the project is assessed. Feasibility report and an outline plan for the development are results of this phase. Basic process flows of the business are analyzed in Business Study phase. (Nelson n.d.(a).)

In Functional Model Iteration phase functional models of the components are produced. New prototypes of the models are produced iteratively until quality of the product is acceptable and it can be implemented.

In the next phase, Design and Build iteration, the prototypes are fleshed out and tested. Users review the prototypes and give the feedback. (Nelson n.d.(a).)

In Implementation phase, the prototypes are transferred into production. New features are incorporated into the work environment.

4.3.7 The Rational Unified Process (RUP)

RUP is a special model of the more generic Unified Process. RUP is not actually one of the pure agile methods, but because RUP is based on an iterative and incremental foundation that is common to agile, it can be counted on to the agile methods.

RUP is system development process, but in addition, it is a system development process framework. It means that RUP is a structure, from which a process can be created. An organization does not need to adopt the whole RUP process, but the process can be tailored to meet the needs of the organization. The specifics of the process vary, but the main concepts remain the same (Ambler 2005: 17.)

RUP has six main practices: developing software iteratively, managing requirements, using component-based architectures, visualing model software, verifying software quality, controlling changes to software. (Abrahamsson, Salo, Ronkainen, Warsta 2002: 59.[4])

- Develop software iteratively: Iterative development is the basic practice of the RUP. Software is developed in small increments and short iterations.
- Manage requirements: Identifying the requirements of the system that possibly change over the time. Requirements are prioritized, filtered and traced.
- Use component-based architectures: Those components which are most likely to change can be isolated and to be more easily managed. The components can be also re-used.

[4] Original source of the information: Kruchte, P 2000: The Rational Unified Process: an Introduction. Addison-Wesley.

- Visually model software: By using a common visualization method (for example Unified Modeling Language, UML), system architecture and design can be demonstrated clearly to all parties.
- Verify software quality: Verification is done on every iteration and thus faults and defects can be noticed earlier in the development cycle.
- Control changes to software: Any changes to the requirements must be managed, and the effects of the changes made to the software must be traceable.

RUP is composed of four phases (see figure 7): Inspection, Elaboration, Construction and Transition (Larman 2004: 180).

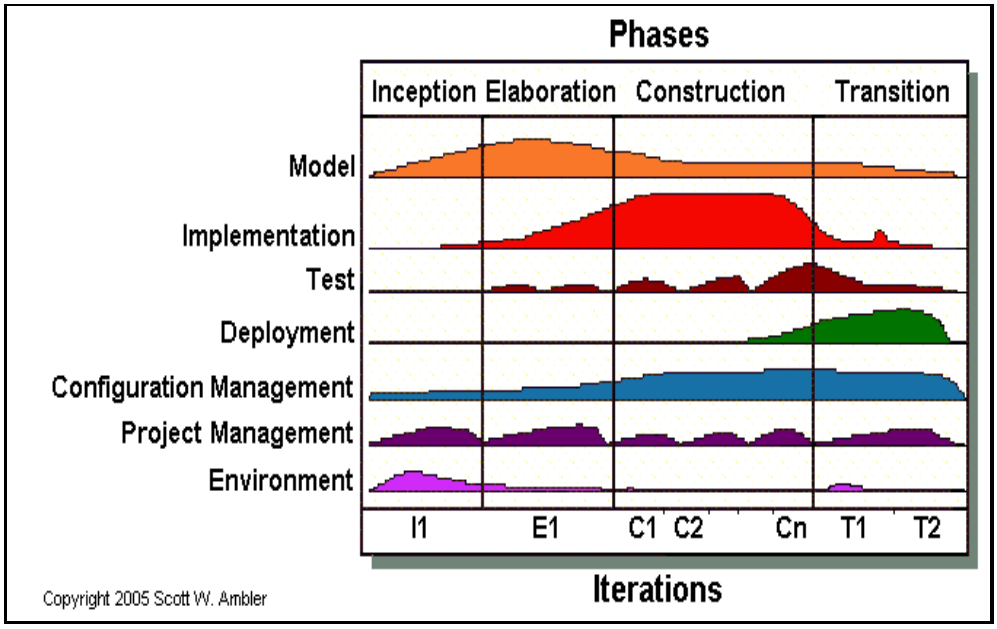

Figure 7: RUP lifecycle (Ambler 2007a)

Inspection phase is short-term, takes ideally only few days. Iterations are not usually needed. In this phase, life-cycle objectives are stated, critical use cases are identified, candidate system architectures are composed, and the schedule and cost estimations are laid for the entire project. Also

estimations are made for the following elaboration phase. (Nelson n.d.(c).)

In Elaboration phase, plans how the system should be built and how it will work are defined. Detailed models and descriptions are made, such as use case diagrams, use case descriptions, sequence diagrams and class diagrams. Also working prototype can be made. (Nelson n.d.(c).)

The actual product is created and the code of the product is written in the Construction phase. The product and the code is also tested. (Nelson n.d.(c).)

In Transition phase, the ownership of the product is assigned to the customer. Verification can be continued also in this phase, and training of the product can be provided to the customer. (Nelson n.d.(c).)

4.4 Comparison between different agile methods

It is extremely important to choose correct, suitable agile method or a combination of different agile methods for the team. Because all the agile methods are not applicable to any type of a project, it is necessary to carefully consider what agile method would fit to the project. Perhaps it would be useful to combine different agile practices together from different methods.

Projects have variation a lot for example in their length, complexity, susceptibility to risk, recourses, resource competences, stability of the requirements etc.

If all these details needed to be taken into account in the agile process, it would lead to selecting and tailoring the agile method from the beginning for each project over and over again. In practice that is not possible neither advantageous, since taking new method into use will always require some kind of start-up period. During the start-up phase there

usually is some resistance against new methodologies and the motivation level of the team may suffer due to confusing atmosphere. Once the method is selected the basic practices should be kept the same and the process should be developed step by step without disruptive changes between the projects.

The following sections introduce some most obvious advantages and disadvantages found in different agile methods that should be taken into account when selecting the method and practices for the team.

Advantages and disadvantages of Scrum

Scrum is light process framework that can be complemented with practices and processes from other more detailed defined methods like XP. Self organized teams are the core of the method and the team should take much responsibility in achieving the targets. Scrum itself does not define any programming practices for the implementation phase but it defines what is expected from the team, and some basic practices that the team should follow in order to achieve its goals (for example 15 minute daily meetings with accurately defined questions). This promoting of the team work and self organization can be seen as major advantage since it should raise the motivation level.

In Scrum any changes are denied after the sprint requirements have been defined in the beginning of each sprint. For the team this is good since it provides some peaceful time to perform all the tasks needed to complete the sprint requirement list. On the other hand, for example sprint of one month may cause unacceptable delay, if some major customer is asking a quick solution for some small specific problem. Actual worst case delay may be then nearly two months before the solution is seen in release.

Scrum breaks the problem in hand to small parts that are manageable by the team. The daily builds, constant integration, and testing will guarantee that the bugs are found.

Advantages and disadvantages of XP

Major advantage of XP is that it is widely used and there exists lot of different information sources. XP defines process with frequent builds and iterations and goes into more details in actual management and programming practices than Scrum. Disadvantages are that the method is applicable for small teams (5-10 programmers only) and that the on-site customer requirement cannot be fulfilled in our case. The organization of the team has already spread over several locations and therefore the one-site requirement presented in XP is not applicable either.

XP is somewhat free-formed allowing the developers to address new issues or requirements on the fly, which could be useful in some situations requiring rapid reactions to customer needs. The amount of formal meetings is minimized, the daily stand up meetings are seen as effective way of sharing the information.

The code quality is under constant peer review because of the pair programming. On the other hand some individuals may resist pair programming at least during the start-up period.

XP enhances the production efficiency by reducing the amount of documentation, which may be seen either good or bad thing. The product may be ready sooner, but in worst case without updated documentation to be provided for the customer.

XP has some useful practices which could be taken into use in parts of the team but it does not provide a complete solution for our team.

Advantages and disadvantages of Crystal

Crystal defines different kind of processes for small and large projects so it should provide suitable methodology. It does not define any strict tools or practices and those could be taken from other methods. Crystal Clear might be not applicable since it is defined for only six developers in maximum and requires a shared office space. However, Crystal Orange is targeted for medium-sized projects and could be more easily adjusted for our purposes. It promotes for effective communication and that team members should be located on one site, which is not applicable for us. Testing is seen as integral part of the development and each smaller team working in the project should have a test engineer.

One major disadvantage is that the method has not been used in our company and hence there is no in-house experience in applying the method in practice.

Advantages and disadvantages of FDD

FDD addresses model centric design which could have some impacts in starting new, and especially continuing old projects with no existing models.

The weakest practice of FDD is the individual code ownership that will raise the risk level in schedule wise for example if some key specialist gets sick in critical development phase. The information sharing is much better for instance in XP, because due to the pair programming practice, at least two people are sharing all the details on the part they are working on. The iterations are not so tightly defined as in other agile methods and in that sense the process definition would require some more work to be done in start-up phase.

FDD defines practices also for big teams and how multiple teams work in parallel so it is scalable for different kind of projects. However the

iteration content is not as well defined as in other agile methods. FDD uses inspection to remove defects and improve the quality. Testing is mandatory part of the process.

Advantages and disadvantages of ASD

In ASD collaboration, iterative development component by component and customer feedback are addressed. ASD describes general guidelines for development process but leaves very much to be planned for everyday practices.

The advantage is that the practices could be tailored to fit for the team requirements. On the other hand, disadvantage is that developing the practices fitting in to the ASD process, perhaps requires more work and time than the methods describing more strict practices.

Advantages and disadvantages of DSDM

DSDM is the most formal of all the agile methods and requires more documentation as well. The process contains much architectural design in the beginning of the projects. Testing is addressed heavily and each project team is required to have at least one test engineer, which is good.

Business value is expected to have highest priority and a specific approach is presented to define how important a certain requirement is for on-going iteration.

Major disadvantages are that the process seems to be very heavy and that the access to material describing the practices in detail is charged and controlled by Consortium.

Advantages and disadvantages of RUP

Common visual modeling method and documentation are addressed for insuring effective communication. Testing and verification are considered as integral part of iteration round which is excellent.

Commercial tools for RUP exist, which could speed up the start-up phase but also would require extra investments.

RUP is again method that more describes and gives a framework for developing and tailoring the process for the project team. Again this might lead to a long start-up time in taking the methodology in to use.

5 USER EXPERIENCE

An interview study was made about the practice of using agile method in a team. Five persons were interviewed and all of them work in the same company but in different teams and with different products. They all also represent different responsibilities (Verification engineer, System architect, Quality manager, R&D Manager and Software design engineer).

The purpose of the interviews was to find out what agile method is used mostly and why and how it was introduced. I also wanted to know, which were the most common problems and difficulties when agile method was taken in use and the old method was superseded. Questions of the interview can be found from appendix A.

Also the purpose of the interview was to clarify the common practices the agile method brought along.

5.1 Background

Almost all interviewees have been using some agile method nearly two years. Only one interviewee was now using some other method than agile, but the rest was still using agile with their daily work.

Scrum in its pure or somehow moderated form was commonly in use. I did not pick up any team/person who had used something else than Scrum in our company. There are agile wiki pages in our intranet, and also XP Programming, Crystal and other methods are mentioned there, but in our circle of acquaintances Scrum was the most commonly in use.

The role of the interviewees in the agile teams varies much. One of them is a software design engineer and a member of the agile team, one was some time ago a verification engineer and a member of the agile team, but worked occasionally also as a Scrum master. One worked first as a Scrum master, but now she is a quality manager and acts as a

responsible manager of the agile process. One interviewee is team leader but acts also as a Scrum master. One is the system architect and customer trainer and act as a team member of the agile team.

5.2 Introduction of the agile method

Part of the teams, which members was interviewed, took agile in use gradually. But rest of the teams adopted agile method at once, without any pilot teams or pilot projects.

Interestingly it was no self-defeating to introduce agile method totally at once. Before interviews, I thought that it can not ever success to change one software development method to another at once, because it is totally new way of work to the most of the people. And of course there is always the natural resistance which delays the adoption of the new method. During the interviews it came evident, that introducing a new method can be made at once, but it takes a lot of work and hard commitment to get a new agile method to work.

Maybe one reason that agile method can be taken in use at once is, that the age structure in our company is quite homogeneous, the average age of the employees is quite low and most of the employees has been worked in the company relatively short time.

However it took quite long time to get used to the new method. Almost everybody said that it took at least a year to achieve the most workable way to utilize the agile method. Perhaps it would have been easier to follow strictly some agile method instead of adopting few features from one agile method and few from other.

The resistance of the change is maybe bigger when adopting the new method is made at once. But that way the effect of the resistance stays at the acceptable level because of its brevity.

It took only a few weeks to employees in agile teams to understand new working practices. However one interviewee said that education should have been planned more carefully than actually made. So, although it is possible to introduce a new method at once, it is important to introduce the new method somehow (for example teach the new method by way of pilot project) and perhaps also arrange an actual training.

5.3 Choosing the agile method

All teams, which representatives were interviewed, concluded to choose Scrum as their agile method.

One big reason for that decision was that it was possible to participate to the lectures of Graig Larman, the expert of the Scrum method. As a matter of fact, Larman recommended to one interviewee that they should adopt the Scrum in its pure form. Larman also recommend that after the team has familiarized itself with pure Scrum, they can tailor method to format that fit for the team in question.

5.4 Building the agile team

All teams of the interviewees were consisted of five to ten employees. According to the agile method, those teams were cross-functional, in other words teams were composed of persons with different roles. For example there were following roles represented in one interviewee's team: architect, several software developers and verification engineers, usability expert, domain expert and of course the Scrum master.

All teams are responsible for one software function, the whole team might be composed of even 70 persons, and team has been divided to sub teams of ten persons or less. In one team, there was in one point 15 persons, but it was considered to be too big and hard to manage: meetings were drawn out and because of the excessive workload; performance of the scrum master's own tasks out side of scrum work became weaker.

5.5 Agile practices and daily work

Every team has their own way of use of agile practices. Although every team used Scrum as their agile method, practices varied somehow. However every teams' working methods and –habits had same aspects and elements, they were modified to be suitable to the team.

5.5.1 Meetings and reviews

Daily Scrum meetings are arranged, depending on the team, once a day or few times a week. One team held meetings initially every day, but after a while they noticed that best way to this team is to get together only few times a week. Even though it is against the rules of the Scrum.

Scrum Master did not participate in all daily meetings, but teams gathered without the master to go through all specified questions.

Daily meetings take usually about 15 minutes and every participant answers to all three questions:

- What did you do since last Scrum
- What got in your way of doing work
- What will you do before the next Scrum

In addition to these questions one team had their own extra questions, such as are there needs for help or peer review from other team member, or needs for internal demos.

If some problems came up in the daily meetings, new meetings were arranged for discuss about those problems. Only those workers, who are related to particular problem or who can help to solve it, are invited to that meeting. That way all the other can carry on their own tasks and they do not have to use working time for nothing.

All interviewees had similar views on the length of the daily meetings; it is important that daily meetings do not take too much time. Purpose of the daily meetings is not to make specifications how things should work but go through what has been done and what will be done.

In addition to daily meetings, all teams also arranged other agile meetings. Usually every month two Sprint Plannings are arranged, where the teams plan following iterations. Also Requirements workshops are held, where features, workflow and other requirements are went through, so that the whole team gets the common understanding for the goal.

Sprint review is held after every iteration. In review all the achievements and workings are talked through. After the Sprint review, retrospective is held for developing the Sprint process.

5.5.2 Iterations and requisited results

Usually iterations take one month, according to the recommendation of the Scrum. One team arrived at a conclusion that normally iteration takes four weeks, but in holiday period, because of the small number of the employees, length of the sprint was extended to eight weeks.

In one team, project is carried on in two weeks iteration. Results have been excellent and the team members are fond of that shorter cycle.

Required results of the iterations varied between the teams. Generally speaking, workable demo was required outcome of the iteration. With demo, team can show how product work and what has been done since the last iteration.

In the demos, only workable functionality is shown. It is not permitted to represent unfinished or unworkable functionality; the status of the function must be ’Done’. Minimum requirements are that items, which status is ‘Done’, are coded, unit tested, reviewed, integrated and more or less functionally tested. One requirement for the status ‘Done’ is that

functionality of the product is also described shortly in the wiki-pages of the team.

In other team, requirements of the results of one iteration are that the specifications are done and published. Also demo is represented, which covers one or several requirements. Testing team has run the test cases and the customer documentation has been published, testing team does not directly belong to the agile teams in that team, contrary to the agile rules.

5.5.3 Actions if schedule fails

In all agile methods, schedules are kept, but content must be reduced if it looks like that desired outcome cannot be attained within the planned schedule. All teams also held to that rule. Schedules were kept strictly; content of the backlog was changed when needed. But also the strict rule was that unfinished product is not permission to demo. If product is unfinished at the end of the sprint, this item is transferred to next or later sprint.

It came to light, that when it is busy, overtime work is done because half-done products cannot be shown in the demo meetings. And also in the daily meetings some developers do not have temerity to confess that there has not been any progress. In one team, overtime ban has been set and content has been reduced, if schedules fail.

5.5.4 Documentation

Usually agile is considered as being a good excuse to fail the documentation. This was not the case in the teams of the interviewees. Any of the team documentation was not fallen off; in some team documentation became even better in particular cases. In one team documentation was quite poor before moving to the agile. And it still is, after using agile few years. So agile it self does not make documentation better, because emphasis of the work is on producing workable product

instead of using time for writing perhaps even useless documents. In one team they dreamed that along with agile, specifications and other documents would not have to be written anymore. This illusion was noticed to be wrong in no time.

Overall documentation is always needed to some extent. For example in Extreme Programming, pair programming ensures, that strict documentation is not needed between developers because main emphasis is on communicate via voice and working in tandem with a colleague. But still other stakeholders need written information and documentation is needed to write to them. Also testing documentation (for example test cases) must be up to date, even if main stress is on actual test work.

5.5.5 Verification of the software

Every interviewee impressed on, that the amount of the verification has been grown substantially. Usually investment in verification documentation is higher and along with the documentation, also testing it self has got better.

Because agile teams are cross functional, also verification engineer belongs to the team. That is considered to be a positive thing, because that way a tester gets in on development from the start and the bulk of the bugs can be found in the early stage of the software development. Anyway in one team faults have been found unexpectedly late in the project and that is why faults have been corrected right at the end of the release. Herein I started to wonder if it is a real agile method in use in this team. In agile this kind of situation can not happen because of the continuous testing. If faults are found in very late in the project, the agile method is not used correctly; some kind of minor waterfall cycles might be in use. In that case the team should go through its development method once again and think how to make it better.

5.5.6 Continuous integration, daily builds and smoke tests

According to the interviews, every team uses continuous integration. But term ‘continuous integration’ was used against its right meaning.

Continuous integration does not mean that integration is done once a day and a build every night. Continuous integration means non-stop integration. When some changes are done to the code, automated integration is done and unit test are run for ensuring workable product. If building or unit tests fail, first thing is to fix those bugs. Further development of the product takes place after fixing the faults. That way continuous integration helps, that code is unbroken and valid all the time.

Also daily builds and smoke tests were used in the teams. Smoke tests are run after every build; they inform that the build succeeded and program is in publication condition.

5.5.7 Quality of the software

Quality of the software varied between the teams largely. In one team, attention was paid to the verification in early stage of adopting the agile method so much, that the product was very high class and well-designed already in the first pre-pilot. But in the later project, testing was invested less and it was directly seen in the quality of the product.

In other team quality of the product was not such as the customer desired. The customer was not satisfied with the quality and the customer did not always even know what kind of product they received. This is maybe due to the lack of the customer participation to the approval of the requirements, design and final out come. One agile rule is that customer must take part in design and development work all the time. Also too strict schedule was mentioned to be a one reason for a bad quality. Accomplishment of the product has been quickened but at the same time number of the faults has been increased and the quality has been gone down.

5.6 Problems and achievements (lessons learned)

One big problem when transferring to the agile was developers' resistance of the changes. Resistance existed no matter which way to transfer to the agile; in one go or gradually with a few practice at the time. Resistance of the changes fell naturally off in process of time when new method was learned and when it was noticed, that there were also good aspects.

Every interviewee found good aspects from agile, nobody thought, that it is better to return back to the old way of work. Pair programming was found to be a good practice, though it was not used officially in any of the teams. Especially continuous code review of the pair programming felt as a positive thing, which brings down the number of the faults. One team used pair programming mostly when doing difficult tasks, for example when developing new functionality to the product.

Also common code ownership was brought up as a thing which improves the competence transfer to all team members. This assures that for example in case of acute illness, developing of the program can continue, because basically anyone can carry on the work of other team member.

One positive thing what agile brought along was that every member of the project knows in more detailed level what is the situation of the developing process and project. Daily meetings versus normal weekly meetings helps everybody to be more familiar with meaning of the own work and with the general view of the project.

Agile project demands that a customer commits to participate to the project for the whole project life cycle. Customer must be a one member of the team. Otherwise it might happen, that program is developed differently from the desires of the customer. Hence the kind of changes are accepted to the program, which customer does not prioritize and

important features might be left off against customer’s will. It is important that the customer get the product they need, and the customer should also know what kind of product they are going to get.

6 AGILE METHOD RECOMMENDATION

Choosing the right agile method is not the most important thing when moving to agile. Graig Larman said in his lecture (Tampere 28.2.2008), that team does not DO agile, but BE agile. That means that agile is not only following specific agile practices. Basic idea of the agile is that the focus is on adopting changes, not on following the plans. Practices itself, just as time boxed iterations and daily standups, comes along with mobilization of the agile method. They are only superficial practices, they are not agile itself and using those practices does not make team as an agile team.

There is no one and only suitable agile method for the team. Any of the agile methods is not ready for taking in use as such, but they must be adjusted to the project and the surrounding environment. In circle of acquaintances Scrum is in use with different modifications and features. But interviews and literature brought up that Scrum does not alone solve all problems team has at the moment: lack of quality, problems in change management, delayed finding of faults etc. Scrum gives framework how project management and teams work, but it does not define any engineer practices.

Larman said in his lecture that Scrum cannot be customized, that Scrum is needed to follow totally or it is not Scrum at all. However, it is said in many other source of information (for example Shuch 2005:47) and also interviewees said that agile methods can be and must be modified and adopted in case of need.

One of the basic ideas of the XP is that it is not applicable, or it is not even worthwhile, to adopt all XP practices for all the individual projects. Instead of using all XP practices, they should be tailored to suit the needs of the projects. In this study, I arrive at a conclusion that we can take the best suitable XP practices in use in our team, but in addition, also some other agile method practices can be adopted.

FDD and other less known methods could suit to the team if they are modified and tailored to fit to the team. Problem is that there is no experience and counseling available nearby. In addition there is no literature easily available and even researches of method usage in the real world are not necessarily made. Also educational material and education can be hard to find. That is the reason why it is dubious and risky to adopt that kind of method, from which is unknown how it fit to the in practice.

From Scrum and XP there is arranged high class education, even in Finland. Also inside of the company education is available for Scrum and XP, literature is available, and the most important thing is that know-how can be found from the company.

There is a strong possibility that when the team has familiarized itself with agile and used it for some time, own practices and way of work are changed in process of time. And perhaps it is found out later that some other agile method would suit better for the team and development of the product. A big problem is that there is no method for choosing the right agile method. No perfect list or tool exists, which could help to choose the right method and proper practices. Every company and team selects itself its method and practices and then tailors them to meet the needs of the team. Best way to start using agile method is just select the method and start using it.

The project must be the reason to select a specific agile method. Organization structure cannot be the main criterion; on the contrary project size and complexity among other things should be deciding factors when choosing which agile method is taken in use. Organization is going to undergo in the end a big transformation due to agile.

Next the proposal for suitable agile method for the team is introduced. The proposal is based on the theory in the beginning of this thesis and the interviews.

6.1 Combining Scrum and XP

Scrum would be the most natural selection for an agile method to take into use in our team, since some competence to take it into use already exists in our team at the moment and it has been used in several other teams in our company. We could find a “Scrum mentor” from our company to guide our team into use of Scrum and give us help in troubled situations. This sort of a person is needed in the first projects, when the process is developed and tuned to meet our needs, and we do not ourselves have the know how to do the analysis and tune the process into the right direction. Mentor has to be someone outside of the team to be able to give fair recommendations how the team practices should be developed.

However Scrum is a framework for project management and it misses detailed practices for everyday work. For instance XP could provide some more detailed practices that could be useful and taken into use in development of our product. It is possible to complete Scrum with XP practices, and there exists several studies and user experience on the subject, which support the opinion that this might be good solution for our team also (Sliwa 2002). Testing and agile expert Elisabeth Hendrickson gave a lecture on 20.6.2007 and recommended combination of Scrum and XP for getting the best possible outcome from agile methods. For example ADM (Advanced Development Methods Inc.), the enterprise behind the Control Chaos web-pages, has developed XP@Scrum method. Figure eight describes the way these two methods are unified on XP@Scrum.

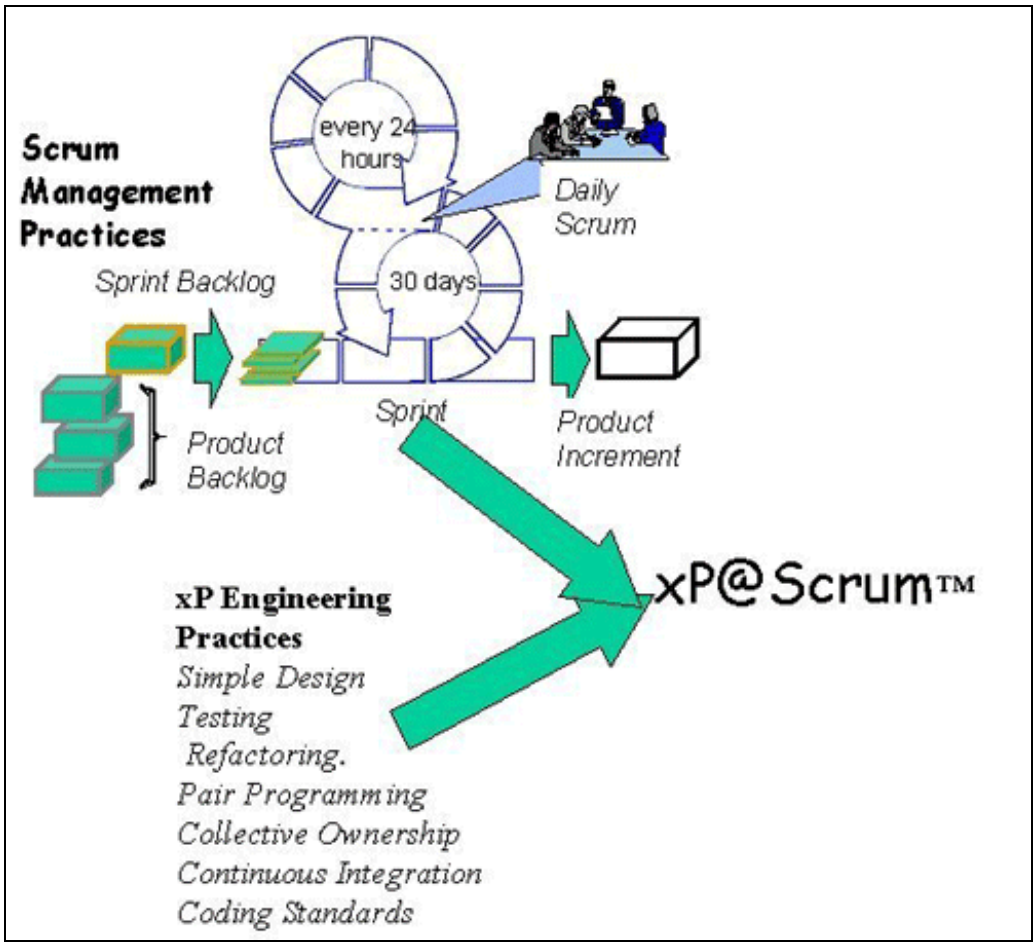

Figure 8: XP@Scrum (XP @ Scrum 2008)

In following chapters it is described what kind of effects taking Scrum and XP into use has on organization to which this thesis has been written and how the daily work of the team is going to change. In addition I have listed and explained those agile practices, which are in my opinion the most suitable to our team.

6.2 Effects on the organization structure and way of work

The largest change for our organization structure will be the cross-functional teams. At the moment the Research and Development (R&D) organization of our product has been divided into three different teams, two development teams and one testing team. When the transfer to Scrum is done the existing teams will be broken down to approximately seven people cross-functional teams. Each of these new teams will

consist of expertise and competence in different areas, implementation, documentation, testing, and architectural design. The challenge is that in these cross-functional teams every member has to be a real expert ones own field and also have some knowledge on the work of other team members. According to Scrum principles there are no specific roles in the teams. In my opinion this is not possible in practice, since in the development of a complicated product like ours it is impossible for everyone to be specialist on every area. In our team there are experts in specific fields, and that kind of expertise does not develop quickly, on the contrary it takes years to attain it. Also it is not possible that all scrum teams include all kind of roles (technical writer, architect, developer, tester etc.) because we do not have that kind of resources. So, specific persons must partake to several scrum teams' activities.

One big challenge is how to fit one specific team to the agile/scrum model. This team is development team, but it does not write code, but they produce so called configuration files to the product. Agile is meant to the pure coding team and for example almost all XP practices are suitable for the programmer (pair programming, continuous integration and daily builds). So, it is important to consider carefully how this team can act in the agile project.

Team members must be social and the atmosphere has to be good since tasks are performed together in the same office space. Changing the office environment to fit the agile requirements might be a quite big thing in fact. Agile addresses on co-operation and sitting in one shared space. Every team should work in one room and since the team size is quite large this requires open-plan offices. Nowadays the teams are mostly located in 2-3 person rooms and the resistance to move into open-plan may be considerable since people usually feel that the level of privacy is lower in open-plan offices.

At the moment one problem is the customer requirements handling. Product is not always meeting the needs and requirements of the

customer. The whole focus of agile is that customer is constantly part of the development process and should be even considered to be taken as one team member. In our case the latter idea is not applicable. Greg Larman suggested a solution to this problem in his lecture on 28.2.2008. In Scrum the Product Owner can act as “in-house” customer since product owner has all the information on customer needs and requirements. Despite of these facts the product owner cannot be constantly available for the R&D team to act for the customer. Therefore so called Product Owner Proxy (PoD) will work between the R&D team and the Product Owner. PoD works inside the R&D team and acts as messenger between the Product Owner and the team. This gives the teams a possibility to be constantly aware of the customer requirement changes and re-plan and adjust the work accordingly.

However, use of PoD does not solve all the problems with requirements handling. Product owner needs to be also aware of what has been done to fulfill the requirements and what the actual results are. Therefore in Scrum before every sprint, in Sprint planning meeting, the Product Owner, Scrum Master, the Scrum team, and possibly the customer representatives gather to decide the sprint targets. In this manner the Product Owner has an active part in developing the product and can affect on the level of customer satisfaction on the product.

After the sprint Product Owner holds so called retrospective together with Scrum Master and team members. In retrospective, members review the Sprint phases and evaluate the success and failures. When it comes to the quality, the most important feature is the demo after the sprint where the designer shows what has been done and how it works. Demo works as a prototype product to show what has been done during the sprint and to gain instant feedback from the customer if the designed feature fulfills the needs. This requires active participation of Product Owner since none of the designers is able to decide or to know about the real needs

of the customer. The product may work as planned and specified but if it does not work as the customer wants it is failed one.

The testing and the role of the test engineers in the organization will change a lot also. These changes are described in chapter 6.3.11 Testing.

Many of the practices in use nowadays will change when transferring to use of agile methods. These changes are overviewed in the following chapters.

6.3 Specified agile rules and practices

The principles of the Agile Manifesto are considered as compulsory when using the agile method, otherwise the method cannot be considered as one. However, as already seen from the variety of different methods presented in chapter 4.3, the practices vary from method to method.

Agile methods can be adopted in incremental manner, step by step. Everything does not need to be changed overnight and it should not, it is possible to define and take into use new practices when the needs are identified and the organization is ready for it. If we decide to use Scrum, complied with XP practices, the first step could be using the Scrum for project management. Some of the XP practices that are seen the most useful could be tried simultaneously.

Evidently it is also possible to do it the other way around, start with taking the development level practices into use first and then build Scrum management on top of that. The problem in this approach is that the R&D team is not using any of agile method practices at the moment. Also changing only some of the development level practices is not real agile since the customer interface and customer related practices would not change at all. It is better to start from the project management level

and in that way start to define the functionality of the whole organization and the teams in agile manner.

It is extremely important to analyze and define what practices will bring us the best result and not just to select the ones that feel the most comfortable and easiest to start with.

6.3.1 Continuous requirements' changes

Most important is the way how it is regarded to the changes. One main point of the agile is to accept continuous, mandatory changes and try to adopt them instead of trying to hold on to specification once planned. Changes must be managed carefully, all the changes are not needed to take into account and added to the product, but it is important to give chance to the changes. That is the reason why iterations must be short enough, so changes can be adopted at short notice.

Accepting the continuous changes might be one stumbling stone when moving to the agile. At the moment planning of the product is done in the beginning of the product. Of course changes are even now done quickly without long warning times, but usually changes are considered to be more or less negative things. That is why it requires big changes to the scheme of things to approve the fact that it is not always possible to know what will be done in the next sprint. Future cannot be predicted as exactly as earlier.

6.3.2 Short Iterations

Important and of course mandatory practice is short iterations. One of the main principles of the agile is that the product must be in delivery condition in regular, short intervals, at intervals of few weeks or maximum few months. In Scrum, iteration takes usually four weeks. After that actual and workable functions should be shown.

At the moment the product is done in several, partly overlapping projects, of which every one takes from few months to even one year. The demo is not given in the middle of the development process, but almost the first “public” demonstration is held when the testing phase begins. This happens very late in the project.

Demonstrations might be hard thing to approve for the developers because they have been used to that the product is not shown in public before it is ready for testing. Short sprints and demos after the sprints dragoon developers into showing the results of their work to the audience, even if the product is not complete at all.

The shorter the iteration, the better is known what the situation of the project is. The longer the iteration, the longer the correction takes if it is realized that the product is not working as it should work. Also the receiving the feedback quickly helps to reach the optimal results.

6.3.3 Product backlog

Product backlog is basically compulsory part of the product management and planning. Without Product backlog it is not possible to know what to do and planning the sprints and sprint backlogs cannot be done. So, Product backlog must be available already in the first agile project, it cannot be take in use sometime in the future.

Product owner owns the product backlog list and prioritize the items in the list. Owner makes the list of features and functions, which are wanted to include in the product. Product backlog is updated, modified and added continuously. All undone requirements are found from backlog. Items are removed from the list when they are added to the product. If items are not ready (malfunction or unfinished item) after the sprint, they will be returned back to the Product backlog list.

6.3.4 Sprint backlog

Sprint backlog is a list on items, which the team is going to carry out in that sprint. One interviewee said, that kind of official sprint backlog should have existed at the beginning of the agile usage, because it is the only way to know exactly which tasks are going to be performed in that sprint. Of course the list has existed earlier, but in black and white and near to hand it serves the purpose best.

6.3.5 Daily stand up meetings

At the moment each independent development and testing teams holds their own weekly line organization meetings, in which team members go through all finished and future tasks and other happenings. These meetings should be maintained (but shortened), because weekly meetings are good way to talk through all line organization related things. But in the future daily stand up meetings are arranged in addition to weekly meetings.

Daily scrum (in other words daily stand up meetings) is held every day, and it should take under half an hour. Every cross-functional team member participates to the meeting together with the Scrum master. Meetings are usually held in the same place and in the same time every day, usually in the corridor or common work area if available. Every team member is supposed to participate to the meeting every day.

Problems are not solved in the meeting, but extra meetings are arranged for solving those problems. In daily meeting every participant (excluding Scrum master) tells what he has done since the last meeting (held preceding day), what he is going to do today and are the any obstacles to doing those tasks.

But like one interviewee said, they found out that it was best for them to reduce the amount of daily meetings. Now they hold meetings only three

times a week. However it might be good idea to stick to daily meetings, because it offers good possibility to understand both own and other team members' work and what is still undone. Meetings do not take effective working time a lot, because it takes usually only few minutes per day.

6.3.6 Retrospective

Making the best benefit out of the mistakes made is to learn from them. One very useful feature of the agile is retrospectives. They improve the quality of the product and way of work.

At the moment, at least the verification team is using some kind of feedback giving. Team members send the list about experiences to the team leader after the project. How well those listed things are taking into account in the future projects is unknown.

In agile, retrospectives are more official meetings after the sprint, where team goes through all feelings and facts about what went right and what wrong. Teams are gathered together to some kind of workshop, where they are considering these things.

Retrospectives are held in agile more often than for example in traditional waterfall method; after every sprint. That way feedback is received in very quick rate and agile process at least in theory develops and gets better from sprint to sprint.

6.3.7 Pair-programming

Pair programming demands great changes in working methods for team's current members. Some of the developers have been developing the product almost from the start and they have worked with same component or sub area. Own strict area has been determined to every developer. So, opposite might exist, because not all employees want to work with somebody else.

One opportunity might be that every self-organized team could decide itself, that do they want to use this practice. Even inside the team it could be possible that part of the developers use pair programming. The others can work alone.

Pair programming can be recommended, because the quality of the program would certainly be improved and the amount of the faults would become less, because one feature of the pair programming is on-line and continuous code review. Other developer inspects while the other one writes the actual code. Also decisions and doings are considered together, alone might some details and point of views go over the board.

Pair programming demands constant presence, telecommuting does not work. However it is agile's one main principle that all operates nearby, by choice in the same room. Also working time of both developers must be alike; this might be a big problem in our team.

6.3.8 Common code ownership

Common code ownership might impress that chaos is guaranteed. Everybody has access to every code and every one can made chances when ever. Common code ownership has however so many good aspects that it is worthwhile to consider adopting it.

At the moment every team member is liable for own code, nobody else has access to the code. No one else can make corrections and changes to it. If the developer is for some reason incapable for doing changes to the code, work stands idle. Moving to the common code ownership helps maintenance and distribution of the code.

Pair programming is partly common code ownership, but it can be extended to cover also other team members. All team members are liable for the code; if the bug is noticed, it will be fixed. No matter who has written the code and the bug. But after every change, unit tests that

the developers have been written earlier must be run. That insures that the changes made are correct and functional.

6.3.9 Continuous integration

Continuous integration requires a lot of work before it starts to produce results properly, but it has so many advantages, for example better quality, that taking it to use must be considered. The amount of extra work should not be too much for the team, since the team will anyway do automatic daily builds after transfer to agile.

Continuous integration will anyway require change in working habits, since at the moment there exists no practices like constant integration and automatic testing at all. Writing and developing the automatic testing will increase the amount of work, especially in the beginning, but on the other hand it should decrease the time needed for debugging and we should be able to find the errors earlier than before.

6.3.10 Daily builds and smoke tests

The team has already started to plan how to perform automatic builds. There is a plan to start the builds in near future. Nowadays the package is mostly done in manual manner; one build may take even several hours depending on the amount of changes. Packages are done approximately once in a week. That is far too rare, what comes to finding errors and keeping up the quality. If the product is built for example only once per week and it happens to be broken, it might take several weeks before all the faults are found and the product is fixed again.

So called smoke tests that are the part of the automatic builds and confirm the functionality of the product are not in use at the moment neither. Building those test sets will take some time but so does the check of the build manually. When the automatic builds will start the manual checks become impossible since the builds will happen daily not weekly. There are no resources to do the job manually every day so the

automatic smoke test cases need to defined and test system for running those cases has to be developed.

6.3.11 Agile Testing

I had a possibility to talk with Graig Larman after his lecture. When I told him that I work as a test engineer, he told me that my job description will change totally after transfer to agile has been made.

Nowadays the team's testing process follows quite faithfully the traditional waterfall testing process except that the components are verified straight away when they are ready and all components' common release verification is the last phase of the project. So, testing phase is in the end of the project and all the faults are found very late in the project.

Currently the work mode is such that the test engineers work in their own team. Designers perform their own module tests and after that, it is the testing team turn to do their component verification and release verification tests.

One of agile principles is that everybody will do testing, not only the test engineers. The software has to be done from the beginning so that it is testable. Testing is constantly going on, greatly due to the continuous integration. Everyone at the team is responsible for the quality despite of what is ones title. (Hendrickson 20.6.2007, lecture.)

Hendrickson made a point on her lecture in Tampere 20.6.2007 on the fact that the test engineers need to be integrated, in a way, to the designers. They should work in the same office room and share the tasks of track testing and updating the programming status.

In agile, software is tested already in the beginning of the development work, throughout the lifecycle. One principle of the agile testing is that all code should always be tested. Functionality, which has not been tested, cannot be delivered under any circumstances. Second principle is that tests are written before or in tandem with the code itself. So, test cases

cannot be written afterwards. Third principle is that all team members, including developers and testers, write tests. And fourth is that automation must be used, it is not an alternative. (Leffinwell 2007: 156.)

Agile testing is divided in three different types, automated acceptance tests, automated unit tests and manual exploratory testing.

Programmers write the unit tests themselves, and they are always automated and thereby tests can be run often in contrast to manual tests. Unit tests can be also used with the continuous integration system so that every time integration is done, unit tests are run. (Leffingwell 2007:157.)

Acceptance cases are traditionally run at the end of the development (in our projects they are called component and release verification tests). In agile world acceptance testing are performed concurrently and incrementally, every time when new piece of functionality is added to the system. Acceptance tests determine if the product meets the expectations of the end user (so called black box test). (Leffingwell 2007:157.)

Manual Exploratory Testing takes place after the automated acceptance test. It provides additional feedback and all those parts of the program, that cannot be verified in automated test, are tested manually. (Hendrickson 20.6.2007, lecture.)

In addition of those testing practices mentioned before, it is possible to use for example a method called Test Driven Design (TDD). It combines test-first development (tests are written before the code) and refactoring. If anything, TDD is a programming technique instead of the tool for the testers. (Ambler 2007c.)

As a summary, the testing is going to change all out when moving to the agile. It demands hard work before testing automation will work properly and also the roles of the current testers will change totally.

7 TRANSITION TO AGILE

It is possible to transit from earlier development process to the agile process totally in one go or partly, little by little. It is not the most important thing to adopt individual practices but to accept the idea behind the agile. Moving to the agile is total paradigm shift to from one develop method to the other.

7.1 Adopting process

When moving from earlier way of doing software to the new method, many things are going to change. The biggest change relates to way of thinking and work, but also some concrete changes are going to happen. Teams and organization structure change, titles change, tasks change and even working area might change.

Before starting being agile certain things must be ready. It is not enough only start to use short iterations or other single agile practices. In following chapters few more or less mandatory cases are presented, which must be ready or done before agile process can be taken in use in the first project.

7.1.1 What must have been done

Few things must be decided and ready before starting the first agile project. In order to get the project work properly at full blast.

Team must be organized. In present team there are three quite big groups, which will be split to smaller teams when moving to the agile. Team also are formed again, there is no more separated development and verification teams, but in the future teams are going to be cross-functional.

Clarifying the roles of the members in the agile project is an essential part of the initial state of the agile project. Product Owner and other

project management members have to be known, and the customer or the representative of the customer need to be engaged with the project from the start. Also the members of the development teams have to be clarified, who is doing and what.

The schedule of the project has to be made before the project can be started. The length of the iterations must be decided and decision introduced to all project members.

Agile training for team members and other stakeholders must be arranged. Management and agile teams must know which are the main principles and practices of the agile, to be able to use the method.

7.1.2 What is good to be ready

Not all should be ready or clarified before the first project. However some things should at least thought through before shifting to the agile. If it is possible and if there is time, it is worthwhile to arrange following matters beforehand, so the moving process is easier and it does not take time in the project to arrange these things.

Open-plan offices or other common space are supported in agile. It could be good idea to arrange so that all team members would locate to the same room or near from the start.

Routine activities should be automated. Testing automation, nightly building and opportunity to continuous integration should be arranged all ready in day one. The pilot project serves no purpose, it these things cannot be tested and tried during the pilot.

In addition it would be good idea if project has its own wiki pages, which helps to share information between the all project stakeholders and inside the teams. We have good wiki pages inside our company and we can also utilize those pages.

In this phase it might be a good idea to decide also other practices, which are going to be used in the project. Of course some practices can be adopted also later on and in the pilot project practices can be tried, take in to use and dropped out. But in order for minimize the chaos of the starting phase; it could be good idea, if the developers know all the practices which can be used.

7.2 Pilot Project

It might be better to start with a minor pilot project instead of big, critical project. In pilot project agile practices and processes can be tried without fear that some important and valuable product for customer fails. In pilot practices can be added or removed if it seems that that practice is necessary or unnecessary for the team. Different length of iterations can be tested at the same time. Therefore the pilot project should not be too short, because if long enough, several iterations can be gone through, and the agile process can be developed from iteration to iteration. The problems noticed in the previous iteration can be corrected in following iterations. Also in pilot project agile processes can be tried to integrate with already existing processes.

In pilot project it is clarified the risks and problems which arise along with Agile and those problems can be solved before starting the actual first Agile project.

Piloting of the agile method should start by finding those people who are willing to try work with new method. When the pilot is over, these people can naturally tell to others how agile works for real, and what it brings along. Participants of the pilot project can also evaluate what kind of education is needed and how other employees can be introduced to the use of agile.

It is good to use the help of the external consult already in the pilot phase, if know-how cannot be found inside the company. Also the pilot

participants should study at least something about the selected agile methods and practices. It is difficult and frustrating to learn something totally new in the middle of the work. At least a minor theory base also decreases the change opposition. For example this thesis offers a compact learning material to study the basics of the agile methods.

Overall the pilot project can give and learn a lot. When it is performed and planned carefully, it gives a good base for the first real agile project.

8 CONCLUSIONS

There is no practical experience or specialists of the agile methods in the team. The theory part of this thesis provides a theory base what agile is and what it contains. It was hard and tedious process to wade through agile literature and information. This thesis offers complete information package of most important agile methods and practices.

The proposal to combine two different agile methods is not certainly to first nomination of the product management for the agile method to our team. Reason for my proposal is that this is the only way how we can obtain the agile tools to both management and development work. Other practices I proposed to our project can be of course taken in use, but these practices are good way to start. Practices do not make the team to be an agile. Iterative way of making products and adopting the continuous changes makes the team agile.

Although I interviewed only the employees of the one company, everyone stood for the different occupational group and point of view to the agile methods. It was interesting to hear opinions on for and against agile. I heard a lot of good comments about what should definitely be taken into account and what can be almost ignored. Every person had their own thoughts what good the agile method has been brought along and what bad. That way by interviews and theory learning I was able to perceive the best way on being agile.

The thesis has been done to our company and to our team to be precise, the outcome is however be generalized to other comparable organizations. However every organization, every work community, every worker and every project is individual. Thus the agile method should be tailored to every project, but the combination of the Scrum and XP can be widely used in different kind of projects.

Agile method is going to be taken in use in near future in the team. This thesis can be however developed further; it is possible to modify or expand this thesis by adding our own experiences of mobilization the agile method and make extensive handbook to the other teams and organizations. Further development can be the research of what the implementation of the agile method has been brought along, how the adopting of the method happened for real, and what could have been done better and where we succeeded.

List of references:

Abrahamsson Pekka, Salo Outi, Ronkainen Jussi, Warsta Juhani 2002. Agile software development methods. Review and analysis. Espoo: VTT Publications.

Ambler, Scott W 2002.
Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. New York: John Wiley & Sons, Inc.

Ambler, Scott W 2007a.
Agile Modeling and the Rational Unified Process (RUP). [online] [referred 13.01.2008].
http://www.agilemodeling.com/essays/agileModelingRUP.htm

Ambler, Scott W 2007b.
Agile Modeling Throughout the XP Lifecycle. [online] [referred 15.01.2008].
http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm

Ambler, Scott W 2005.
A Manager's Introduction to The Rational Unified Process (RUP). [online] [referred 13.01.2008].
http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

Ambler, Scott W 2007c.
Introduction to Test Driven Design (TDD). [online] [referred 23.03.2008].
http://www.agiledata.org/essays/tdd.html

Coram, Michael and Bohner, Shawn 2005.
The Impact of Agile Methods on Software Project Management. [online] [referred 06.04.2008].
http://ieeexplore.ieee.org/iel5/9677/30561/01409937.pdf?arnumber=1409937

Helesuo, Pekka 2004.
XP-ohjelmointi vs. RUP-prosessi. Pro Gradu. Joensuun yliopisto, Tietojenkäsittelytieteen laitos, Matemaattis-luonnontieteellinen tiedekunta. Joensuu.

Highsmith, Jim 2004.
Agile Project Management. Boston: Pearson Education, Inc.

Highsmith, Jim 2000.
Retiring lifecycle dinosaurs –Using Adaptive Software Development to meet the challenges of a high-speed, high-change environment. [online] [referred 07.01.2008]. www.jimhighsmith.com/articles/Dinosaurs.pdf.
- Software Testing & Quality Engineering, Vol. 2 Issue 4.
www.stickyminds.com/BetterSoftware/magazine.asp

Jeffries Ron, Anderson Ann, Hendrickson Chet 2001.
Extreme Programming Installed. New York: Pearson Education Corporate Sales Division.

Jeffries, Ron 2001.
What is Extreme Programming? [online] [referred 20.7.2007].
http://www.xprogramming.com/xpmag/whatisxp.htm

Larman, Graig 2004.
Agile & Iterative Development. A Manager's Guide. Boston: Pearson Education, Inc.

Leffingwell, Dean 2007.
Scaling Software Agility –Best Practices for Large Enterprises. Boston: Pearson Education, Inc.

Manifesto for Agile Software Development 2001.
[online] [referred 20.7.2007]. http://www.agilemanifesto.org/

McConnell, Steve 1996.
Daily Build and Smoke Test. [online] [referred 9.9.2007].
http://www.stevemcconnell.com/ieeesoftware/bp04.htm

Mitra, Amit, Gupta, Amar 2006.
Creating Agile Business Systems with Reusable Knowledge. Cambridge: Cambridge University Press.

Nelson, Scott n.d.(a).
Software Development Methodologies: Dynamic Systems Development Method (DSDM). [online] [referred 12.01.2008].
http://www.scottwnelson.com/article15-methodologies-dynamic-systems-development-method-dsdm.php

Nelson, Scott n.d.(b).
Software Development Methodologies: Feature Driven Development (FDD) [online] [referred 12.01.2008].
http://www.scottwnelson.com/article17-methodologies-feature-driven-development-fdd.php

Nelson, Scott n.d.(c).
Software Development Methodologies: The Rational Unified Process (RUP). [online] [referred 12.01.2008].
http://www.scottwnelson.com/article10-methodologies-rational-unified-process-rup.php

Principles behind the Agile Manifesto 2001.
[online] [referred 20.7.2007].
http://www.agilemanifesto.org/principles.html

Schuh, Peter 2005.
Integrating Agile Development in the Real World. Massachusetts: Charles River Media, Inc.

Schwaber, Ken, Beedle, Mike 2002.
Agile Software Development with Scrum. New York: Prentice-Hall, Inc.

Schwaber, Ken n.d.
SCRUM Development Process. [online] [referred 20.6.2007].
http://jeffsutherland.com/oopsla/schwapub.pdf

Sliwa, Carol 2002.
XP, Scrum Join Forces. [online] [referred 25.07.2007].
http://www.computerworld.com/softwaretopics/software/appdev/story/0,10801,69183,00.html

What is Scrum?
[online] [referred 25.07.2007].http://www.controlchaos.com/about/

XP @ Scrum.
[online] [referred 21.01.2008]. http://www.controlchaos.com/about/xp.php

Lectures:

Elisabeth Hendrickson 2007.
Agile testing. Lecture. 20.6.2007 Tampere

Graig Larman 2008.
Introduction to Lean, Agile and Iterative Development. Lecture.
28.2.2008 Tampere

APPENDIX A

Questions of the interview:

1. How long time your team have used Agile method?
2. Which Agile method your team is using? Are you using pure agile method or modified/mixed method?
3. What is your role in team?
4. Why just this agile method was chosen?
5. How this method was taken in to use? Gradually or at once?
6. How many persons are working in your team?
7. What kind of meetings do you arrange? Daily, weekly etc.
8. What are the meeting practices? Questions?
9. How long are the iterations?
10. What are the criteria of the iterations? What are the required results?
11. What good agile method brought along?
12. What bad agile method brought along?
13. What happened to the quality of the software? Improved or deteriorated?
14. Has the completion of the program accelerated?
15. Are you using pair-programming? Other practices?
16. What problems you faced up during the software development method replacement? How those problems could have been avoided?

17. In which level was the documentation before the agile method? How much implementation changes documentation required when agile method was taken in use?
18. How about the documentation now?
19. How exactly the criterias are adhered to?
20. How long period of transition was needed?
21. Did you use external consultant?