HELSINKI UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering

Mika Tanskanen

ORGANIZING OF FASTER RELEASE CYCLE – SOFTWARE DEVELOPMENT ASPECT

Master's Thesis

Espoo, January 28, 2008

Supervisor: Professor Marko Nieminen, D.Sc. (Tech.)

Instructor: Kirsi Lagus, M.Sc.

HELSINKI UNIVERSITY OF TECHNOLOGY Department of Computer Science and Engineering	ABSTRACT OF MASTER'S THESIS
Author Mika Tanskanen	**Date** January 28, 2008 **Pages** 109 + 11
Title of thesis Organizing of Faster Release Cycle – software development aspect	
Professorship User-Centred Product Development	**Professorship Code** T-121
Supervisor Professor Marko Nieminen, D.Sc. (Tech.)	
Instructor Kirsi Lagus, M.Sc.	

The aim of this constructive research was to research how large Research & Development (R&D) organizations could more dynamically and rapidly adapt to continuously changing market needs. Changing market needs bring changes to the content of ongoing product release programs, mostly in late phase of these programs. These changes involve the addition of some feature to the ongoing program or the removal of some already planned feature. As a result, the content of programs typically becomes too large, thus making it necessary to change software (SW) development and management in order to organize faster release cycles. This research focuses on organizing a faster release cycle.

Organizing a faster release cycle was found to be a topical issue at the Nokia Networks Business Unit, which subsequently became part of the Nokia Siemens Networks Corporation. This research focuses on two large R&D organizations and seeks to uncover the problems arising during SW development and management. Of particular interest in this thesis were those situations requiring changes to the content of programs at a late phase of development, as well as the strategies used to solve these problems. Problems with proposed solutions were collected during interviews that were arranged in the R&D organizations. The answers found during the interviews were compared with results from the literature discussing agile methods. This data was then used to create a construction. In this research, the term “construction” is used to denote a software process model.

This solutions identified in this research included automatic and continuous testing, minimizing parallel daily tasks, continuous competence sharing, continuous integration, incremental and iterative development of SW, backlogs for management and supervision of ongoing tasks, dynamics, taking the meaning of the communication into consideration, splitting of tasks into suitable parts, and suitably adjustable content in the programs. These solutions, together with other improvements would enable organizing faster release cycles through R&D organizations. Consequently, the eXtreme Faster Release Cycle (XFRC) process model was constructed from this research. Exploiting the XFRC process model would most likely enable a faster release cycle in both the target corporation and as well as many other corporations in a variety of different industries.

Keywords
Agile methods, faster release cycle, organizing, product release program, software development

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto	DIPLOMITYÖN TIIVISTELMÄ
Tekijä Mika Tanskanen	**Päiväys** 28. tammikuuta 2008 **Sivumäärä** 109 + 11
Työn nimi Nopeutetun tuotejulkaisuaikataulun organisointi – ohjelmistokehityksen näkökulma	
Professuuri Käyttäjäkeskeinen tuotekehitys	**Koodi** T-121
Valvoja Professori Marko Nieminen, TkT	
Ohjaaja Kirsi Lagus, FM	

Tämän konstruktiivisen tutkimuksen tavoitteena oli tutkia miten lähinnä ohjelmistotuotekehitykseen erikoistuneissa, laajoissa tuotekehitysorganisaatioissa (R&D) voitaisiin mukautua markkinoiden alati muuttuviin tarpeisiin entistä dynaamisemmin ja nopeammin. Markkinoiden muuttuvat tarpeet aiheuttavat muutoksia meneillään olevien tuotejulkaisuiden sisältöön, useimmiten niiden myöhäisessä vaiheessa. Tällöin meneillään olevien tuotejulkaisuiden puitteissa kehitettäviin tuotteisiin lisätään jokin uusi ominaisuus tai poistetaan tai siirretään myöhemmäksi tehtäväksi jokin jo suunniteltu ominaisuus. Seurauksena on se, että tuotejulkaisuiden sisältö kasvaa varsin usein liian suureksi ja ohjelmistokehitys sekä sen johtaminen asettaa muutospaineita nopeutetun julkaisuaikataulun organisoinnille, jonka suunnittelemiseksi tämä tutkimus osaltaan päätettiin tehdä.

Nopeutetun julkaisuaikataulun organisoinnin suunnittelun todettiin olevan ajankohtainen myös Nokia Networks – liiketoimintayksikössä, josta sittemmin tuli osa Nokia Siemens Networks yritystä. Tutkimuksen kohteeksi valittiin ohjelmistokehitykseen ja sen organisointiin liittyvien ongelmakohtien selvittäminen kahdessa laajassa R&D organisaatiossa. Ensisijaisina tutkimuskohteina oli tutkia mitä ongelmakohtia liittyy lähinnä sellaiseen tilanteeseen, jossa tuotejulkaisun sisältöön kohdistuu muutoksia lähinnä sen loppuvaiheessa ja miten kyseisiä ongelmakohtia voitaisiin ratkaista. Tutkimuksessa huomioitiin sekä ohjelmistokehityksen että johdon näkökulmat. Ongelmakohdat ratkaisuehdotuksineen kerättiin haastattelemalla henkilöstöä R&D organisaatioissa. Haastattelutuloksia peilattiin ketteriä menetelmiä käsittelevän kirjallisuuskatsauksen myötä löytyneisiin tuloksiin. Näin saatiin aikaan perusta konstruktiolle, joka tarkoittaa prosessimallia tämän tutkimuksen puitteissa.

Etenkin automaattinen ja jatkuva testaus, rinnakkaisten työtehtävien minimointi, osaamisen jatkuva jakaminen, jatkuva ohjelmistokomponenttien integrointi, inkrementaalinen ja iteratiivinen lähestymistapa ohjelmistokehitykseen, meneillään olevien ja tulevaisuuteen ajoittuvien tehtävien ym. hallinta sekä seuranta erilaisia listoja (eng. backlog) hyödyntäen, dynaamisuus, kommunikoinnin merkityksen huomioiminen, tehtävien pilkkominen riittävän pieniksi osakokonaisuuksiksi ja sopivasti mitoitettu tuotejulkaisuiden sisältö vaikuttavat olevan tutkimuksen perusteella osaltaan ratkaisuosatekijöitä. Kyseisiin osatekijöihin liittyvät kehitystoimenpiteet osaltaan mahdollistaisivat tehokkaamman ohjelmistokehityksen organisoinnin, joka saisi aikaan myös nopeutetun julkaisuaikataulun organisoinnin R&D organisaatioissa. Täten tutkimuksen lopputuloksena aikaansaatiin konstruktioksi eXtreme Faster Release Cycle (XFRC) prosessimalli. Kyseisen prosessimallin soveltaminen käytäntöön mitä todennäköisimmin mahdollistaa nopeutetun julkaisuaikataulun toteutumisen sekä kohdeyrityksen sisällä että myös monissa muissa yrityksissä, miltei toimialariippumattomasti.

Avainsanat
Ketterät menetelmät, nopeutettu julkaisuaikataulu, organisointi, tuotejulkaisuprogram, ohjelmistokehitys

PREFACE

This Master's Thesis was part of research at Nokia Corporation and this thesis was carried out in the Software Business and Engineering Laboratory of the Department of Computer Science and Engineering at Helsinki University of Technology (HUT) between autumn 2005 and autumn 2007. The idea of this research arose from the needs of Nokia Corporation to visualize faster release cycles. Therefore, this research was established in the autumn 2005. Writing this thesis was done besides a fulltime job at Nokia Corporation (since 1.4.2007 at Nokia Siemens Networks Corporation) and part-time studies at HUT. The thesis was completed in the autumn 2007.

This research would not have been accomplished without collaboration with many people from HUT and Nokia Corporation. Thus, I would next like to thank all those people who enabled this research by participating at different stages of the research.

First of all, I would like to express the very high appreciation to my supervisor, Professor Marko Nieminen from HUT, for his advice during this research.

I would like to express the extremely high appreciation to my instructor Kirsi Lagus from Nokia Corporation. I am truly grateful that she was able to dedicate her time to this research. I would also like to thank my previous boss Seppo Koivunen and current boss Jukka Seppänen, for their support and understanding for this research. Overall, I am grateful to my employer for giving me the opportunity to do this research.

Many interviewees and workgroup members in Platform R&D and Application R&D organizations devoted their time to participate to this research. I appreciate their attitude very highly and thus I would like to dedicate word of thanks to them. Unfortunately their names have been left out of this Master's Thesis for confidentiality reasons.

Research Professor Pekka Abrahamsson from VTT and Researcher Kristian Rautiainen from HUT have commented on second last version of this thesis. I would like to express my gratitude to them for their useful comments.

Finally, most of all I would like to express the highest appreciation to my lovely sweetheart Anne for the patience, encouragement she has given during this research. Furthermore, I am most indebted to Anne for reviewing linguistic form of this Master's Thesis.

Espoo, January 28, 2008

Mika Tanskanen

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AM	Agile Modeling
ASD	Adaptive Software Development
CMM	Capability Maturity Model (for Software)
CR	Change Request
CVS	Concurrent Versions System
DSDM	Dynamic Systems Development Method
FDD	Feature Driven Development
FT	Functional Testing (planning, execution and reporting)
HSDPA	High Speed Downlink Packet Access
HUT	Helsinki University of Technology
HW	Hardware
IDEAL	Initiating, Diagnosing, Establishing, Acting and Learning
IS	Implementation Specification
ISD	Internet-Speed SW Development
ISO	International Organization for Standardization
MI	Module Implementation
MT	Module Testing (planning, execution and reporting)
PP	Pragmatic Programming
Q&A	Quality & Assurance
R&D	Research and Development
RUP	Rational Unified Process
SD	Software Design
SEI	Software Engineering Institute
SEMS	Software Engineering Management System
SFS	System Specification
SPI	SW Process Improvement
ST	System Testing (planning, execution and reporting)
SW	Software
TDD	Test Driven Development
VTT	Technical Research Centre of Finland
XFRC	eXtreme Faster Release Cycle – process model, designed by M. Tanskanen
XP	eXtreme Programming

1. INTRODUCTION TO THE RESEARCH

Designer knows that perfection is not achieved when there is nothing to add but when there is nothing to remove.

Antoine de Saint-Exupéry

1.1 Background

Nokia Corporation initiated the subject of this research in order to find solution to the question: how to make release cycle faster regarding Change Requests (CR) related working that pertains to possible changes of content of release programs in late phase of the programs. The reasons for this research can be split into two categories. First category consists of market situation related reasons. It means that continuous learning, dynamical behavior and flexible adaptation for continuous changes regarding market needs are needed through Research and Development (R&D) organizations in contemporary growing global market area. This seems to be general problem in the telecommunication business, because the products might be based on new concepts that are always not known in detail from customers and corporation perspective. Second category consists of reasons that had been noticed according to corporation's internal needs. There was noticed that current release programs have too large content and management of large content takes lot of time, partially due to CRs, so search of alternatives for more efficient SW development including content management perspective were started. Thus, the Faster Release Cycle project was established in the autumn 2005.

Platform R&D and Application R&D organizations along with other R&D organizations were part of Nokia Networks business group which was one of business groups of Nokia Corporation during this research. Since 1.4.2007 these R&D organizations were included to be part of Nokia Siemens Networks Corporation. Operating on international telecommunications equipment market is the above mentioned organization's line of business. Members of those large organizations develop radio access, mobility core and wireless broadband network solutions for customers who are network providers and operators.

Representatives from both Platform R&D and Application R&D organizations participated in the project and few working groups started their activities as weekly workshops. Searching possible solutions how to make current release cycle faster was common denominator for these working groups. The working groups were split into six different categories: i) Continuous Technical Management, ii) Program and Project Management, iii) Software (SW) Development: Platform and Application development working groups, iv) Maintenance, v) Customer Documentation and vi) Hardware (HW) Development. Author of this thesis participated in activities with couple of SW development working groups: Platform and Application development working groups. Different activities with above mentioned working groups created basis and necessity of this research.

The research was decided to start and research material was decided to collect during literature review and interviews. At first the preliminary study was done by participate in activities of working groups and by discussion with few people working in R&D organizations. According to the preliminary study and aims of the Faster Release Cycle project, the research problem (chapter 1.3) was defined and detailed planning of this research was started. Also the aims (chapter 1.2) of this research were defined, scope of the research (chapter 1.4) was defined and research approach (chapter 1.5) was selected.

1.2 Aim of the research

The aim of this research was to find answers to research main question and sub-questions in order to solve the research problem (chapter 1.3) by constructing appropriate solution model. Hereinafter this solution model is called as an eXtreme Faster Release Cycle (XFRC) process model. Proposed process model was constructed on the basis of literature, interviews and pilot experiences. Usability was in major role during this research and process model was constructed from usability perspective as the usability definition says:

Usability is scope that describes a product can be used by specified users to achieve specified goals with productivity, efficiency and satisfaction in a specified context of use.
(ISO 9241-11 1998, p. 2)

1.3 Research problem

Research problem as main focus of this research was organizing of faster release cycle from software development aspect. In the beginning of this research both managerial and especially SW designers' perspectives were seen to be essential for organizing of faster release cycle. It meant study of how to organize especially SW designers' daily work so that release cycle could become faster. Constructing of new process model seemed to be one alternative to make release cycle faster. That process model would be dynamically scalable model e.g. for organizing of SW development through R&D organizations. In addition, it would not be depended of possible late changes to content of release programs. The organizing related results were decided to analyze from usability aspect. SW designers are "users" within this aspect and organizing of their work was mainly researched from this usability perspective during this research. Thus, SW designers could develop SW productively, efficiently and enjoyable to achieve specified goals in specified context during release programs when their tasks could be organized more suitable than before this research (cf. ISO 9241-11 1998, p. 2).

According to above, the research main question was determined as follows:

> *What kind of SW process model would enable the faster release cycle when content of on-going release programs is continuously changing in late phase?*

Before construction of the process model that enables faster release cycle information regarding current state of Platform R&D and Application R&D organizations was needed. To discover current state the experiences that describe possible problems with proposed solutions were needed to collect through these organizations. Mainly CR process and general SW development practices including problems with proposed solutions were researched. In addition, similar experiences of other corporations and basic principles for constructing the process model were needed to discover from the literature. Therefore, in order to construct solution to research main question the following three research sub-questions were formed:

1. *Which are the basic principles for constructing the process model that enables faster release cycle?*
2. *What kind of problem(s) pertain to ongoing product release program(s) when some feature is added to that program in late phase and how to solve the possible problem(s)?*
3. *What kind of problem(s) pertain to ongoing product release program(s) when some feature is (re)moved from that program in late phase and how to solve the possible problem(s)?*

Answers to first research sub-question were found along with the literature review (chapter 2). Answers to second and third research sub-questions are based on interview results (chapter 3). These answers were compared with the literature review results that describe above experiences and principles for constructing the process model. Due to analysis of above results, the XFRC process model (chapter 4) was constructed for organizing of faster release cycle. More details of this research are discussed in chapters 1.4 and 1.5. Detailed structure of this thesis is described in chapter 1.6.

1.4 Scope of the research

Literature review (chapter 2) and use of interview method (chapter 3.1) are main parts of this research. The scope of the literature review was limited to overview to agile methods and faster release cycle related experiences (chapters 2.2 - 2.4) of other corporations. Especially

Scrum and eXtreme Programming (XP) were selected under detailed research. Scope of agile method overview and related experiences was consequently limited to concern mainly XP (chapter 2.1.2) and Scrum (chapter 2.1.3) methods because Scrum was already under pilot process (chapter 3.4) in one SW development group in Platform R&D organization. In addition, many experiences of both methods are available. Detailed study of other agile methods (chapter 2.1.4) was mainly excluded in this research but overview to those methods is shortly discussed in chapter 2.4. The scope of interviews is discussed in details in chapter 3.1.2.

1.5 Constructive research approach

1.5.1 Motivation

One of research approaches is constructive research approach (Kasanen et al. 1993, pp. 243-264) and it was selected also for this research. Also e.g. case research (Järvenpää & Kosonen 2003, p. 19-20) and action research (Järvenpää & Kosonen 2003, p. 21-22) were other alternatives when selecting research approach. However, constructive research appeared to be a natural choice for this research because it is suitable approach when solutions model(s) are needed to be constructed for solving e.g. (SW) process related problems. The constructive approach provides clear phases for research process and provides possibility to select different research methods for the support of research. It also enables demonstration of solution, i.e. testing of suitability of construction. According to demonstration results, the solution model can be addressed to work in the real world. If some solution model does not work in the real world it can be modified and demonstrated after re-modification.

1.5.2 Research process flow

Constructive research process can be split to six main phases. The phases are described here (Kasanen et al. 1993, pp. 243-264):

1. Find a practically relevant problem
2. Obtain an understanding of the research topic
3. Construct a solution model
4. Demonstrate that the solution works
5. Show theoretical connections and research contribution of the solution concept
6. Examine the scope of the applicability of solution

At first the research problem (chapter 1.3) and related research main question with sub-questions were selected according to preliminary (chapter 1.1) research. Understanding of the research topic originated from own experiences and participation in *Faster Release Cycle*-project's workshops increased detailed knowledge. Constructing the solution model (XFRC process model) was done at the end of this research. Interpretations for the XFRC process model have been done according to literature review (chapter 2), interview results (chapters 3.2 and 3.3) and Scrum piloting (chapter 3.4) experiences. Actual demonstration that the above process model works was excluded because timing and scope of this research were limited. However, demonstration related further actions that are discussed in chapter 6.3 and experiences of Scrum piloting (chapter 3.4) belong to part of demonstration that the process model works. The interview results compared with the literature review results show theoretical connections of the XFRC process model. Scope of the applicability of the process model is discussed shortly in chapter 4 where the XFRC process model is illustrated too.

1.6 Structure of the thesis

This thesis reports the results of a constructive research process for organizing of faster release cycle. First the background, problems, aims, scope and approach of this research is discussed in this chapter 1. Chapter 2 contains literature review that was done by searching relevant data from standards, books and conference articles. The interviews were arranged in Platform R&D and Application R&D organizations. Fundamental research problem related interview results that were compared with the literature are discussed in chapter 3.

Literature review and interview results were summarized and analyzed. The XFRC process model has been constructed according to analyzed data. The XFRC process model is described in chapter 4. Conclusions of this research are described in chapter 5. Other discussion about this research has been written to chapter 6.

The structure of this thesis is clarified in Figure 1 below:

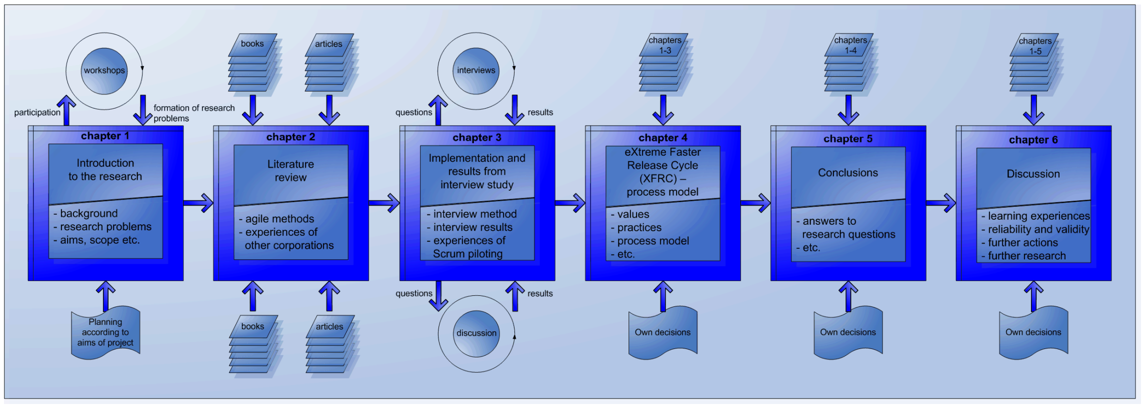


Figure 1. Structure of the thesis

2. LITERATURE REVIEW

So much has already been written about everything that you can't find anything about it.
James Thurber

This chapter introduces overview to Agile Modeling (AM), agile methods and related experiences of other corporations. This literature review was needed in order to create theoretical basis of agile SW development and to find answers to first research sub-question (chapter 1.3).

Overview (chapter 2.1) to AM and agile methods is based mostly on books and conference articles. The agile methods are e.g. XP and Scrum. In this research both of them were selected for scientific study because Scrum was already under pilot process in one SW development group in Platform R&D organization. Furthermore, many experiences from other corporations, which have piloted agile method(s) as among others both above methods for improving their SW process in their organizations, were available. In addition, the hypothesis was that e.g. combination of XP and Scrum might be suitable for organizing of faster release cycle. The best practices of few other methods have only been described shortly in this chapter.

Experiences (chapters 2.2 - 2.4) of other corporations are exclusively based on the research reports and conference articles. The experiences consist of well-tried practices and problems with proposed solutions, suggested by corporations, for those problems. Well-tried practices are summarized in Tables (4-9, 10, 13 and 14) and discussed in detail in text. The problems with proposed solutions are mainly listed only shortly in above Tables. The purpose of the experiences is to demonstrate how other corporations have used agile practices, i.e. what kind of practices have succeeded or failed during e.g. piloting of some agile method(s).

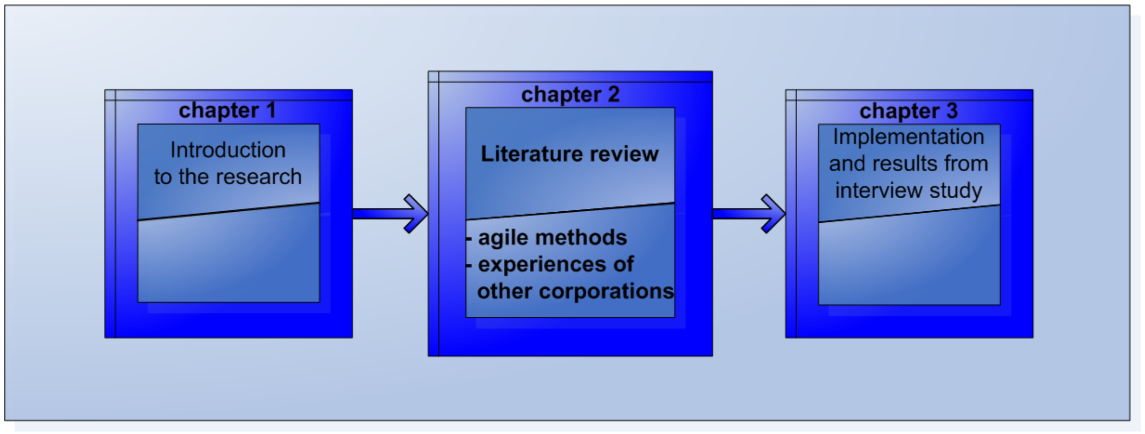

chapter 1
Introduction to the research
chapter 2
Literature review
- agile methods
- experiences of other corporations
chapter 3
Implementation and results from interview study

Figure 2. Introduction to chapter 2

2.1 Overview to Agile Modeling and agile methods

2.1.1 Agile Modeling

Agile Modeling (AM) is practice-based methodology for effective modeling and documentation of SW (Ambler 2007). In addition, it is way to model organizing team structures and ways of working with agile methods. One or more agile method (chapters 2.1.2 - 2.1.4) can be base process model of software development systems when AM is used. The AM methodology is a collection of practices, guided by principles and values. The underlying idea of AM is to encourage designers to produce sufficiently advanced models to support design needs and documentation purposes, i.e. amount of models and documentation is tried to keep as low as possible. (Ambler 2007; Abrahamsson et al. 2003)

2.1.2 eXtreme Programming

The eXtreme Programming (XP) is a lightweight process for small and medium sized teams developing rapidly changing requirements (Aarnio 2001, p. 18; Beck 2000). XP consists of shorter iterations than e.g. Waterfall Model (Royce 1970) and Iterative Model (Larman et al. 2003). Nonetheless, it contains same phases which can be e.g. System Specification (SFS), Implementation Specification (IS), Software Design (SD), Module Implementation (MI), Module Testing (MT), Functional Testing (FT) and so on. These development lifecycles are described in Figure 3 (Beck 1999).

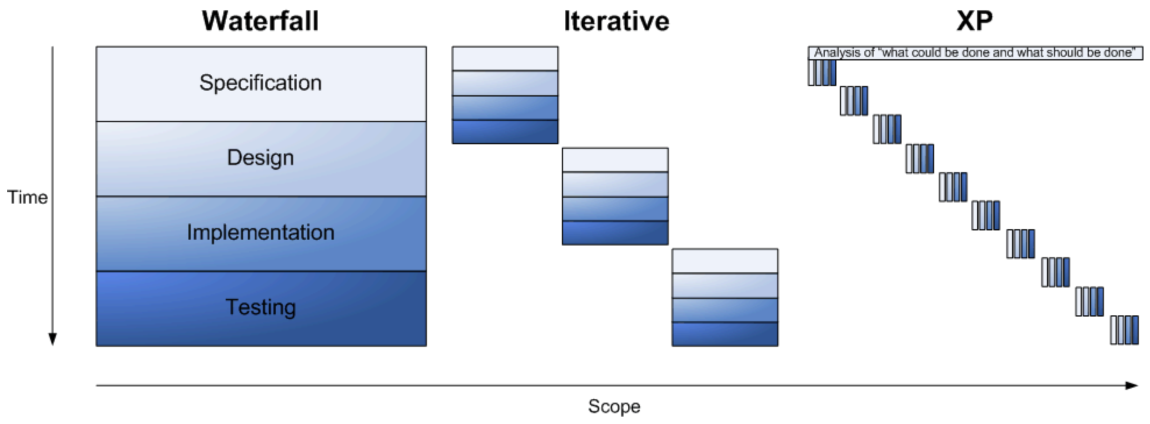


Figure 3. Waterfall, Iterative and XP lifecycles

The values of XP, as well as of AM values, are communication, simplicity, feedback and courage. Most common source of problems in any project is poor communication and XP encourages to open and honest communication. Simplicity brings many advantages: better testability, better readability of code, simple but not too simple documentation, short iterations with small releases (release programs), problem solving and re-factoring. Early and concrete feedback by customer enables controlling of efforts. Feedback is also relatively immediate through working team. Customer and team are typical main roles according to XP. E.g. programmer, testing person, coach and manager can belong to team.

Besides above values, during XP process also few key practices should be followed (Aarnio 2001, pp. 19-24; Beck 1999). Above simple documentation belongs to simple design that is one of XP's key practices. These key practices have been listed here:

1. Planning Game
2. Small release programs
3. Metaphor
4. Simple design
5. Testing
6. Re-factoring
7. Pair programming
8. Continuous integration
9. Collective ownership
10. On-site customer
11. 40-hours weeks
12. Open workspace
13. Just rules

Above practices and detailed description of the process phases is described in next sections in this chapter. In addition, these practices with entailed benefits are discussed in Appendix 1: XP practices.

Typical practice in XP is above pair programming in which two designers participate in e.g. MI-phase with each other at a single computer. Benefits of pair programming are: i) code reviews are effective in finding defects (cf. quality), ii) rapid feedback, iii) familiarity of e.g. program block increases and iv) changing pairs occasionally increases competence sharing. It is good to have experienced and novice designer working as a pair so novice's knowledge is increased e.g. in programming. When designer is working with other people trust is essential, e.g. when designer makes decisions or when some other people need help. Also courage is needed in order to reach aims of project. (Aarnio 2001, p. 18-24; Ambler 2002, pp. 19-26, 83-88, 131-133, 143-165, 185-187, 194)

XP process model is described in Figure 4. (Ambler 2002, p. 191)

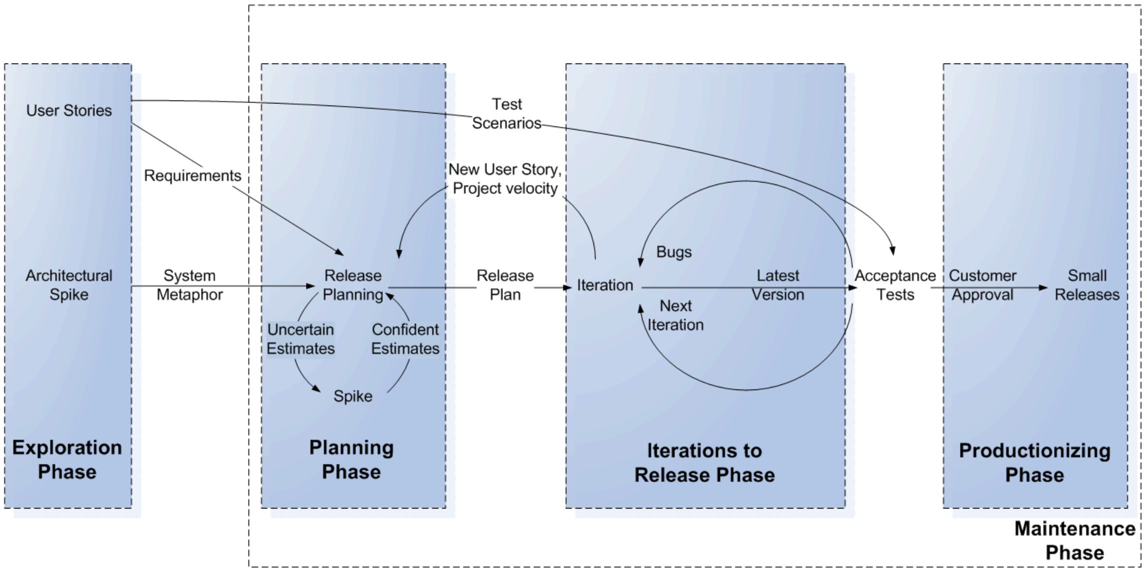


Figure 4. XP Process Model

At the start of the XP lifecycle, the architecture is selected and on-site customer picks the next release program by choosing the most valuable features (called user stories in XP). User story contains a name and short description of a needed feature. The customer (one on-site customer in this case) writes the user stories that enable designers (e.g. 10 people team) to estimate how long story or stories will take time to implement. This phase is called *Exploration Phase*. (Aarnio 2001, pp. 19-21, 24-27; Ambler 2002, pp. 190-192, 199-206; Beck 1999)

Exploration Phase is followed by Planning Phase (called also Planning Game). In this phase content of coming release program is planned according to prioritized user stories. According to estimations, the customer decides the scope and timing of release program(s). After that user stories are assigned to the iteration that contains typical SW process phases. This phase is called Iterations to Release Phase. During iteration team members implement their tasks that they want to be responsible for. This is possible because of collective ownership and team‘s possibility to make decisions regarding programming. The XP team meets daily (not at weekend or holiday) and designs what team members have done and what they will do before next meeting. This meeting is called Stand-Up Meeting. Daily work consists of Stand Up Meeting and breaking up team into pairs which meet in quick design session before implementation and testing etc. Code is implemented, tested and integrated daily and also re-factoring is possible if needed. Integration is not allowed until tests are executed. The unit tests (typically e.g. MT cases) are planned and implemented before implementation. It is called Test Driven Development (TDD) (Mugridge 2003). Overtime work is not allowed, i.e. 40 hours is maximum amount of work weekly. Typical XP designer's day is described in Figure 5. (Aarnio 2001, pp. 24-27; Ambler 2002, pp. 192-196, 214-222; Beck 1999)

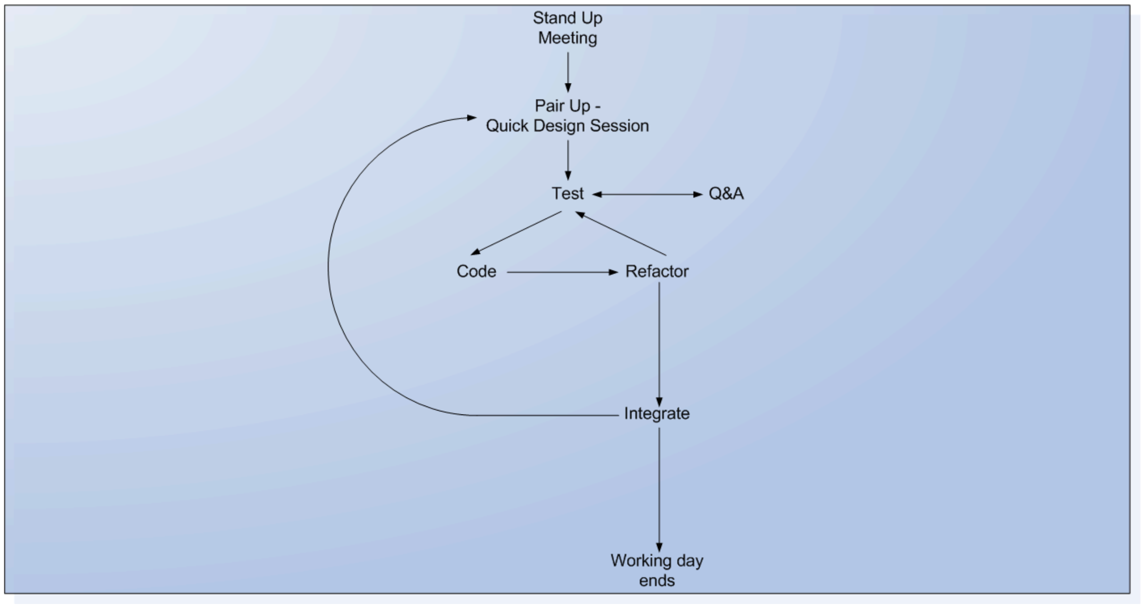


Figure 5. A typical XP designer's working day

After iterations the functionality is tested with acceptance tests. The customer has specified the acceptance tests while team has implemented *Iterations to Release Phase*. The customer is also responsible for verifying acceptance tests and reviewing test results. When functionality has been tested and approved system can be released. This phase is called *Productionizing Phase*. The Productionizing Phase is followed by *Maintenance Phase*. (Aarnio 2001, pp. 26-27; Ambler 2002, pp. 196-197; Beck 1999)

As above mentioned overview shows the XP process model contains five main phases and many useful engineering practices for agile SW development. Besides engineering practices also awareness about managerial and organizational aspects are essential so that it would be possible to organize of faster release cycle. Thus, Scrum paradigm was decided to select for scientific study in this research because it emphasizes exactly those aspects. Overview to Scrum is described in next chapter.

2.1.3 Scrum

Scrum is an agile, lightweight process that can be used to manage and control software and product development using iterative, incremental practices.
(Schwaber 2007a)

Scrum represents a paradigm shift in software development (Schwaber & Beedle 2002, p. 110). Scrum can be scaled through large, local and multi-site organizations (Schwaber 2003, p. 119-125) so it has been selected under study in this research. Scrum consists of practices of an iterative, incremental process that has been developed at Bell Labs in 1930s (Larman et al. 2003). The iteration (sprint) is heart of Scrum and output of fixed period, 30 days iterations, is an increment of product (Schwaber 2003, pp. 5-8, 136; Schwaber & Beedle 2002, p. 50). The increment of the product can be part of some feature. This kind of feature can be e.g. High Speed Downlink Packet Access (HSDPA) feature that is developed during SW product release programs in Platform R&D and Application R&D organizations. Any phase (e.g. IS, SD, MI and MT or SFS, IS, SD, MI, MT and FT and so on) can be included to all iterations.

Scrum roles can be described by three main roles: i) Product Owner, ii) Team (consists of 5-9 persons) and iii) Scrum Master (Schwaber 2003, p. 6). Responsibilities of those roles are described in Table 1 on next page and Scrum process model is described in Figure 6 below (Schwaber 2003, pp. 6-7, 9, 101-118, 142-143; Schwaber & Beedle 2002, pp. 8-9, 28, 31-32, 36, 38, 118, 140-141, 147-154)

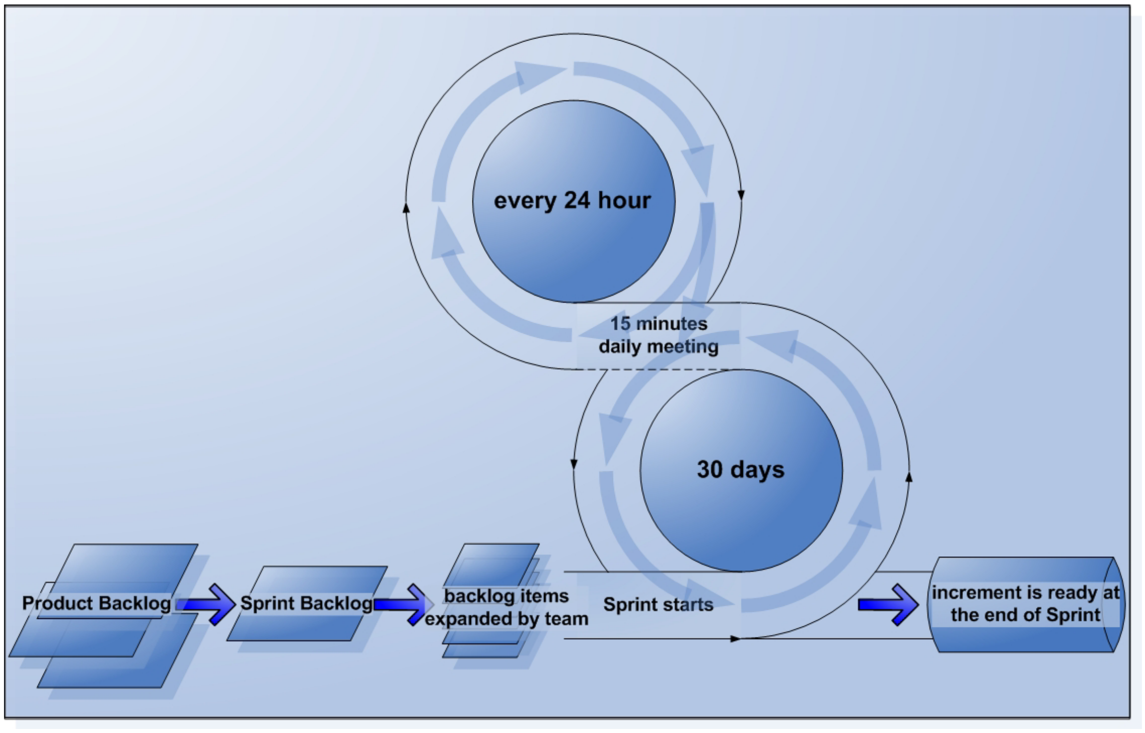


Figure 6. Scrum Process Model

Table 1. Scrum Roles and responsibilities

Scrum Role	Responsibility
Product Owner	- creating requirements using the Product Backlog - prioritizing Product Backlog contained requirements - participation in Sprint Planning Meeting, Sprint Review Meeting, Sprint Retrospective Meeting - conforming Scrum values (commitment, focus, open-ness, respect and courage)
Team	- building of functionality according to requirements - planning how to do development work (self-organizing team) - constructing and maintaining a list (Sprint Backlog) of tasks to perform during each sprint - using and conforming existing architectures, standards, technology etc. - conforming Scrum values (commitment, focus, open-ness, respect and courage) - collaboration with other Scrum teams that have same (Scrum) or similar (e.g. XP or some other suitable agile method) structures (cf. also chapter 3.4 contained ex-periences) - participation in Sprint Planning Meeting, Daily Scrum Meeting, Sprint Review Meeting, Sprint Retrospective Meeting
Scrum Master	- employing the Scrum process to build product (success of Scrum) - representing management and the team to each other - ensuring that impediments are promptly removed - collaboration with customer and management - teaching Scrum to everyone in the team - participation in Sprint Planning Meeting, Daily Scrum Meeting, Sprint Review Meeting, Sprint Retrospective Meeting - conforming Scrum values (commitment, focus, open-ness, respect and courage)

Start of the sprint, the team reviews and plans what it must do during iteration to be started. This review session is called *Sprint Planning Meeting*. It is similar than Planning Game in XP (Vriens 2003). It has two parts and one part takes about four hours. The first part is spent by presenting a Product Backlog to the team. *Product Backlog* (Appendix 2: Scrum's Product Backlog) contains list of changes that will be made to the product for future release programs. Changes can be constituted of e.g. functions, technologies, enhancements and bug fixes, i.e. all work that can be foreseen for a product. Product Owner collaborates with the team and they discuss what will be done for the next sprint. Product Owner owns the Product Backlog. During the first part of Sprint Planning Meeting, Product Owner presents a highest priority Product Backlog to the team. The team tells to Product Owner how much it believes can turn into a completed increment of product functionality by the end of the sprint. Also *Product Burn-Down* chart (Appendix 3: Scrum's Product Burn-Down chart) can be used to indicate remaining work concerning coming sprints. The Product Burn-Down chart is estimated in days. In consequence of above, managing the anticipated end date and progress of the entire project is easier. (Schwaber 2007a; Schwaber 2003, pp. 10-13)

When the highest priority Product Backlog has been selected and the team has decided what it must do during sprint, the second part of Sprint Planning Meeting can be started. During second part the team constructs *Sprint Backlog* (Appendix 4: Scrum's Sprint Backlog) and possible *Release Backlog* (Appendix 5: Scrum's Release Backlog) if needed. Also *Sprint Burn-Down* chart (Appendix 6: Scrum's Sprint Burn-Down chart) can be used to indicate remaining work in hours during sprint. Sprint Backlog contains estimations and task names for development work to be done during sprint. Sprint Backlog is estimated in hours. Release Backlog is subset of Product Backlog that is selected for a release. In other words, Release Backlog is the same as Product Backlog but restricted to a release program. Release Backlog is estimated in days. The reviewed Sprint Backlog is hanged across the wall of meeting room(s) that have been reserved for daily meetings arranged by every Scrum team. (Schwaber 2003, pp. 5, 8, 133-136; Schwaber & Beedle 2002, pp. 47-48, 71-72)

Ongoing sprint consists of development work and daily meetings. Sprint fixes typical project related variables: time available (every sprint is always 30 days), cost (in people and resources) and delivered quality (usually organizational standard). Also scope of functionality can be changed during sprint if needed, i.e. content of release program can be removed but not added. However, content of release program can also be added but only if the team allows that. Each Scrum team meets daily and one meeting (also called the *Daily Scrum Meeting*) takes time not more than 15 minutes. During the meeting, the team members tell what they have done since last daily meeting, what they will do before the next meeting and what obstacles have been noticed. Remaining work is written into Sprint Backlog which is updated by team members. Used time is not tracked in Sprint Backlog because the progress is followed mostly through the Scrum daily meetings (Itkonen et al. 2003). The estimation is the total estimated hours remaining, regardless of the number of people that are doing development work. Thus, meetings ensure that every member knows what everyone else is doing and thereby mentoring or collaboration is promoted. Also tacit and explicit knowledge can be found and perceived better due to knowledge sharing during daily meetings. (Schwaber 2003, pp. 8, 15-136, 141; Schwaber & Beedle 2002, pp. 33, 40, 43, 52, 72, 111-113, 118; Nonaka & Takeuchi 1995)

At the end of the iteration, the team presents the increment of working functionality it has built. This session is called *Sprint Review Meeting* and it takes about four hours. During Sprint Review Meeting the developed functionality is demonstrated to Product Owner and stakeholders (people with an interest in the outcome of the project) can inspect the functionality. Also timely adaptations to the project can be made. Any unimplemented functionality is re-entered onto the Product Backlog and it is reprioritized. After Sprint Review a *Sprint Retrospective Meeting* is arranged (see Appendix 7: Scrum Retrospective Meeting chart). Then during about three hours the team discusses about experiences regarding completed sprint and plans how to make things better during the next sprint. (Schwaber 2007a; Schwaber 2003, pp. 6, 101, 137-139, 142; Schwaber & Beedle 2002, pp. 53-56)

As above overview to Scrum described there are many managerial and organizational aspects in Scrum. Rules for changes of content of sprints, specified roles and responsibilities are included in those aspects. E.g. fixed period iterations (sprints), meetings, backlogs and continuous integration are practices that entail many benefits concerning those aspects. Scrum practices with benefits are also discussed in Appendix 8: Scrum practices. The aspects that were discussed in this overview to Scrum complement the engineering practices of XP and are essential part of organizing of faster release cycle.

2.1.4 Other agile methods

Also e.g. Adaptive Software Development (ASD), Crystal Family, Dynamic Systems Development Method (DSDM), Feature Driven Development (FDD), Internet-Speed SW Development (ISD), Pragmatic Programming (PP) and Rational Unified Process (RUP) are agile methods (Abrahamsson et al. 2003; Paetsch et al. 2003). Detailed study of those methods has been excluded from this research because enough information and similarities have been found already between XP and Scrum methods. In addition, larger scale research would be needed if other methods would have been decided to take under detailed study. Therefore, only the most essential values and practices of other methods are described in Table 2:

Table 2. Key practices of other agile methods

Method	Key Practices	More information
ASD	- incremental and iterative development with prototyping - short projects and release cycles - customer focus group review at the end of cycle	(Highsmith 2000)
Crystal Family	- incremental time-boxed delivery with prototyping - automated regression testing - reviews and workshops	(Cockburn 2004)
DSDM	- incremental development with prototyping - testing is integrated through the lifecycle - time and resources are fixed so scope of release program content can be adjusted	(Stapleton 1997)
FDD	- prioritized (by team) feature list - feature based development by small teams - 30 minutes weekly meeting	(Palmer & Felsing 2002)
ISD	- fast release cycles and short development cycles - based on "*Synchronize-and-stabilize*" approach by Microsoft (also "*Synchronize-and-stabilize*" approach contains good practices for organizing of faster release cycle like agile methods contain) - development with parallel release programs - customer involvement - stable architecture - reuse of SW components	(Baskerville et al. 2003) About "*Synchronize-and stabilize*": (Cusumano & Selby 1997; Cusumano & Yoffie 1999)
PP	- incremental and iterative development with rigorous testing and user-centred development - collection of short-tips that focus on day-to-day problems (about 70)	(Hunt & Thomas 2000)
RUP	- iterative development - Use Cases	(Hirsch 2005)

2.2 Experiences of other corporations – eXtreme Programming cases

2.2.1 Case: Ericsson

Ericsson Corporation is the largest supplier of mobile systems in the world. In 2004 one large design organization at Ericsson decided to improve their SW process by piloting of agile methods. The piloting was a part of the continuous SW Process Improvement (SPI) activity performed at the corporation. The aims of the piloting were: i) keep up the efficiency of the existing way of working, ii) competence build-up, iii) investigate possibility of using XP practices (mainly pair programming, Planning Game and collective ownership in this case) in the SW development, and to iv) study the impact of pair work on the motivation of the designers. Also on-site customer practice had been planned to use during piloting but it had been excluded because support for implementation phase was already got from the existing organization.

The experiences were collected during and after piloting, from minutes of the pilot (steering) meetings that were arranged weekly (one hour per week). The experiences were also collected by interviews of designers and testing personnel. Experiences that were noticed during piloting at Ericsson have been summarized in Table 3:

Table 3. XP related experiences according to piloting at Ericsson

Well-tried practices
- Planning Game (task planning) - Small tasks - Pair programming - Collective ownership - Sitting arrangements - Meetings - Code reviews
Problems with proposed solutions
- Problem: It was not always easy to see what should be done for a task. On the other hand, functionality was easy to leave out of the original tasks as the task specifications were too specific. - Solution: Designers chose to combine some tasks.

- Problem: Competence build-up can be needed at a time when project deadline is approaching fast.
- Solution: When time, competence and complexity aspects are taken into consideration competence build-up can be facilitated although project would be in late phase.

(Auvinen et al. 2005)

The article demonstrated that pilot project was success. According to the article, the main benefit of the piloting seemed to be competence build-up, i.e. SW development competence level increased. The Planning Game was the most beneficial. It gave clarity to the implementation work itself and furthermore helped in tracking the timeliness of a process.

Before piloting the data about the status of the project was vague, facilitation of (approximately weekly) pair switching was needed and features were needed to split into small tasks. Planning Game was seen to be one answer and it was held twice during this piloting. When Planning Game practice was used, work related to features was split into smaller tasks. Also complexity value (High, Medium, Low) was defined for these tasks. Due to use of Planning Game and smaller tasks tangible deliverables were got. The designers liked the fact that splitting of the work into smaller tasks made it more systematic. A task was defined as something which required from one to three days to implement. Feature was composed of few tasks and estimation concerned one week or maximum two weeks work. Better structured work content improved task identity. Task significance was increased as the knowledge on related tasks grew. Also motivation and satisfaction increased from the designers' viewpoint.

Pair programming was proven to be one of well-tried practices in this piloting. Pair programming was only recommended, not a rule, i.e. designers decided on whether to work alone or as a pair. In spite of that, many designers decided to participate the piloting of pair programming. According to results, pair programming was more efficient than working alone. It offered an efficient way of ensuring quality in an early phase, by having an extra pair of eyes checking the code as it was being written. It also increased the sense of team work and sense of responsibility. Furthermore, it facilitated problem solving and learning.

Collective ownership was implemented so that pairs do all the needed changes for a feature in two subsystems, not only to one subsystem as before piloting. The new approach seemed to help work allocation and to increase overall understanding of the architecture of both subsystems. Overall, using of collective ownership and pair programming in SD and MI phases was beneficial, also for use through large organization.

Communication improved between designers and testing personnel. All designers were sitting in the same room and pair programming made knowledge sharing of designers possible and improved SW development competence level. Pair programming also made knowledge sharing between designers and testing persons possible because there were at least two designers who were familiar with piece of code a testing person was asking about. Due to sitting arrangements and pair programming, feedback was fast and it increased during piloting. Also above mentioned pilot (steering) weekly meetings increased communication in general. During those meetings pilot steering group and designers discussed about: what features had been done, what features were currently ongoing and was the schedule kept.

In addition, due to use of agile practices the deliveries were made on schedule without deviations from the original estimations and quality of product increased. According the analysis that had been done during piloting, amount of fault reports had decreased with 5.5% because pair programming was used and code reviews were conducted.

(Auvinen et al. 2005)

2.2.2 Case: IBM

IBM develops and manufactures industry's most advanced information technologies, including computer systems, SW, storage systems and microelectronics. There are many own internal web development teams at IBM. In one team the traditional waterfall model was used for developing most web-based applications. The waterfall model seemed to be heavy-weight and decision making of requirements was slow through many boards. Decreasing development time and decreasing time to market were needed too so that release cycle could be made faster. Also more flexibility, decreasing costs and increasing user satisfaction were needed. Therefore, it was decided at IBM to pilot XP practices within piloting team.

In this case the piloting team consisted of designers (e.g. information architect and visual designer), project manager for webmaster programmers and project manager for the design group, usability specialists, coaches and a customer entity. Team members had read some of the XP literature and also training sessions had been arranged before piloting. After achieving basic knowledge of XP they believed that XP might work well in their development environment.

The potential of piloting was to make the development process more responsive to users' needs and changing business requirements. Assumption was that depending on piloting results the renewed development process with XP practices could make sense as a development methodology in a diverse web development environment in IBM. Experiences that were noticed during piloting at IBM have been summarized in Table 4.

Table 4. XP related experiences according to piloting at IBM

Well-tried practices
- On-site customer - Planning Game (partially succeeded) - Simple design - 40-hour week (sustainable pace, i.e. no overtime) - Small release programs and short iterations - Metaphor - Meetings - Coding standards (not formalized during piloting) - Pair coaching
Problems with proposed solutions
- Problem: Customer did not want to set release dates so they considered each iteration as a release. - Solution: They had convinced the customer about usefulness of multiple iteration release cycles.
- Problem: There had been no platform on which to share the tests, so each designer had a few tests that can't effectively be run by others. - Solution: This was addressed and solid testing architecture for tests was started to establish.
- Problem: The lack of experience with TDD practice. - Solution: Training sessions with hands-on demonstrations were arranged.
- Problem: Minor problems regarding re-factoring, pair programming, continuous integration and collective ownership were noticed. - Solution: Designer resources were increased. Since the designer resources were increased, pairing had increased to the extent that pairing had become less of an issue. Also other improvements were planned (e.g. use of pairing within code review process) and started.

(Grossman et al. 2004)

According to the article, XP made sense as a development methodology in a diverse web development environment. Also few problems were noticed, but improvements for solving those problems were started during piloting in the team.

On-site customer practice had worked extremely well. During piloting there was one on-site customer representative in the team. The customer had quickly learned how to write user stories for features during the Planning Game. It was observed to be an activity that offers good approach to produce user stories. On the other hand, the Planning Game partially failed. The team had hard time picking up the concept of “ideal time” for estimation purposes and most of staff had responsibilities for several projects. This was decided to improve in the long-term. In addition, beginning of a self-managing and self-organizing community with acceptance of a sense of shared responsibility was seen. Also team’s skills were developed and significance of estimations was emphasized within team. As a consequence of these improvements, the Planning Game became more efficient. (Later on, the team started referring to the Planning Game as “the Planning Process” because terms such as “Game” trivialize the seriousness of the activity).

The team was building SW with simple design practices what was needed “today”, keeping flexible for changes that will be done in the future. Design was kept as simple as possible and most of written stories were sized at around two ideal designer days. Simple design made a good fit with two-week iteration. Overtime work was not allowed during iterations.

Customer did not have any difficulties in prioritizing and selecting user stories for each iteration. Due to use of two-week iterations also staging was simple and quick, without requiring management, and yet able to respond quickly to changes. Furthermore, there was high degree of communication among the team both in the Planning Game and during iterations. Additionally sharing of metaphors through the team helped everyone on the team to understand the essence of the project. Overall, it was easy to convince everyone to use above iteration cycle and everybody were pleased with the results. Also three-week iteration cycle was tried but the results were not as satisfactory.

Coding standards were not formalized during piloting but it seemed to be a valuable (future development) practice and according to the article it could be taken into use later. Also few additional practices as stand-up meetings, iteration retrospective meetings and pair coaching seemed to be valuable practices according to piloting experiences. During meetings team members shared their experiences. They also got feedback and courage during those meetings. In addition, meetings were good channel from learning perspective as e.g. pair coaching practice that seemed to proven beneficial practice during this piloting.

(Grossman et al. 2004)

2.2.3 Case: Lund Institute of Technology

Lund Institute of Technology is located in Sweden. Within one programming course few XP core practices were studied as support for teaching the basic concepts in software engineering. These core practices were pair programming, Planning Game, TDD, small release programs and on-site customer. The course also increased the students (107 people on course) practical skills in testing, configuration management and re-factoring, which they can use in their in later courses. Also few additional practices as First Iteration, on-site coach, team-in-one-room, "Spike-time", reflection and documentation practices were added to concept of the course.

The course was organized in a theory part which was followed by a project part. During the theory part the XP practices were studied. During the project part students were grouped into teams (8-10 people per team) that did XP style project in six iterations during six weeks of study, i.e. fixed time box (planning time, Spike-time and lab-time). All teams developed essentially the same product (a system for tracking races in the motorcycle sport Enduro) during the course and were given the same set of requirements (user stories). All teams also had the same person as customer. Experiences that were noticed during piloting at Lund's Institute have been summarized in Table 5:

Table 5. XP related experiences according to piloting at Lund's Institute

Well-tried practices
- Pair programming - Planning Game - On-site customer - Small release programs - First Iteration - Configuration management using Concurrent Versions System (CVS) - On-site coach - Sitting arrangements - Collective ownership - Time (called Spike-time in the article) allocations for e.g. learning - Meetings
Problems with proposed solutions
- Problem: TDD mostly failed. In general the students had big problems in understanding how to write test cases in a useful way. On the other hand, some teams

wrote tests first as TDD suggested. - Solution: More training will be given during future courses.
- Problem: Based on experiences of few students sometimes there might be conflicts regarding how the code should be developed when working as pairs. - Solution: An experienced designer (e.g. coach) can try to solve conflict or meeting with other designers can be arranged.
- Problem: It was suggested that it is better to start with the simplest task and later evolve it to the more advanced ones, if needed. Thus, simple design practice failed. Some students misinterpreted “Do the simplest thing that can possibly work” to mean “Change as little as possible to incorporate a new feature”. - Solution: Examples and anti-examples were expected for the future courses.

(Hedin et al. 2003)

The overall results of this programming course were very positive. Researchers felt that students had good understanding for fundamental problems and techniques in SW development in middle size teams including testing, inspection, interaction with on-site customer, requirements and release process. During this course, students also learned how to use iterative method and were exposed to the problems of changing requirements. The experiences of this course are described in following paragraphs.

Some students had participated in another project where they had learned e.g. how to use pair programming in an educational situation. On the basis of their experiences, they advised that pair programming is good way to share competence and work efficiently. However, the designers should themselves decide who to pair with. The personality matching is worth considering when pairs are formed. According to them, personality matching seemed to be in higher role than competence matching when designers work as pair. In addition, changing of pairs was recommended because it is important from a learning perspective and team building.

In the beginning of the course, many students were skeptical regarding estimation of tasks without prior practical knowledge. However, most of iterations with Planning Game approach were succeeded because students achieved practical experience how to estimate new tasks in compliance with the user stories. Customer wrote these user stories before each iteration.

During the course students made three small release programs – one every two iterations. The practice First Iteration (so called “Zero Feature Iteration”) made possible to get early information on the system architecture. Due to use of that practice, it was possible to evolve the architecture at early stage before too much time had been used to build functionality. The on-site coaches were let to build a tiny first iteration of the system because experience of the system was needed before start of a real first iteration. After the First Iteration the students were able to start immediately by adding functionality to the existing system. The coach tracked of what team members were doing during iterations.

At start of iterations two hours long planning meetings were arranged with coaches. There it was discussed about experiences from previous iteration and on-site coach told instructions for next iteration. During each planning meeting, time was allocated for e.g. reading and learning about various issues, thinking about solutions for particular task, learning about SW tools, re-factoring, code reviews and doing experimental programming with new solutions. Above time allocations called Spike-time in the article.

Customer visited planning meetings and participated in evaluation of releases. An advantage of this was that the students were forced to see the release from the customer's viewpoint and students also found realistic illustration of how requirements can unexpectedly change. They also learned the problems which occurred when many designers were changing the same source code. Configuration management (e.g. versioning and merging of code) using CVS was proven to be one mechanism for handling of those problems. CVS had been set up and sharing of result was easy with team members and frequent synchronization of code succeeded. Also re-factoring, separated tasks and short check-out times proved to be beneficial mechanisms for handling of above problems.

In addition, communication and increase of team effectiveness seemed to be in a major role. Spontaneous reflection (e.g. stand up meetings) that occurred in some teams, helped in team building, as well as improved the effectiveness of the team. Effectiveness was increased along with various sitting arrangements. Due to these arrangements, the team was present in the same room. The students communicated easily and getting of help was fast. Overall, there was high interaction and collaboration within the team. Furthermore, keeping the whole team in one room seemed to help their success also with collective ownership of code.

(Hedin et al. 2003)

2.3 Experiences of other corporations – Scrum cases

2.3.1 Case: AG Communication Systems

AG Communication Systems is a middle-sized corporation in Arizona. The corporation is experienced in SW development in the telecommunications market and there are several hundreds of employees working in different size teams. The teams are working with different software development project tasks. Some projects were more successful than others so learning the secrets of those successful projects started. In organization it also was decided to research implementing of their SW development process in order to better meet shifting business demands. About that time, they discovered the Scrum and it seemed to overlap significantly with what they saw in consequence of successfully projects. Due to this observation the piloting of Scrum was started in three diverse SW development teams (A-Team, B-Team and C-Team). Experiences by AG Communication Systems have been summarized in Table 6:

Table 6. Scrum related experiences according to piloting at AG Communication Systems

Well-tried practices
- Meetings - Small tasks - Prioritized backlog(s) - Small teams
Problems with proposed solutions
- Problem: It was not easy to spend lot of time in daily meetings. - Solution: Two or three meetings per week seemed to work best.
- Problem: Some designers had difficulties in planning their workload. - Solution: Sprint goals increased awareness of expectations and "kept people on track".
- Problem: Management might ask designers to other projects and designers might have many (other) parallel tasks. - Solution: The team decided to modify the backlog to reflect new strategy and designers focused exclusively on the sprint. The team also noticed that Scrum would work best when each designer focused exclusively on the sprint.

(Rising & Janoff 2000)

According to experiences of three teams, Scrum was appropriate for projects where requirements can't be defined with details and chaotic conditions are anticipated throughout the product development life cycle. After all they found that small teams can be flexible and adaptable in defining and applying an appropriate variant of Scrum. Especially meetings proved to be beneficial practice during this piloting of Scrum.

A-team decided to pilot a one-month sprint. At first, a short Scrum presentation was arranged and daily meetings were started. Scrum master facilitated meetings. The team began to spend time in daily meetings to increase knowledge of Scrum and learned how this team would adapt Scrum to its application. From the start of piloting the team had difficult to spend lot of time in daily meetings. Later on the team decided to arrange meetings every other day, i.e. three times per one week and two times on following week. It seemed to work best. During these meetings increase of co-operation and volunteerism were evident as successes of small tasks that were completed during sprint. Tasks included all design-cycle phases. Team members took an interest in each other’s tasks and were more ready to help each other in their problems. Detailed problem solving did not happen during these meetings but discussion about details was done after meetings. After all the A-team completed a successful sprint and delivered the planned components on time.

B-Team found itself in situation where corporation made major strategic change into team’s first sprint in the beginning. It affected the Sprint Backlog contained tasks and team decided to modify the backlog to reflect the new strategy. The tasks were prioritized for each sprint and sorted by person. Designers also participated in other projects and it caused that team over-committed itself. Meetings, rapid feedback and prioritizing of tasks helped to keep designers commitment aligned with the goals of sprints. Team also noticed that Scrum would work best when each designer focused exclusively on the sprint. Overall, B-Team completed a series of increments during sprints.

C-Team tried applying Scrum for preliminary testing and bug fixing of the feature that the team was responsible for. There were scheduled few testing times and a half-hour meeting was arranged before and between each test time. During these meetings, team redefined some testing procedures to make things faster. Meetings let all team members to know what was planned and volunteer to work next testing time. Also possible problems were discussed during meetings and Scrum master told information outside the team to team members. Thus, meetings increased communication and competence sharing. Meetings were also efficient way for tracking progress. At the end of sprint the feature was ready for integration testing and the team had achieved its goal. C-Team was willing to continue use of Scrum practices also on their future release programs likewise A-team and B-team.

Besides above experiences, Rising & Janoff (2000) suggested that product development and marketing groups must work together to provide the features with the highest value for the release programs. They also suggested that marketing should prioritize the features and product development should provide estimates for the effort. Marketing and product development should also agree target set of the features.

(Rising & Janoff 2000)

2.3.2 Case: University of Ljubljana

The University of Ljubljana situates university in Slovenia. Designers at university developed their own SW for university information systems because the systems were so specific that there were no commercially available solutions in the marketplace. Making SW development more visible and adaptable they were looking for an agile approach for project management. Therefore, they decided to pilot Scrum within one project that consisted of three sprints in this piloting.

During piloting was developed so called maintenance module that enables the maintenance of all data required for student record information system. Three designers and Scrum Master (also working with Product Owner related tasks) belong to the piloting team and the project was managed by above mentioned Scrum Master.

Well-Tried practices related experiences that were noticed during piloting at University of Ljubljana have been summarized in Table 7. Significant problems regarding this piloting had not been described in the article.

Table 7. Scrum related experiences according to piloting at University of Ljubljana

Well-tried practices
- Prioritized backlog(s) - Meetings (e.g. communication and competence sharing increased)
Problems with proposed solutions
No significant problems were described in the article.

(Mahnic & Drnovscek 2005)

According to the article, only well-tried practices were described and overall benefits of the Scrum practices seemed to be very useful. The use of Scrum improved communication and maximized co-operation. Team members were able to organize their work by themselves considering their preferences and special knowledge. There were prioritized Product Backlog and

Sprint Backlog for planning and estimation of goals. Nobody was allowed to change the sprint goals. Team members maintained the Sprint Backlog and participated regularly in daily meetings that allowed everyone in the team to see status of all aspects of the project in real time. Everything was visible to everyone. The problem resolution and clearing of obstacles were the best parts of meetings. Observation and advantage of the team's experience and ideas became as well possible due to these meetings. Also volunteerism, communication, competence and motivation within team increased. Furthermore, team members were taking an interest in each other's tasks and they were more ready to help each other.

At the end of each sprint, a sprint review meeting was arranged and team demonstrated the tested increment of product functionality. The team, Scrum master and few end users participated in these meetings. Consequently, end users saw functionality of increments and were able to suggest possible elaborations to the product. At the end of the project, team members felt good about their job and contributions. In their opinions they had done the very best they possibly could. From management point of view it was good that the SW development process became visible, controllable and manageable.

(Mahnic & Drnovscek 2005)

2.3.3 Case: Solystic

The Solystic Corporation builds automated electronic systems for the engineering, manufacturing, project management and logistic support personnel. Due to different business critical reasons the Solystic developed its SW development process with agile practices and piloted Scrum in their organization. Also other agile methodologies such as Crystal Family had partially been included to that piloting. An average 20 persons participated in the piloting team that was in charge for developing complex and feature rich information system. The communication settings, the planning techniques, the development practices and the team structure had been selected for being major parts of the piloting. Experiences that were noticed during piloting at Solystic have been summarized in Table 8:

Table 8. Scrum related experiences according to piloting at Solystic

Well-tried practices
- Time-boxed short (30 days) iterations - Use Case based incrementally SW development - Iteration planning - Daily builds (partially succeeded)

- Continuous testing - Automated regression testing - Meetings - Scenario-based software architecture evaluation techniques
Problems with proposed solutions
- Problem: During meetings, it was difficult to tell other team members publicly what was going wrong. - Solution: Courage was necessitated and it was necessary to give lot of energy to develop proper culture to inspire confidence in the team. In addition, to handle uncomfortable situations, the meetings with simple format were suggested: o Firstly, address the emotional part by giving each participant the opportunity to express how they feel about the situation. o Secondly, try to state the problem and find a solution.
- Problem: The estimation process and techniques had not been well known among designers in self-organizing team. - Solution: Time management skills were decided to teach more.
- Problem: Daily builds were not succeeded. - Solution: Builds after few days seemed to succeed better than daily builds. Almost daily builds were suggested and configuration management was tried to improve.
- Problem: Short iteration reinforces the risk to be shortsighted. - Solution: Architecture review addresses possible problems within large-view scope (overall project and business).

(Derbier 2003)

Piloting of Scrum seemed to bring along many good practices for developing SW process at Solystic. Also few other practices seemed to be beneficial but mainly Scrum practices had been piloted. Especially time-boxed short (30 days) iterations, meetings, iteration planning, face-to-face communication that enabled fast feedback, the (almost) daily build, and automated regression tests proved to be beneficial practices. The estimation, planning, build and delivery processes, and scenario-based SW architecture evaluation techniques (i.e. requirements can turn into user stories like in XP) were also partially discussed.

A half-day Iteration Retrospective meeting was arranged at the beginning of the each iteration. Initiative iteration planning was done during these meetings. Configuration management had been one hot topic in the meetings. The team also found that it was necessary to spend time to discuss very general and fundamental subjects about SW development, what was on-going and what possible fears (e.g. what was going wrong) were noticed within team.

Approximately 30 minutes Scrum meetings were arranged every other day during these iterations. The meetings were arranged in a dedicated room. Detailed planning and tracking of iteration was done with collection of notes on a whiteboard. The notes contained the title of features that were realized with Use Cases. The Use Cases were estimated in days and were assigned to someone in the team. There was also the Product Backlog in order to track for the estimation at completion. Furthermore, the team used whiteboard for writing the decisions that must be taken outside Scrum meeting. In the next meetings it was discussed what decisions had been taken. This ensured synchronization of decisions. Also different improvements were written on the whiteboard and discussed during these meetings.

SW was developed incrementally based on above Use Cases. Installing of automated regression testing practice was suggested to start as early as possible. The unit testing had been started almost at the beginning of the project and it seemed to work well. Instead automation of functional tests had been started later. However, testing of SW was continuous and regression tests were automated. The team was also asked to give percentage estimation of how much tests had already been automated and this number was made public.

The end of sprint seemed to be hard because it was difficult to find the appropriate level of pressure in order to meet project deadlines without setting up a new project. On the other hand, it was noticed that software designers are optimistic and tend to over-commit. It was suggested that repeatedly make the people understand that the end is duration and not event, and to understand what happens during the end.

(Derbier 2003)

2.4 Experiences of other corporations – other cases

2.4.1 Case: Philips

Philips is one of the world's biggest electronics corporations. It produces e.g. televisions, audio & video systems, monitors and household products. At one department of Philips it was decided to achieve certification of Capability Maturity Model (CMM) (Raynus 2002) level 2 and ISO9001:2000 (Schoonmaker 1997) in two years by using agile methodologies. The department is part of Philips Research organization that carries out SW projects in research domain.

Delivering time (release cycle), quality and scope of functionality were identified to be the critical success factors for their projects. For getting CMM level 2 and ISO9001:2000 also minimum bureaucracy, usage of practices and process transfer through other Philips organizations in later phase were starting points for maximizing the success factors and reaching their CMM and ISO goals. After studying few methodologies they decided to take XP as their start point and piloting of it started. After one year piloting period and work with XP, also Scrum was decided to merge with XP. In consequence of merging, combination which is called XP@Scrum (Schwaber 2007b) was taken into piloting. Reason of merging was that XP focuses on engineering practices and Scrum focuses on managerial and organizational aspects. In addition, XP did not give much help regarding documentation, modeling and the use of UML and design patterns. Complement of XP practices was also studied from books dealing with e.g. different patterns and other agile methodologies. As a consequence of this, they learned more about e.g. documentation, modeling, use of UML and design patterns too. Experiences that were noticed during piloting at Philips have been summarized in Table 9:

Table 9. Combination of agile methods related experiences according to piloting at Philips

Well-tried practices
- Time-boxed short (30 days) iterations - Small teams - On-site customer - Small tasks - Meetings and reflection workshops - TDD - Pair programming
Problems with proposed solutions
- Problem: Every designer or customer didn't appreciate emphasizing verbal over

written communication as XP and Scrum recommended. - Solution: Respect regarding diversity was suggested.
- Problem: XP and Scrum did not contain direct and adequate support (in CMM and ISO terms) for Quality & Assurance (Q&A) and configuration management. - Solution: Quality officer was selected to work as a coach and support management.
- Problem: Proper risk assessment was needed because only risk identification had been done in the organization. - Solution: Studying how to improve the assessment of risks with e.g. impact analysis was started.

(Vriens 2003)

According to article, applying of XP@Scrum practices can be succeeded if organization takes the combination into use (piloting) without prejudice and adapts practices for needs of their organization. Piloting proved that applying XP@Scrum practices had mainly succeeded. Many practices had been proven to be beneficial from perspective of SW process development as well as achieving of certifications of CMM and ISO9001:2000.

Both on-site customers and designers appreciated working iteratively making small increments. Early delivery of business value and learning by experience were mentioned to be certain senses regarding success of incremental SW development. Both senses decreased stress and increased confidence as the requested functionality was delivered incrementally. Customer decided what was going to be delivered and in what order after hearing feedback from the designers and accepting the consequences. Therefore, customer made decisions on what will be implemented in particular iteration and what not. Designers decided how to develop SW and participated in daily meetings during iterations.

SW was developed within small teams during one month (30 days) time-boxed iterations which proved to be good rhythm. At the beginning of iteration new requirements in the list of user stories had to be identified and at the end of the iteration these requirements had to be approved. Later on, also problem reports and CRs were included to this list. Pair programming was typically used for resolving above mentioned problem reports and for typical programming related tasks.

Small design sessions were arranged before a new task. If the conclusion of this session was that needed code seemed to be relatively simple or routine to implement the code was allowed to write solo. Otherwise, pair programming was used for writing code and writing tests with TDD practice. When tests were written first it forced designers to look at the code from the viewpoint of end user in higher abstraction level. It was practical way to obtain the principle of low coupling and high cohesion because tests had to be independent.

Although use of pair programming brought many benefits it seemed to cause some extent prejudice from perspective of customer. At first customer had rejected pair programming because of the belief that costs would double. However, after seeing the benefits in the form of higher defect prevention (and removal) and decreased release cycle time customer accepted pair programming.

At the end of iterations the review meeting was arranged and the team demonstrated currently implemented functionality to the customer. They discussed about acceptability of functionality during these meetings. In addition, the reflection workshops were arranged to increase maturity and effectiveness of the team. It was suitable way to communicate best practices between the various teams too.

(Vriens 2003)

2.4.2 Case: Avain Technologies

Avain Technologies has experienced in building secure digital transaction solutions. The corporation’s strategy focus was to move to the product business and grow the corporation to become a global player in the market. This meant increasing amount of designers and other personnel and strict requirements for SW quality. The need for more defined SW development process was evident. However, the corporation did not want to lose its values as flexibility, efficiency and innovative work culture. About that time, Software Engineering Management System (SEMS) research project (Rautiainen & Lassenius 2001) had been started at HUT and participants in different roles from Avain Technologies participated in the project too. Cycles of Control framework was created in this project and new approach based on that framework was piloted at Avain Technologies.

The Cycles of Control framework provides a common language which contains the practices, pacing and phasing of the incremental SW development process. An iterative and incremental development process recommends that working SW would be delivered early to get user feedback in good time. At the same time technical feedback with relation to non-functional aspects (e.g. system performance) can be made available.

The framework consists of cycles that are described in Table 10:

Table 10. The Cycles of Control according to SEMS approach

Name of cycle	Description
Portfolio Management	- provides the interface between business and product development - provides guidelines for how product development efforts should be organized in terms of SW development cycle and release cycle - manages the set of product and services offered by the corporation, e.g. by product roadmapping
Release Project Management	- handles the development of individual product versions
Increment Management	- manages incremental development of product functionality within release programs
Heartbeats	- provides scheduling and monitoring for daily or weekly tasks, and synchronizing the effort of designers or teams to get an indication of system status during development

The Cycles of Control framework by SEMS approach is described in Figure 7:

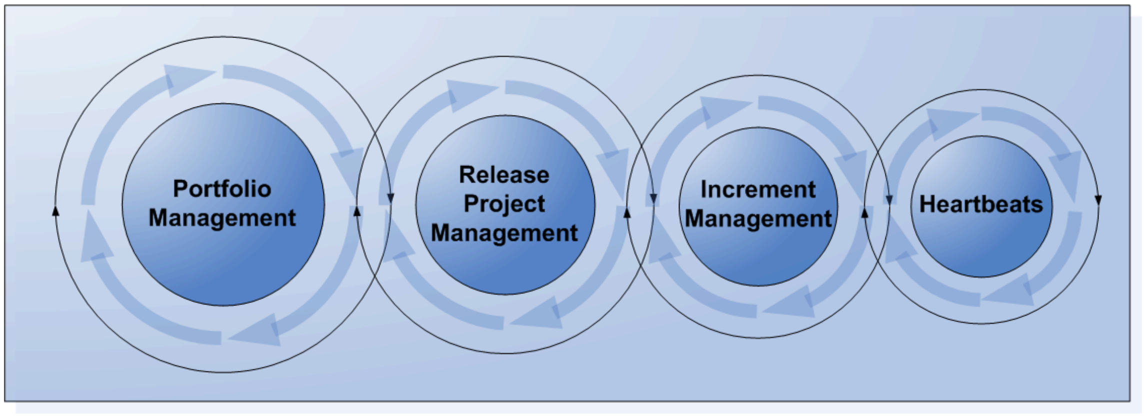


Figure 7. The Cycles of Control framework by SEMS

There is multi-site organization structure at Avain Technologies and the organization consists of four departments (Business, Development, Services and Administration). Improving of the communication between Business and Development was a high priority challenge for new process and emphasized by management. Thus, during process development the Cycles of Control at Avain Technologies was adapted as Table 11 on next page describes.

Table 11. The Cycles of Control at Avain Technologies

Name of cycle	Description
Strategic Planning Cycle	- Product Backlog and idea pool are used to express the product vision (Appendix 2: Scrum's Product Backlog)
Release Cycle	- can be either customer related or internal - contains planning & exploration, development and stabilization phases - the lengths of phases are planned at the beginning of the release program and embedded into sprints - Release Backlog for goals, themes and prioritized list of items (e.g. feature) for the release programs is created
Sprint	- as in Scrum Sprint (chapter 2.1.3)
Scrum Cycle	- as in daily Scrum (chapter 2.1.3) - daily Scrum can be also e.g. ½ weekly or weekly Scrum

The cycles of improved product development process adapted by Avain Technologies (cf. also Figure 9) are described in Figure 8:

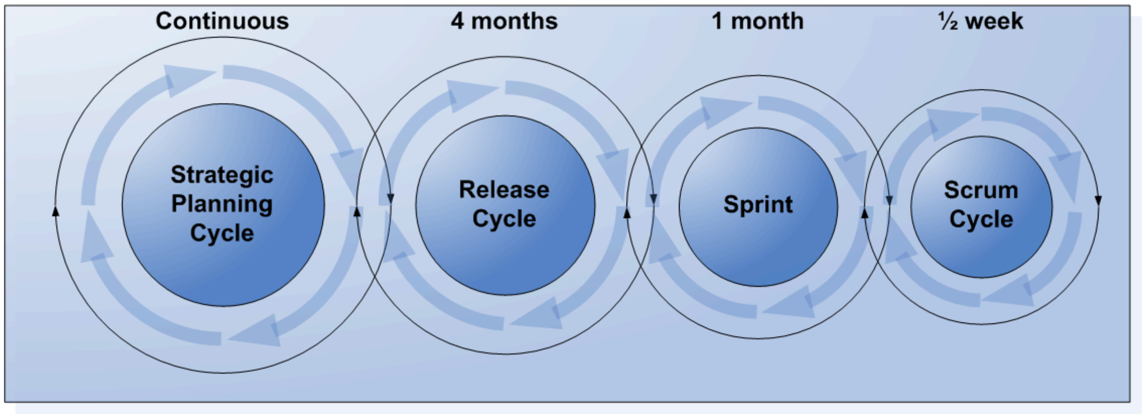


Figure 8. The cycles of improved process at Avain Technologies

Mainly XP@Scrum (cf. chapter 2.4.1) contained practices were combined into the framework and these practices were piloted during the cycles. Experiences that were noticed during piloting at Avain Technologies have been summarized in Table 12 on next page.

Table 12. Combination of agile methods related experiences according to piloting at Avain Technologies

Well-tried practices
- Automated testing with TDD practices - Backlogs: Product Backlog, Release Backlog and Sprint Backlog - Pair programming (mostly for hard tasks) - CVS - Coding standards - Meetings and reflection workshops - Communication model - Small tasks - Prototyping (for estimation the effort of large features)
Problems with proposed solutions
- Problem: The component team could not successfully synchronize their work with the product teams. - Solution: Organizational structure of the teams was changed. Product owner was nominated, customer relationship between designers was improved and designers were able to work better within their area of expertise.
- Problem: Testing environments and HW were not established during the first release program so system testing was difficult to start during the first release program. - Solution: TDD and hand-off documentation for e.g. system testing were used later.
- Problem: Common understanding of future release programs was unclear and types of the releases had not been described clearly. - Solution: A product roadmap was created. Content of release programs was described in the Release Backlog. Meaning of alpha, beta and commercial releases were described.
- Problem: The amount of new ideas generated for the product during the first release program was very low. - Solution: Continuous product planning and active participation in meetings. New ideas were stored to idea pool. The idea pool was in specified place where everyone in the corporation can see it.
- Problem: After first phases of (internal) release cycle, the release project remained in the development phase without reaching the stabilization phase. - Solution: Stabilization phase was expected in the later release programs.

(Itkonen et al. 2003)

Overall results regarding use of the adapted Cycles of Control framework seemed to be good in spite of few problems that had been noticed during this piloting. Use of the new framework increased understanding of the development process through the corporation. It helped the linkage between Business and Product Development. The framework also helped in identifying the crucial control points between last mentioned organizations and enabled defining well-functioning connections between them. The adapted development process seemed to be effective way to achieve many benefits such as advanced SW development practices.

The development process increased especially communication between Business and Development. Also communication model between the different departments of the corporation was defined. After improving of communication, Business knew the release dates and contents well in advance. Due to that, Business was able to allow release dates and contents for marketing so that sales could start in right time. Business also clearly saw what kind of possible changes were done regarding e.g. content of release programs. Besides improved communication also short duration of cycles helped to see possible changes early. Due to improved communication and short duration of cycles, the organization's risk of committing to unrealistic development plans had decreased significantly.

The development process also improved many current practices that were in use within SW development in the corporation. In addition, it was believed that the process improvement did not bring unnecessary bureaucracy and improvement had progressed faster with the framework than without it.

All designers agreed on using SW development practices. Especially automated testing with TDD practices, adapted pair programming mostly for hard tasks (otherwise co-operation between designers), CVS for version control and coding standards were agreed. XP practices such as simple design, collective ownership, continuous re-factoring and 40-hour week were not emphasized to designers as much as other practices. Above XP practices were however listed in the process definition.

Automated testing was one Q&A practice that had been used with TDD practices from the first release program. Designers had opinions that automated tests are useful and they felt that the existence of these tests had made them more confident when making changes to the code. The TDD practice had been more difficult to adapt and it was used only occasionally. On one hand the TDD was considered to be troublesome but on the other hand useful.

Different meetings were arranged for e.g. planning, evaluation and estimation of work. No Scrum daily meetings were needed, instead the Scrum meetings were held twice (on Mondays and Wednesdays) in week. Release Planning meetings were arranged at the beginning of release programs and sprint planning meetings were arranged at the beginning of sprints. Release planning meeting took one day. Whole team and few managers participated in that meeting and all features in the Product Backlog were briefly discussed. The features had been split and described in the Release Backlog. Each feature from the Release Backlog had generated several tasks to Sprint Backlog for each sprint. These tasks were estimated roughly by team members. Therefore, skills concerning estimation and lightweight estimation techniques increased in the team. Estimations regarding tasks were done in man-scrums, i.e. how much effort is required to complete each of them. Prototyping was used for estimation the effort of large features because it is not possible to accurately estimate the effort of large features. Instead it is possible to approximately estimate how much effort is needed to implement some prototype and after prototype how much effort is needed to implement real feature. The results also described that wrong estimations in the beginning are natural and it was suggested that it should be understood by Business too.

Sprint demonstration session was arranged at the end of each sprint and reflection meetings were suggested to arrange in the future. Designers considered the meetings a very good practice regarding e.g. learning. Additionally it was efficient way to find out what others were doing and what has been done. The demonstration session was good way to see and evaluate new increments of the product. On the other hand, there were problems getting managers from Business to participate in these meetings. Reasons were e.g. time management related problems and Business did not consider these meetings to be essential. However, significance of these meetings was emphasized to managers from Business and they participated in the last demonstration.

The backlogs seemed to be useful and beneficial practice for content management even though availability of the backlogs was suggested to be better. Sprint Backlog was used for managing content of sprints. “Red-Flag”-practice was used in the Sprint Backlog to manage unexpected work that was not related to achievement of the sprint goal. Constant amount of effort (40%) had been reserved for Red-Flags related work and non-development work. Product Backlog seemed to be suitable for managing content of future release programs and Release Backlog for managing content of current release programs.

(Itkonen et al. 2003)

2.4.3 Case: Smartner Information Systems

Smartner Information Systems Ltd. (Smartner) is a small software product corporation at mobile technology market. It offers mobile technology for operators and provides application services for enterprises that need tools for building mobile services and solutions. Smartner improved its product development process because more flexibility and control were needed for market needs. At the beginning, the traditional waterfall model was used at Smartner and later it had been changed to be iterative and incremental. More process development and few main requirements had been defined for process. The main requirements for the improved process were: schedule oriented (schedules should be met, adjustments done by dropping functionality), fast reaction to change (there must be a structured way to change plans often), customer oriented (customers participate in planning and testing of the products), managed requirements (long-term planning as well as short-term specifications), and extensive testing.

The cycles of improved product development process adapted by Smartner (cf. also Figure 8) are described in Figure 9:

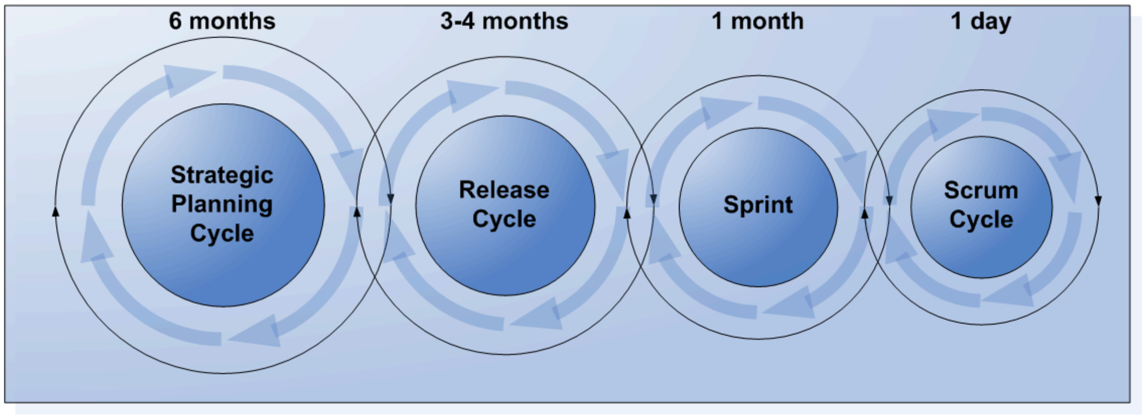


Figure 9. The cycles of improved process at Smartner

Adapted agile practices were piloted during the cycles. These practices had been chosen from four methods: RUP, XP, Scrum and Microsoft's "Synchronize and stabilize" methods. According to article, the data were collected from process-related documents which personnel had written and notes that had been made at the meetings. Also semi-structured interviews had been arranged. Experiences that were noticed during piloting at Smartner have been summarized in Table 13:

Table 13. Combination of agile methods related experiences according to piloting at Smartner

Well-tried practices
- Backlogs: Product Backlog, Release Backlog and Sprint Backlog

- Meetings - Pair programming (mostly for hard tasks) - CVS - Coding standards - Collective ownership - Simple design - Small release programs and short iterations - Automated daily builds with automated tests and test report
Problems with proposed solutions
- Problem: Outsourcing failed, mainly concerning quality of work and knowledge transfer between Smartner and the independent subcontractor. - Solution: Outsourcing was adapted to projects so that would be easier to control.
- Problem: Dependencies between the Sprint Backlog and order in which to perform the tasks were not written down anywhere. - Solution: Dependencies were written down.
- Problem: Bottom-up (by designers) approach seemed to succeed only partially because management was not fully committed to the new process. - Solution: Management processes were developed.

(Rautiainen et al. 2003)

The case study regarding Smartner’s experiences proved that practices from different agile methods can be successfully combined to make a coherent entity, as researchers told according to article. Also combining of flexibility and control succeeded when the SW development process was improved. Flexibility was achieved with short sprints, after which new decisions about project scope could be done when planning the following sprint and scope of sprints would be frozen until end of current sprint. Control was achieved through mapping sprints to release cycle control points, in which visibility of process increased. The visibility of SW development process increased also due to adapted agile practices e.g. in project planning. Due to improved process, designers had opportunity to work with assigned tasks. Certain main activities and the use of many practices created so called daily rhythm that was a key to more predictable development during sprints.

During improvement of the above mentioned process the management processes for roadmapping, release programs and sprint planning were described and deployed in detail. The product roadmap was used for upcoming release programs. Customers and partners participated in strategy release management board by providing and evaluating ideas for future release programs. The roadmap was updated after each release program and release programs were scheduled into it. Also high resource allocation could be planned based on Product Roadmap. All ideas for features were updated to Product Backlog that provides a systematic way to collect feature suggestions continuously and describes the main scope of the upcoming release programs. Approximate effort estimation was done to these features into preliminary Release Backlogs and the features were reprioritized. Reprioritizing was done by product team and helped refining the scope of release programs. When some new features with higher priority were added to the Release Backlog it meant that other features must be excluded.

Release cycle lasted three months after which a beta release program of a product was ready. It was divided into three parts: i) scheduling the sprints and refining the projects' scope, ii) decision-making at release cycle control points that were mapped to development process control points (sprint reviews) and iii) product release program. Features were built during approximately 30 days sprints. Smartner corporation had got similar results as Avain Technologies (chapter 2.4.2) regarding progress of sprints. Therefore, those results are not discussed here in detail. E.g. meetings, communication between different departments (e.g. Business and Development), Sprint Backlog, CVS and estimating efforts of tasks were used by similar way in both corporations.

The features were build incrementally with few main activities as typical SW work, synchronizing of effort and code (daily), demonstrations of implemented functionality (short demo weekly and sprint demo monthly) and adjusting (e.g.. updating the effort estimations) of the project. Simple design and keeping end-user in mind were two main principles in SW development. Also re-factoring, coding standards, collective ownership and certain way of using pair programming and automated daily builds were noticed to be essential parts of SW development. Pair programming was used mostly for hard tasks (i.e. problem solving) and competence sharing. A build was done automatically every night and code versions for the build were taken from version control system. The tests were run automatically against the new build. The test report was generated automatically and it was available the next morning.

Sprint ended with sprint review and sprint demonstration of what had been achieved during sprint. Sprint demonstrations helped the management team to follow the progress of development and make more accurate decisions. These demonstrations also made possible that customers had possibility for early access to the new product version. Therefore, they were able to influence the content of release program if needed.

(Rautiainen et al. 2003)

2.5 Summary of literature review

In consequence of this literature review, theoretical basis of agile SW development was created and answers to first research sub-question (chapter 1.3) were found. Those answers consist of basic principles for constructing the XFRC process model that enables faster release cycle. Overview to agile methods (chapter 2.1) and related experiences of other corporations (chapters 2.2 - 2.4) brought along such principles. Mostly XP and Scrum methods were discussed.

The experiences of other corporations clarified which agile practices had been proven to be beneficial according to piloting of one or more agile method and which not. Firstly, the experiences clarified what kind of well-tried practices concerning use of agile method(s) have been noticed during piloting agile method(s) in corporations. Secondly, the experiences clarified agile method(s) related problems and most of experiences also suggested possible solutions for solving those problems. Experiences consist of three XP piloting cases, three Scrum related cases, one XP@Scrum case and two cases concerning adapted Cycles of Control framework.

XP focuses mainly on engineering practices for agile SW development. Especially collective ownership, on-site customer, pair programming for mostly hard tasks, Planning Game, simple design, small release programs and TDD had been proven to be beneficial practices of XP. Also e.g. code reviews, coding standards, on-site coach, re-factoring, sitting arrangements, Spike-time for e.g. reading and learning about various issues, and "Zero Feature Iteration" practices were discussed in XP cases.

Scrum focuses mainly on managerial and organizational aspects regarding agile SW Development. Scrum seemed to make SW development process visible, controllable and manageable. Most beneficial practices of Scrum seemed to be freezing development scope for a month (sprint), backlogs, Use Cases and adaptable small teams.

Likewise XP@Scrum and adapted Cycles of Control framework seemed to be way to achieve many benefits for SW development. Both of them consist of primarily above practices. Moreover, few other well-tried practices were discussed: communication model, reflection workshops for increasing team effectiveness, idea pool for continuous planning of product, prototyping for large features, and Red-Flag for unexpected work. Automated and continuous testing (TDD included), continuous integration with e.g. daily builds, incremental and iterative way to develop SW, small release programs, small tasks, and various meetings seemed to be common denominator to almost all cases that were discussed in this literature review.

3. IMPLEMENTATION AND RESULTS FROM INTERVIEW STUDY

All truths are easy to understand once they are discovered; the point is to discover them.
Galileo Galilei

This interview study was needed in order that answers to second and third research sub-questions (chapter 1.3) can be found. At first this chapter introduces overview to the interview method (chapter 3.1) and implementation of interviews within this research. Thereafter the summarized interview results are described in chapters 3.2 and 3.3, and experiences of Scrum piloting are discussed in chapter 3.4. The interview results are answers to asked interview questions (Appendix 9: Interview Questions). These results describe current state of Platform R&D and Application R&D organizations. Mainly problems with proposed solutions (chapters 3.3.1 - 3.3.11) by interviewees are discussed. Also well-tried practices (chapters 3.2.1 and 3.2.2) were collected from both organizations during interviews. The results are based on opinions and experiences of individual interviewees. If e.g. many, few or only one interviewee has told something it has been mentioned singly in interview results. Direct quotations are not used to describe opinions and experiences but interview results are described as normal text. Proposed solutions by interviewees have been compared with the literature. Comparison related references are described by chapter numbers if particular article etc. has already been referenced in the literature review. Otherwise, current reference system is used.

Comparison of interview results with the literature follows mainly following sequence:

1. Description of the problem
2. Description how the problem correlates with research problem
3. Proposed solution(s) by interviewees
4. Comparison with the literature
5. Additional proposed solutions according to the literature

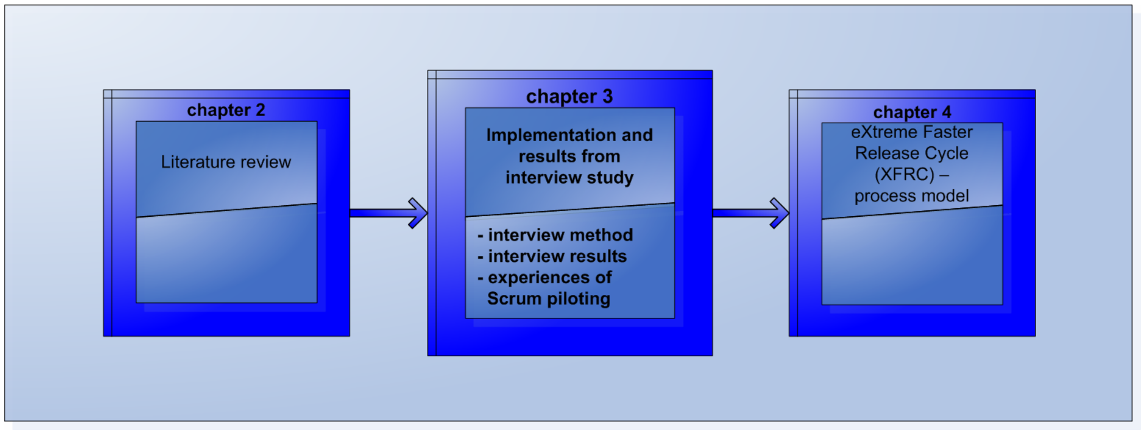

chapter 2
Literature review
chapter 3
Implementation and results from interview study
- interview method
- interview results
- experiences of Scrum piloting
chapter 4
eXtreme Faster Release Cycle (XFRC) – process model

Figure 10. Introduction to chapter 3

3.1 Overview to the interview method

3.1.1 Motivation

Interview method is one way to collect data during research. Interview method was selected because it is multipurpose method. Discussion is typically faster than e.g. writing and additional questions are easier to make than after e.g. mail questionnaires. Assumption was that response rate could be higher than e.g. in the questionnaires. The assumption came true and almost all interviewees came to interview sessions. On the other hand, it would have been possible to send questionnaires to e.g. hundreds of participants through large SW development organizations, but in this work collection of research data was accomplished by interviews from tens of interviewees as planned.

In addition, interview method has also typically been used during similar researches that concern software process improvement in large organization. Interview method has been used to collect e.g. experiences that were discussed in chapters 2.2 - 2.4. It has also been used during many other researches such as research by Aho (2006). In particular research also agile methods were discussed and data was collected with interviews. Basically the interview method is well-known method and it has been used during many researches. In consequence of above explanations, the interview method seemed to be suitable also for this research.

3.1.2 Interview flow

The interviews were conducted in Espoo in Finland. Exact sampling was 24 people from Platform R&D organization and 27 people from Application R&D organization: 20 SW Engineers, 3 SW architects, 4 line managers, 7 project managers, 4 program managers, 8 testing personnel and 5 Release Content Owners. Interviewees were contacted by a mail invitation. An average of three people participated in interviews that were arranged in different times during autumn 2005. Every interview session took about 1.5 hours.

Open-end questions were planned before interviews so the interviews can be called structured interviews. At first questions concerned the background of interviewees and next questions were research problem related. Research problem related questions gave rise to lot of discussion about CR related problems. Also a lot of proposed solutions to problems were collected during interviews. The additional questions were also reserved for discussion after research problem related questions. This interview method was followed in all interviews and questions were almost same for every interviewee from different competence areas and roles. The interview questions are described in Appendix 9: Interview Questions.

3.2 Interview results and comparison with the literature – Well-Tried practices

3.2.1 Platform R&D aspect

During the interviews of personnel from Platform R&D the well-tried practices were collected. These well-tried practices have been summarized in Table 14:

Table 14. Well-tried practices by interviewees – Platform R&D aspect

Well-tried practices by interviewees
- (Re)moving of some feature seems to be quite easy
- Independent SW groups and independent FT group
- Communication has been succeeded in multi-site organization if easy things are handled
- Workshops
- Split tasks
- Simulation Game
- Project managers managing only one project at a time
- Use of Scrum in SW development (chapter 3.4)

(Re)moving of some feature from release program does not seem to be so big problem as adding of some feature to ongoing release program. According to many interviewees, although some feature would be (re)moved, SW work could continue if projects would give enough time to continue. SW work is done in independent SW groups and functional testing is done by independent FT group. Few interviewees told that independent SW groups and independent FT group seemed to be good solution albeit co-operation between testing and SW groups could be improved. Co-operation between above groups is described in detail in chapter 3.3.5.

MT documentation has been good to write while implementation is done, although the documentation is (case by case) heavy to read. In addition, a few interviewees saw that maybe it is possible to write MT documents also before implementation (cf. TDD related experiences in chapters 2.2.2 - 2.2.3 and 2.4.1 - 2.4.2). These opinions proved that there is willingness to use TDD in the future.

According to many interviewees, communication has been succeeded in multi-site organization if easy things are handled. On the contrary, language barrier and handling of complex things have caused communication problems. When people in multi-site organization familiarize with each other, communication and working is easier. Sometimes project managers visit in other country and vice versa. Also virtual meeting tool with talking connection via conference phone has helped communication between international organizations. Further-more, general opinion was that workshops have been good solution for e.g. competence shar-ing and increasing communication.

It has been easy to split e.g. feature related tasks to many persons. Thus, tasks can be com-pleted faster compared with tasks without splitting. Few interviewees told about Simulation Game that was used for supporting SW work mainly in SD phase. Use of Simulation Game was succeeded and it was great opportunity to learn and share competence regarding some feature or part of it. In this case the Simulation Game means that some feature or part of it is simulated during particular session. E.g. few designers and SW architect(s) participate in the session. There can be used e.g. some Use Case based sequence of some feature to illustrate what kind of functionality should be in practice. Deviating from above case, in some other cases the Simulation Game is suitable also e.g. for change management when different proc-esses are improved (Taskinen 2002, pp. 70-83).

During discussion about project management arrangements, general opinion was that it has been good solution that one project manager manages only one project. On the other hand, one designer suggested that full-time line managers are not necessarily needed. Instead de-signer suggested that line managers could also manage some project and they could also partly work as specialist.

3.2.2 Application R&D aspect

Besides Platform R&D aspect the well-tried practices were collected also from Application R&D. These well-tried practices by interviewees from Application R&D have been summa-rized in Table 15:

Table 15. Well-tried practices by interviewees – Application R&D aspect

Well-tried practices by interviewees
- Split tasks - Workshops - One Release Content Owner per release program - Small groups - Iterations from IS phase to MT phase - Short code review sessions for minor changes of code - Formal (official) reviews for major changes of documentation and code

- Modularization of code
- CVS

Split of tasks seemed to be beneficial practice in Application R&D as well in Platform R&D. Many interviewees told that it is easy to split e.g. feature related tasks to many persons. Thus, tasks can be completed faster compared with tasks without splitting. According to many interviewees, also workshops seemed to be well practices as in Platform R&D.

Furthermore, organizing of Release Content Owners' work and structure of SW groups seemed to be succeeded in Application R&D. One Release Content Owner per release program has been good solution because it does not cause too much parallel tasks to Release Content Owners. Small SW groups have been good organizational arrangements according to few designers and few line managers.

Many designers told that iterations from IS to MT have been beneficial practice. According to many interviewees, formal code reviews for minor changes have not been arranged. Instead short code review session in which typically two (or three) designers check that code correspond to original plans has been arranged. In addition, they felt that formal review is typically needed only for major changes of documentation and code. They also suggested that if minor changes are done e.g. during four iterations, unofficial review could be enough and formal review could be arranged after iterations or until next feature or major changes are done.

Also e.g. modularization of code was discussed during interviews. One opinion was that there is somewhat improving of modularization of code ongoing in Application R&D. In addition, about version control of code was discussed. According to many interviewees, CVS has been good solution for above version control.

3.3 Interview results and comparison with literature – problems and proposed solutions by interviewees

3.3.1 Lot of parallel tasks in daily work

According to the interviews, there seems to be too many parallel tasks in designers' daily work. It seems to be the biggest problem in many groups in Platform R&D and Application R&D organizations. E.g. implementation of new feature(s), CR work, SW work to parallel projects and related meetings, maintenance, support to other group(s) and general tasks cause too much parallel tasks in designers' daily work. Many designers can work in even three projects simultaneously. These problems are in agreement with the results that were discussed in chapters 2.2.1 - 2.2.2 and 2.3.1.

From research problem perspective parallel tasks slow down release cycle because of strict schedules (chapter 3.3.4), re-prioritization of tasks (chapter 3.3.8), and moving from one task to another during workday. According to many designers, when some higher prioritized task should be done it causes interruption to ongoing task and the ongoing task is delayed. For example, designer is doing CR work for some new feature and only minor changes would remain to complete that task. Suddenly comes very high prioritized fault correction that must be done immediately and there are no other resources available for the job. In consequence of that, ongoing CR work is interrupted and investigation for solving of fault is started. Some task is done e.g. in one hour and other task is done e.g. in couple of hours during a workday, which causes a lot of task switching and additional overhead.

Many designers wish they could concentrate on only one task during day and moving from a task to another should not happen during day. They also hope better organizing of above things and general suggestion was that dedicated persons to each task could be good solution. According to many designers, there could be dedicated responsible person(s) for SW work, CR work, maintenance, testing support and other possible responsibilities. For example, when fault correction should be done, one person inside group could prioritize and do all fault corrections without interruptions to other designers' work. This kind of responsibility sharing requires also competence and according to many interviewees it should be shared more through organizations (chapter 3.3.2). On the other hand, they also clarified that if all members of team are not experienced enough and competence should be increased it takes time and it slows down e.g. release cycle. However, they assumed that on the long run their competence could increase and release cycle could become faster.

The results regarding suggestion for dedicated persons are in accordance with Scrum practices. In Scrum method (chapter 2.1.3) team decides during sprints what it will do and dedicated responsibilities in team are encouraged (Schwaber 2003, pp. 6-7, 101-118, 142-143; Schwaber & Beedle 2002, pp. 8-9, 28, 31-32, 36, 38, 47-48, 71-72, 118, 140-141, 147-154). Similar results were also discussed in chapters 2.3.1 and 2.4.3. Last-mentioned chapter describes also that freezing development scope for e.g. one month sprint has proven to be one solution to decrease parallel tasks. When the development scope is frozen for e.g. one month at a time it helps in giving the team a chance to work on their assigned tasks. It seems to be the fact if team members do not participate in many parallel projects during these sprints.

In XP, team members implement their tasks that they want to be responsible for during iteration. It is possible due to team‘s possibility to make decisions (chapter 2.1.2) regarding programming and collective ownership (chapters 2.2.1, 2.2.3 and 2.4.3) that needs competence in team as dedicated persons (Ambler 2002, pp. 192-196, 214-222). If there are many responsibilities in team so meetings can be arranged before iterations and coach can keep track of what the team members are doing (chapters 2.2.1 - 2.2.3 and 2.4.1). Thus, knowledge of what group members are doing, supervision and synchronization of tasks could succeed and delays in ongoing tasks would decrease.

In addition, Mobile-D approach has been developed by Technical Research Centre of Finland (VTT) and University of Oulu in co-operation with companies developing mobile SW products and services (Abrahamsson et al. 2004). Adapted Mobile-D could be one alternative to decrease parallel tasks in daily work because it recommends that iterations could contain few different types of development days. According to Mobile-D approach, such days would be *Planning Day*, *Working Day*, *Release Day* and *Integration Day* (Abrahamsson et al. 2004). Last mentioned day would be needed if multiple teams are developing same product (Abrahamsson et al. 2004). From research problem perspective, parts of Mobile-D approach could be adapted to be as dynamically scalable periods without parallel tasks. Duration of periods would depend on how long time some split task has been estimated to take time. If duration has been estimated to take time e.g. over one day a task can continue on next day(s) without overtime work. Basically duration of some task is not frequently just e.g. one day. On the other hand, it is possible to estimate duration to be approximately e.g. one day.

3.3.2 Lack of competence sharing

According to the interviews, competence should be increased through R&D organizations so that designers could work with different but not parallel tasks (chapter 3.3.1). Also needed resources (chapter 3.3.4) could be available for any phase of SW process. Therefore, from research problem perspective lack of competence sharing slows down release cycle.

A proposed solution according to interviewees was that job rotation inside group could be done e.g. couple of time in year. Thus, competence could be increased inside group. Furthermore, when talking about the competence inside the group, risk seemed to be that only one person knows about some program block because in many cases only one person is responsible for some program block. However, job rotation could minimize that risk.

According to couple of line managers, job rotation could be extended also to other groups. Designer would be in same group, but task(s) could also come from some other group. Due to extended job rotation, learning about other competencies (also e.g. marketing, product management, testing etc.) would be possible. In that case, core competence should be good (e.g. specification work is difficult if core competence is not good).

Similar results were described in chapters 2.1.2, 2.2.1 - 2.2.3 and 2.4.1 - 2.4.3. In addition, similar results are also reported elsewhere in the literature (Aarnio 2001, p. 23; Ambler 2002). These results are similar because above literature discusses about e.g. pair programming and pair coaching. Both of them can be considered to partially be as some kind of job rotation. Pair programming increases competence especially in situation when pair is changed occasionally. Furthermore, it seems to be good alternative to have experienced designer and novice designer working as a pair so novice’s knowledge can increase regarding e.g. programming. This is called pair coaching (chapter 2.2.2). Due to use of pair coaching, designer(s) could learn e.g. specification work (cf. job rotation inside team). When pair would be changed, specification work related competence could be increased through team if e.g. SW Architect and some designer are working as pair in turn. Furthermore, when designer and e.g. FT personnel are working as pair, so designer could learn FT and execute FT cases if needed resources would not be available.

According to FT personnel, generally one people tests one CR related things for one release program. Their opinion was that they could learn and test it also for other release programs. According to Application R&D ST personnel, it could be good solution if Use Cases could be used for planning of ST cases. Thus, it would be possible to see “how customers really use some feature”. These results are congruent with the results that were discussed in chapter 2.3.3. In addition, RUP practices (Hirsch 2005) suggest that Use Cases are suitable in different planning phases. Therefore, Use Cases could also be used for clarifying how some CR related tests should be executed within different release programs.

The literature describes that the test cases could be planned and written as TDD suggests because it has proven to be quite beneficial practice according to experiences of other corporations (chapters 2.4.1 - 2.4.2). Use of TDD probably could increase competence sharing if e.g. pair programming or coaching would be adapted to testing phases. Testing could also become more effective because planning of tests would be started early. The TDD practice is discussed more in chapter 3.3.5 and partially in chapter 3.3.10.

According to experiences of other corporations, other suitable practices for increasing of competence seem to be: meetings & workshops (chapters 2.2 - 2.4), prototyping for large features (chapter 2.4.2), Planning Game (chapters 2.2.1 - 2.2.3), training sessions (chapters 2.2.2 - 2.2.3), Zero Feature Iteration (chapter 2.2.3) and coding standards (chapters 2.2.2 - 2.2.3. Furthermore, the central process elements in organizational learning, knowledge management and teamwork have been summarized in the literature (Taskinen 2002, p. 68). Due to all above agile practices competence could be increased as knowledge increases. After achieving of enough competence regarding different tasks team members could work with any phase of SW process.

3.3.3 Bureaucratic decision making

According to many interviewees, decision making related to CRs is sometimes slow regarding approval/rejection of CR. Due to this SW architects can not start specification work in good time. From research problem perspective it causes that start of design and implementation work by designers is delayed too. One alternative according to interviewees was that decision making could be done by low-level management, i.e. by line managers. Therefore, decision making related bureaucracy could be decreased. On the other hand, decision making concerning CR requires that overall effects on a product are known. Line managers typically know these effects concerning part of product that is developed in particular SW group. However, clarification of overall effects is possible when line managers communicate with relevant people and those people would share competence in order that above effects can be found.

Similar results concerning CR related decision making done by low-level management was not very found from the literature. It is possibly because of corporations' internal bureaucracy why that is not necessarily published. However, similar results were described in chapter 2.4.1 concerning how to achieve CMM level 2 and ISO9001:2000 by taking e.g. minimum bureaucracy into consideration. In addition, details regarding decision making are described from organizational aspect in the literature (e.g. Huczynski & Buchanan 2001, pp. 737-767). Team’s possibility to make decisions regarding e.g. programming was discussed in chapter

3.3.1. Furthermore, bureaucratic reviews were discussed during interviews and these reviews concern bureaucratic decision making. This aspect is discussed from research problem perspective more in chapter 3.3.10. It is possible that more similar results regarding bureaucratic decision making can be available e.g. in some other conference articles. In the conference articles, that were included in this research, bureaucratic decision making was not discussed much.

3.3.4 Underestimation of needed resources

General opinion was that there are too much CRs in both R&D organizations. Furthermore, efforts of CR related work are too optimistic although it is known that CRs are coming continuously. Many designers proposed that at least 20% of total effort of release programs should be reserved for CR work and ST personnel from Application R&D told that they need more time (at least 20% per one person who could share competence (cf. chapter 3.3.2) inside group) for reading of specifications. They suggested that time could be allocated from end of ongoing release program or from the beginning of next release program.

Above results are corresponding with Spike-time (chapter 2.2.3) and Red-Flag practices (chapter 2.4.2). Spike-time means that time is reserved for e.g. reading and learning about various issues. Red-Flag practice means that unexpected work is described e.g. in Sprint Backlog to show how much effort should be reserved for unexpected work.

Also resource allocation was discussed during interviews. Resource allocation should be continuous, according to few managers. In addition, milestones should be noticed in better way when resource allocations are done. Some projects have been started too early although other projects are ongoing. Briefly, resources should be in right place where resources are really needed as one line manager told during interview. According to many managers, “best” resources are typically needed in all projects. Unfortunately, when “best” resources are working with some other high prioritized task they can not participate in e.g. CR work immediately if some changes should be done to content of release program in late phase. Late changes to the content of release programs are discussed more in chapter 3.3.8. Due to above problems, start of CR work is delayed and therefore also release cycle slows down because a lot of resources are needed for CR work (cf. also chapter 3.3.1) and those resources are always not available in time.

Resource allocations related results are similar with the results reported in the literature (Stapleton 1997; Aho 2006). These allocations were also discussed in chapters 2.2.2 and 2.4.3.

According to Stapleton (1997) time and resources can be fixed so that scope of release content (cf. chapter 3.3.8) can be adjusted. This aspect seems to be good solution at least from fixed time perspective which is discussed more in chapters 3.3.8 and 3.3.9. Fixed time perspective is also discussed along with experiences (chapters 2.3 and 2.4) regarding mostly one month fixed time sprints. Besides fixed time perspective dynamic length iterations are discussed in chapters 3.3.8 and 3.3.9.

Grossman's opinion (chapter 2.2.2) was that designer resources should be increased. From Platform R&D and Application R&D perspective, increasing of (human) resources is not topical action in this time because for the time the being, amount of personnel is tried to keep somewhat current in above R&D organizations. Therefore, those results have not been adapted within this case. However resources could be moved if resources are needed for some specified task in somewhere.

Also prioritization of CRs was discussed during interviews and it seems to cause quite much work. According to many interviewees, participants from Business Area and Product Management from both Platform R&D and Application R&D should participate in prioritizing of CRs. Nowadays only Product Management makes prioritization and information is sent to Release Content Owners.

Similar results are reported in published Master's Thesis (Aho 2006) and discussed in chapters 2.2.2, 2.3.1 - 2.3.2 and 2.4.2 - 2.4.3. Aho researched suitability of agile methods from SW process improvement perspective in one R&D organization at Nokia. According to Aho, prioritizing of the most important requirements and features should be done carefully when the project starts (Aho 2006, p. 62). Having a plan also for CRs enables implementing of those changes during e.g. next iteration (Aho 2006, p. 23).

Many interviewees told that effort estimations and above mentioned prioritization of CRs take a lot of time. Due to that, release programs delay because some high prioritized task can interrupt some other task and similar possible risks of upcoming tasks are not necessarily noticed when planning of estimations. It is a risk that should be noticed when estimations are done.

These results concerning estimations are in accordance with the literature Aho (2006). Risks of the project usually become true in the late phase of the projects and the planning should be done so that the most difficult and risky tasks are done in the earliest possible phase moment. Effort estimations should not be even asked without giving enough time to prepare and the estimations should be based on the best knowledge about the effort, not just a guess. Also the challenges and risks related to the tasks should be noticed when estimation planning is done.

It is also good solution to give calendar time estimate with the working hour estimate. In addition, the management team should concentrate on project monitoring, guidance and resources while the development team should make the development work related decisions. (Aho 2006, p. 46, 62)

Also Portfolio Management is suitable to provide e.g. guidelines how estimations should be organized in terms of SW development and release cycle (chapter 2.4.2). In addition, Product Roadmap and Release Backlog for upcoming release programs seem to be suitable practices from estimation planning perspective. According to experiences of other corporations (chapters 2.2 - 2.4), also e.g. collective ownership, demo-sessions, prototyping, meetings and small release programs seem to be suitable practices. Additionally, due to previous sprints related experiences the estimates become better after each sprint (chapter 2.3.1).

3.3.5 Inadequate co-operation between testing and SW groups

In 2005 there was quite large SW groups in Application R&D. In consequence of organizational change it was decided to split SW work and MT work to different groups in one Competence Area in Application R&D. At the same it was seen that size of SW groups was too big so size of SW groups was optimized to consider only three designers and their line manager. MT group consists of less than five MT personnel and their line manager. Therefore, there is own group for module testing and small SW groups in above Competence Area.

Line managers told that it would be better that module testing would be executed by SW groups so that supervision would be easier. On the other hand, supervision is done by line manager of MT personnel in MT group. Or module testing could be executed at least partially by SW groups, because designers execute at least a bit of MT in any case. MT group could execute regression and fault testing. Also opposite opinion were expressed by designers and MT personnel. Their opinion was that MT group seems to be good solution.

E.g. group formation (Huczynski & Buchanan 2001, pp. 275-308) and group (team) structure (Huczynski & Buchanan 2001, pp. 309-343) are part of organization structures that are discussed more in the literature (Huczynski & Buchanan 2001, pp. 409-554). Group formation and team structure seem to also be part of agile methods related practices that are recommended for small groups. Therefore, the results regarding size of groups are in accordance with the literature that was discussed in chapter 2 because optimal size of groups seems to be under 10 people from agile method’s perspective and related experiences of other corporations (chapters 2.2 - 2.4). According to those results, small groups could be suitable also for

Platform R&D and Application R&D organizations. In addition, agile practices encourage also to communication and fast feedback that seem to be succeeded by e.g. pair programming, according to mainly XP related experiences (chapters 2.2 and 2.4). Pair programming could be applicable also for e.g. FT phase when it could be some kind of pair testing.

Also testing support was discussed during interviews. From FT perspective, testing support is needed from designers. Designers could participate in testing support and FT personnel should ask if they need some information in test planning phase. According to some FT personnel and managers, FT and SW work should be done in same place. They also suggested that SW work and FT work could be combined in same projects so that communication would be better and support feedback could be fast. FT personnel also told that it would be good if they would sit in same floor with designers, but FT personnel would not belong to SW group because in that case they should test only one specified part of some feature and it did not seem to be good solution according to their opinions.

From ST perspective, fault corrections related feedback should be faster from designers and other testing personnel to ST personnel. From research problem perspective above means that needed feedback through SW and testing groups should be faster so that it would not cause extra delay so much. Thus, co-operation could be improved through SW and testing (MT, FT and ST) groups.

Above results are in accordance with the results that were discussed in chapters 2.2.1 and 2.2.3. In those chapters were discussed among other sitting arrangements that were proven to be beneficial practice concerning co-operation. Consequently, different sitting arrangements could increase co-operation also in Platform R&D and Application R&D organizations. Also e.g. meetings & workshops (chapters 2.2 - 2.4) increase co-operation.

When above mentioned arrangements have been done the organizing of testing would be beneficial too. Use of TDD practice could be good solution to plan test cases in effective way as in chapter 3.3.2 was already discussed. According to experiences of other corporations (chapters 2.2.2 - 2.2.3, 2.3.3 and 2.4.1 - 2.4.2) and the literature (Maximilien & Williams 2003), TDD practice seems to really be suitable for planning of almost any (e.g. MT, FT and ST) test case. Maximilien & Williams (2003) have researched TDD in multi-site organization at IBM. In that organization the team consisted of nine engineers and management resources were located in two country. According to Maximilien & Williams (2003), TDD reduced defect rate after FT phase, functioned as the basis for quality checks, increased quality of product, and increased reuse of automated test cases. Additionally TDD perceived as extendable asset that will continue to improve quality over lifetime of the software system.

On the other hand, according to some experiences TDD has been succeeded only partially but training sessions have been arranged for solving of that problem (chapters 2.2.2 and 2.2.3). According to related experiences (chapters 2.3.3 and 2.4.2) and the literature (Cockburn 2004; Maximilien & Williams 2003), when test cases are planned and implemented so these test cases would be worthwhile to be more automated. In addition, in chapter 2.4.3 it was discussed that SW build could be generated automatically every night and after execution of automatically tests (e.g. so called basic test set) a test report could be generated automatically. The test report would be available the next morning as was described in chapter 2.4.3. If testing would be more automated it would probably be quite fast to execute test cases during e.g. regression testing or acceptance testing. Unfortunately, test cases for e.g. MT, FT and ST are only partially automated, and it takes time to automate them more as many designers told during interviews.

3.3.6 Lacking Feature Owner

According to many interviewees, a Feature Owner would be good solution because it takes time if right responsible person of feature is not found immediately. From research problem perspective it delays release cycle in consequence of delayed CR work. The Feature Owner could own and coordinate a feature through its lifecycle, also during maintenance phase. According to a Release Content Owner’s opinion, ownership was good practice earlier but nowadays there are no owners.

Experiences or other research results with relation to Feature Owner were not found from the literature. However e.g. Product Owner that is described within Scrum roles (chapter 2.1.3), seems to be similar than Feature Owner. On the other hand, Feature Owner(s) could also be other person than Product Owner if knowledge of some feature is enough (i.e. competent resources are available) from the perspective of this role. If competent resources would not be available at least competence sharing (chapter 3.3.2) and improved communication (chapter 3.3.7) are needed. In this case some person of e.g. some SW group could participate in learning of needed details regarding some feature at least in group level. If competence of some person in group level would be enough that person could take responsibility of “Chief” Feature Owner tasks and communicate with other Feature Owners working in group level in other groups.

3.3.7 Inadequate communication

General opinion was that communication cause a lot of problems between Platform R&D and Application R&D, and between multi-site organizations (e.g. between Platform R&D in Finland and Platform R&D in other country). In consequence of communication related problems through above organizations, it has negative effect on release cycle time. Solutions proposed by interviewees were that specified tasks would be done in same building (at least in the same country) and there would not be split tasks (e.g. IS work) through multi-site. These results are similar with related results that were discussed in chapters 2.2.1 - 2.2.3. Related results describe utilities of pair programming practice and “designers in same room” from communication perspective. In above cases, communication is easier and faster when distance between communicating persons is not too big. More details regarding communication from organizational aspect are described in the literature (Huczynski & Buchanan 2001, pp. 177-210).

Besides above general problem also other problems concerning inadequate communication were discussed during interviews. Many designers’ opinion was that line and project managers ask same things from designers. They suggested that it would better that only one manager, e.g. project manager makes questions to designer, so that same questions are not asked by both line and project management. From research problem perspective it is faster to respond to some same question once than twice. These results are corresponding communication model related results that were discussed in chapter 2.4.2.

Also participation in release program meetings should be organized better according to few designers. They suggested that participation in release program meetings could be better, i.e. it could be better that one project manager would participate in two meetings instead of that many designers would participate in many meetings. Also customers could participate in meetings according to some project and program managers. During meetings they could discuss about content of release programs. Nowadays only project managers, program managers, Release Content Owners and SW architect of release program communicate with each other about content of release program and mostly Product Management and Marketing Management communicate with customers. Product Management communicates with Program Management who communicates with Project Management and Release Content Owners and so on. These results are in accordance with customer involvement (Baskerville et al. 2003) such as on-site customer related results and also meetings & workshops related results that were discussed in chapters 2.2 - 2.4. On the other hand, if meetings etc. would be arranged too often not enough time would necessarily remain for real work, albeit meetings are part of actual work too. However, best benefits of meetings could be achieved if persons who have knowledge about the subject of the meeting would be invited and they would participate in those meetings. Above invitation and participation processes should be noticed concerning e.g. reviews which are discussed in chapter 3.3.10.

Furthermore, visibility of coming CRs should be better from perspective of designers. Slow visibility of CRs cause that designers do not know in time what kind of changes are coming, testing personnel do not know what should be tested and SW architects can not start specification work in time. Also multi-site aspect causes delay if some feature related task or part of it is done in other country. According to many designers, coming CRs should be informed faster to designers, i.e. mail information should be made in time or visibility of CRs should be available in intranet in time. In addition, couple of managers proposed that communication could be increased also between different management levels and customers. These results are congruent with backlogs (chapters 2.3 and 2.4) and communication model (chapter 2.4.2) related results. Also e.g. meetings & workshops (chapters 2.2 - 2.4) could increase visibility of coming CRs.

3.3.8 Late changes to the content of release programs

Content of release programs seemed to be too large according to many interviewees. From project and program management perspective, every big change (CR) to content causes lot of re-planning regarding scheduling and causes delay to release cycle. These results are in agreement with the agile practices which recommend small release programs and short iterations (chapter 2.1).

According to many interviewees, only fault corrections could be taken into release program in late phase, but not via CRs if amount of work is under 50 hours because bureaucracy takes extra time for every CR. According to a project manager’s opinion, CRs could also be combined so bureaucracy could be decreased. If CRs are taken into release program in late phase it causes a lot of work and might interrupt other tasks of designers because CR might be of higher priority than some other task. In addition, CRs cause lot of changes to content of release programs and content of release program should not be changed in late phase, i.e. no major changes after E1 milestone. Project and program managers told that CRs in late phase causes delays in ongoing release programs and as a consequence next release programs are delayed too. General opinion was that (re)moving of feature is minor problem than adding of new feature in late phase, albeit also (re)moving causes re-planning, planning of compatibility and testing. Other general opinion was that removing of feature is frustrating from perspective of designers. On the other hand, removing of feature is infrequent. Moving of feature to next release program seemed to be more general according to designers.

These results deviated from results discussed in the literature review because according to interviewees, freezing content should be done already in E1 phase and the literature proposed freezing content for only e.g. one month (chapters 2.3 and 2.4) sprint. Sprint can be considered to be as fixed time whereupon SW is developed iteratively and incrementally (chapter 2.1.3). According to the results described in chapter 2.4.2, it was perceived that Product Roadmap is useful for specifying of all planned product release programs. Release Backlog seemed to be useful for specifying of content of release programs. In chapter 2.4.3 it was discussed that e.g. freezing the development scope for a month at a time helps in giving the team a chance to work on their assigned tasks and creates a more relaxed atmosphere. All ideas for features were updated to Product Backlog that provides a systematic way to collect those ideas continuously. New features were added to the Release Backlog, meaning that other features can be excluded due to possible re-prioritization. The Product Roadmap was used for upcoming release programs and release programs were scheduled into that roadmap. The roadmap was updated after each release program. High Level resource allocation (cf. chapter 3.3.4) can be planned based on Product Roadmap according to results that were discussed in chapter 2.4.3. In addition, when (on-site) customers (chapters 2.2.1 - 2.2.3 and 2.4.1) had possibility for early access to the new product version so influencing the content of release program was possible.

From fixed time perspective (Aho 2006; Stapleton 1997), if some high prioritized feature would be added to some release program it could not be taken until next Sprint (chapter 2.1.3) is started if competent resources are available (cf. chapter 3.3.2). In this case, other designers could continue their own tasks and possibly a few of them could start working with above mentioned feature related tasks if their other tasks have not been prioritized with higher priority. At the end the codes from development branch(es) would merge to main branch. Merging could be done with CVS (chapters 2.2.3 and 2.4.2 - 2.4.3) as many designers from Application R&D told during interviews.

On the other hand, lengths of iterations do not necessarily need to be fixed. Instead length of iterations could be adjusted dynamically depending on planned work amount for iteration. Therefore, length of iterations could be adjusted to be from e.g. few days to few weeks. If planned work amount concerning the iteration is ready e.g. after few weeks a next iteration could be started after planning and prioritizing assignments.

3.3.9 Inconsistent synchronization of release programs

The interview results addressed that the release programs of the R&D organizations should be synchronized. It means that because of e.g. long development time of HW components Platform release programs content is frozen and release planning is made before application release program has defined its content. E.g. IS can be reached in Platform R&D although content of Application release program would not have been defined yet. It causes delay to release cycle because it is probably that content of Application R&D release program(s) is changed later and it causes changes also to Platform R&D release program(s). Above causes that generally requirements come from Application R&D to Platform R&D in quite late phase of release program (cf. chapter 3.3.8).

According to few designers, one solution could be that at least module tested Platform SW would be released incrementally for Application R&D needs. In that case Platform SW would be ready only in rough level, i.e. needed procedures without real functionality but with needed interface(s) would be ready and more functionality could be added later. However, Application R&D could test their SW with Platform SW and communicate possible changes to Platform R&D in early phase. At the same time SW work would continue in both R&D organizations and after possible changes the SW would be tested by Platform R&D also in FT phase. After iterations from IS to FT (or at least from IS to MT) Platform R&D would release SW incrementally for Application R&D needs and react possible changes in early phase as above mentioned. These iterations from IS to MT have been good solution according to couple of designers. These results regarding iterative development are in accordance with the results reported in the literature (Boelsterli 2003; Highsmith 2000; Hirsch 2005; Hunt & Thomas 2000; Larman et al. 2003; Rautiainen & Lassenius 2001; Schwaber 2007a). Iterative development was also discussed in chapter 2.

In addition, many designers suggested that if requirements are not known clearly, prototyping and workshops could be good solution. Simulation Game (chapter 3.2.1) was suggested to be part of prototyping. During Simulation Game a team is thinking about e.g. how some feature should work (i.e. scenarios). Thus, it would be almost “FT on the table” according to few designers. These results with relation to prototyping are in accordance with the results reported in the literature (Highsmith 2000; Cockburn 2004; Stapleton 1997). Prototyping was also discussed in chapter 2.4.2.

Regarding workshops the proposed solutions were: i) Particular specification team could make IS work only to one release program, not to many release at same time. ii) One or more

SW architect could participate in IS work. IS work would not start until SW architect can participate, i.e. they would not have tasks with higher priority in their daily work. Workshops related results have been reported in the literature (Cockburn 2004; Paetsch, F. et al. 2003; Vriens 2003). Workshops are quite similar than e.g. weekly meetings because e.g. about new ideas and design related details etc. can be discussed within both of them. Meetings & workshops were discussed in the literature review (chapters 2.2 - 2.4). Overall, results with relation to above mentioned workshops seem to be consistent with the literature.

Also on-site customer, continuous integration (chapter 3.3.11), prioritized feature list (e.g. backlog) and small release programs could be good practices for improving of synchronization of release programs (Aho 2006; Palmer & Felsing 2002; Schwaber 2003, pp. 5, 8, 133-136). In addition, fixed time or dynamic length iterations might be suitable for synchronization of release programs. All above practices were discussed in chapters 2.2 - 2.4.

3.3.10 Heavyweight documentation and bureaucratic reviews

According to many interviewees, when documentation is heavyweight and amount of documents is big it takes a lot of time to maintain documents. From research problem perspective it causes delay to release programs. Many interviewees stated that lightweight documentation and split tasks seemed to be good solutions for solving these problems. Depending on situation lightweight documentation seems to be enough and no perfect documentation is needed according to many designers.

According to experiences of many designers, split of documentation work has succeeded in their group, i.e. 2 – 3 designers have responsible for one document (e.g. IS, SD, MT and so on). Split of e.g. SD and MT documentation tasks seemed to be quite easy according to many designers. Due to split of documentation tasks, the documents have been completed faster they noticed. On the other hand, split of documentation requires more competence sharing (chapter 3.3.2) as they told during interviews. Furthermore, few SW architects suggested that it would be good solution that one SW architect would belong to SW group. In consequence of that, feedback (cf. chapter 3.3.7) between designers would be fast and the documentation could be more lightweight than nowadays. Cf. also chapter 3.3.5 in which e.g. group structure was discussed. These results concerning lightweight documentation and split of e.g. documentation tasks to small increments are in accordance with the practices that are recommended in agile methods (chapter 2). Moreover, lightweight documentation and split tasks are also reported in the literature (Aho 2006; Ambler 2002).

Besides documentation related problems also bureaucratic reviews were noticed to cause delay to release programs and decreasing of review bureaucratic was suggested by many interviewees. Formal inspections of minor changes take time due to bureaucracy (chapter 3.3.3) so they proposed that only review with e.g. couple designers would be enough (chapter 3.2.2). Thus, formal inspections are not needed for minor changes according to many interviewees. The formal inspections are typically arranged not until major changes have been made to documentation. One SW Architect from Platform R&D saw that code reviews should be arranged in SD phase, not later when code is already ready. These results regarding reviews are concordant with the results that were discussed in chapters 2.2.1 - 2.2.3. In chapter 2.2.2 it was discussed about code review process in which pairing was used within code review process. In addition, there was a sense that using of pairing has already been proven to be a more effective review process than the traditional code review sessions. Using of pairing within code review process could be one beneficial practice also in Platform R&D and Application R&D organizations.

Possibly it can also be used at least partially within document review process if parts of e.g. SD phase related documents would be split to small tasks. On the other hand, many design details can be included directly into code and also some temporary (no reviewed) work documents could be used. Furthermore, many designers and one line manager suggested that if SD documentation could mainly be included to code the amount of documents or at least content of SD documentation could be decreased, i.e. lightweight documentation. In such situation most of SD documentation could be reviewed already during code review sessions by use of pairing practice, i.e. decreasing of review bureaucratic. Above lightweight documentation improvements and decreasing of review bureaucratic could be suitable for other phases of SW development too.

Besides SD documentation also IS and testing documentation related problems were discussed during interviews. From MT personnel's perspective, when MT is planned IS and SD documents are not always ready which causes problems. If parallel program block's IS/SD is ready it helps MT planning, but it would be better that own program block related IS/SD would be ready. The biggest problem according to MT personnel is that IS documents are changed and typically those are not completed in time. Their opinion was that the IS documents should be at least partially done when MT planning is started as incremental development practices encourage (chapter 2). In addition, one opinion was that IS documents could contain FT scenarios as few designers suggested. In chapter 3.3.5 it was discussed that TDD could be suitable for planning of almost any (e.g. MT, FT and ST) test case. Therefore, if IS would contain FT scenarios it would mean that TDD partially is applied.

Also review invitation process was one topic during interviews. FT persons are typically invited to participate in reviews of IS documents. According to many designers, FT persons don't participate in these reviews every time because time is not reserved enough, although they should participate when designers invite them (cf. chapters 3.3.5 and 3.3.7). This slows down release cycle because it causes extra work after reviews due to e.g. solving possible unclear things in IS document. Furthermore, few designers noticed that if FT scenarios would be added to IS documents the above participation should be in major role as for reviews of the IS documents. The Spike-time (chapter 2.2.3) or Red-Flag practice (chapter 2.4.2) seem to be beneficial practices in this case. Due to above, FT personnel could more active participate in reviews of IS documents in the future. On the other hand, it could be possible that reviewing of IS documentation would be done incrementally by pair or few person e.g. weekly. Therefore, formal inspections would not necessarily be needed for reviewing of IS documentation.

3.3.11 Lack of versioning knowledge of program blocks

According to some designers, removing of feature from ongoing release program causes extra work with versioning of program blocks' modules and from research problem perspective it causes delay to release programs. Versioning of documents seemed not to be so big problem according to interviewees.

Designers from Application R&D told that CVS is used for versioning before freezing code modules and merging of modules from development branch(es) to main branch, i.e. integration of code. It seems to be quite good solution if many designers are working with the same program block. Thus, code modules are available and visible to other designers who can update those modules if needed. CVS is partially used also in Platform R&D.

The results regarding CVS correspond to results that were discussed in chapters 2.2.3 and 2.4.2 - 2.4.3. Substitutive alternative for CVS seems to be Subversion that contains most of features of CVS and many other useful features concerning version control (Mason 2004). Some Subversion tools seem to already support e.g. continuous integration in larger scale than CVS (Mason 2004). Continuous integration is one of recommended practices of XP and it seems to also be one of beneficial practices for organizing of faster release cycle.

XP encourages integration of code could be done daily (chapters 2.1.2 and 2.2) but it seems not to be good solution according to interviewees because tested code is not necessarily ready after one workday. On the other hand, if development would be more incremental (chapter 2) so e.g. daily or at least weekly integration could also be possible in the Platform R&D and Application R&D organizations. Also generating SW build and test report automatically seem to be beneficial as in chapter 3.3.5 was discussed. Furthermore, collective ownership (chapters 2.2.1, 2.2.3 and 2.4.3) and coding standards (chapters 2.2.2 - 2.2.3 and 2.4.2 - 2.4.3) could be beneficial practices too.

According to one designer, also the following solution could be suitable (case by case) and it would bring at least four benefits: i) When a new feature is added a new code module could be created. Therefore, less regression module tests are needed. On the other hand, amount of code modules is not so big problem as extra time needed for regression tests. ii) In this case the testing is needed only for new module and its interfaces. iii) Modularity of code would increase, i.e. low coupling and high cohesion would be achieved. If some other interface than new interface of module or procedure behind that would be changed it does not cause changes for new module's interfaces and procedures. iv) Removing (or inactivation) of code would be easy in case when e.g. some feature should be removed.

Above results regarding modularity are in accord with the results reported in the literature (Haikala & Märijärvi 1997). In addition, the results are in accord with the results described in chapter 2.4.1. According to the literature, high cohesion and low coupling are recommended basic rules in SW development. High cohesion and low coupling increase maintainability and possibility to reuse of SW modules. (Haikala & Märijärvi 1997, pp. 267-268)

3.4 Experiences of Scrum piloting

According to perceptions, during couple of *Scrum Retrospective Meetings* (chapter 2.1.3) and discussion with couple of designers after one daily meeting, the experiences of Scrum piloting were collected. The meetings and discussion session were arranged between December 2005 and February 2006. Data was collected partially also from internal database that contains Scrum sprints related material such as memos of above mentioned meetings.

Overall results of Scrum piloting seemed to be very positive. The Scrum piloting group members were enthusiastic about use of Scrum method in their daily tasks and they were also ready to use Scrum practices in the future. Scrum entailed new practices and variety compared with traditional methods. One expectation was that Scrum makes programming work more interactive and interesting in the future.

Scrum was evaluated to be suitable for CR work if designers' tasks could be adapted to their daily tasks and CR work could be split to many members if needed. CR work typically concerns other groups' work too. Therefore, it would be good that Scrum would be used also in other groups. Thus, synchronization between sprints would be adequate to development work and remaining on schedule would be probable. More Scrum piloting related well-tried practices and problems that were noticed by discussion during meetings have been summarized in Table 16.

Table 16. Experiences of Scrum Piloting

Well-tried practices by interviewees
- Sprint Backlog - Sprint Burn-Down - Meetings - Automated tests - Demo sessions
Problems with proposed solution(s) by interviewees
- Problem: If there are interfaces to other groups (no Scrum in use) so synchronization does not follow sprints. - Solution: Some visitors from other groups have sometimes participated in daily meetings. It would be good if also other parallel groups would use Scrum.
- Problem: Test automation took time, coverage suffered. - Solution: More time should be reserved for testing.
- Problem: One concern was that there is not time to make dynamic task selection.

Due to that, people with best knowledge had to take task(s) during sprints.

- Solution: Dynamic task selection is tried to make possible in the future.

Since piloting was started SW was developed incremental and iterative way during one month sprints. Sprint planning session was arranged at the beginning of each sprint. Piloting group considered it to be very useful practice because it enabled possible asking of questions regarding requirements. Consequently, requirements become clearer if needed. During these sessions, requirements were given monthly by Product Owner who has good knowledge of product. The requirements were clear and all requirements were implemented.

During piloting, sprints had been consisted of mainly from SD phase to MT phase. Tasks for each sprint were described in Sprint Backlog (Appendix 4: Scrum's Sprint Backlog) and remaining time was shown in Sprint Burn-Down (Appendix 3: Scrum's Product Burn-Down chart). During sprints, daily meetings were arranged. There was shown what has been done and what will be done. Scrum also forced to proceed every day. Furthermore, automated tests that were executed during sprints were noticed to be useful practice. At the end of sprints the demo sessions were arranged. Knowledge had been increased due to demo session and above meetings.

3.5 Summary of results from interview study

For a start this interview study the interview method was introduced in chapter 3.1. The interview method was used to collect well-tried practices (chapter 3.2) and problems with proposed solutions (chapter 3.3) during interviews. Both Platform R&D and Application R&D aspects were included. Also experiences of Scrum piloting were collected from Scrum piloting team from Platform R&D and partially from memos in internal database (chapter 3.4).

In consequence of this interview study, answers to second and third research sub-questions (chapter 1.3) were found. Concerning mainly situation when some feature is added to ongoing release program in late phase the following problems were discovered during interviewees:

1. Lot of parallel tasks in daily work
2. Lack of competence sharing
3. Bureaucratic decision making
4. Underestimation of needed resources
5. Inadequate co-operation between testing and SW groups
6. Lacking Feature Owner
7. Inadequate communication
8. Late changes to the content of release programs
9. Inconsistent synchronization of release programs
10. Heavyweight documentation and bureaucratic reviews
11. Lack of versioning knowledge of program blocks

According to interviewees, (re)moving some feature from ongoing release program in late phase seemed to not be so big problem than adding some feature. However, same problems seem to effect in both cases. E.g. from project and program management perspective, every CR causes lot of re-planning (chapter 3.3.8) regarding scheduling and causes delay to release cycle. Re-planning is also caused by (re)moving some feature from ongoing release program in late phase. To solve all above problems many proposed solutions were collected from interviewees and partially also from the literature.

Also things which appeared as well-tried practices were discussed in the interview study. These well-tried practices mean such practices that are not necessarily needed to change in Platform R&D and Application R&D organizations. Overall, results from this interview study can be used to be part of constructing the XFRC process model.

4. EXTREME FASTER RELEASE CYCLE (XFRC) PROCESS MODEL

Dynamically applicable and scalable process model seems to be a sine qua non of organizing of faster release cycle. Without adequate process model SW development certainly ends in a complete shambles one of these days.

Mika Tanskanen

In consequence of the literature review and interview study, answers to three research sub-questions were found. These answers enabled constructing the XFRC process model that is made public in this chapter. Constructing of the XFRC process model was needed in order to find answers to research main question.

At first overview to XFRC process model is introduced in chapter 4.1. Thereafter, constructions are described in chapters 4.2 - 4.5. The constructions consist of XFRC process model with recommended values, practices, roles and responsibilities for organizing of faster release cycle from SW development aspect. Values are collection of guidelines for successful use of practices by different roles and responsibilities.

The XFRC process model is illustrated in chapter 4.5. Constructions as well as interpretations are based on above answers to sub-questions. Consequently, references are not described singly in this chapter because used references with relation to this research have already been described in chapters 2 and 3.

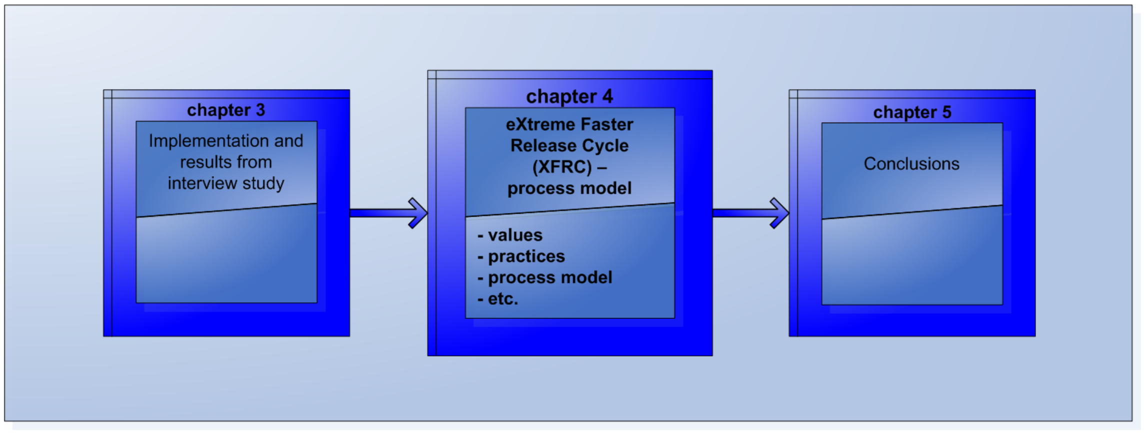

chapter 3
Implementation and results from interview study
chapter 4
eXtreme Faster Release Cycle (XFRC) – process model
- values
- practices
- process model
- etc.
chapter 5
Conclusions

Figure 11. Introduction to chapter 4

4.1 Overview to XFRC process model

Main aspects according to research main question were utilized to construct process model which was assumed to be the beneficial model for organizing of faster release cycle in SW product development organizations. It should be beneficial especially in situation such as the content of ongoing release programs is continuously changing in late phase. Main purpose of the XFRC process model is that parallel tasks could be decreased, competence sharing and communication could be increased, and late changes to release programs could be forecasted. Due to above, SW development in R&D organizations could be organized more efficient and dynamic way than nowadays. In consequence of better organizing, faster release cycle could be made possible in the real world.

The XFRC process model consists of recommended values, practices, roles and responsibilities. Recommended values are discussed in chapter 4.2. Compliance with all described values is recommended in order that the XFRC process model could successfully be used and adapted through R&D organizations. Compliance of the values requires e.g. commitment which is one of recommended values.

Recommended practices are discussed in chapter 4.3. These practices are collection of recommended practices of agile methods that have proven to be beneficial in different organizations according to the literature review and the interview study. Recommended roles and responsibilities are discussed in chapter 4.4. Roles and responsibilities are based on applicable compromise of recommendations that were discussed in the literature review and the interview study. Role differentiation with different and occasionally rotated responsibilities is essential for e.g. decreasing of parallel tasks in daily work. Also many other improvements are needed.

The XFRC process model is illustrated in chapter 4.5. It describes an alternative approach to enable faster release cycle along with efficient and agile SW development, without waste trivia. The XFRC process model is dynamically applicable and scalable through R&D organizations where SW is developed. From usability perspective, SW designers could develop SW more productively, efficiently and enjoyable to achieve specified goals in specified context during release programs when their tasks would be organized with the XFRC process model. That model could be exploited also in other organizations in telecommunication business. At least parts of that process model could probably be exploited in other industries too.

4.2 XFRC values

The research results suggested that following values could be suitable for daily work in R&D organizations:

Table 17. XFRC values

Value	Description
Adequate communication	- increasing co-operation between groups - multi-site aspect should be noticed - minimizing of asking same questions by different management boards - meetings & workshops are arranged dynamically only when needed, not necessarily e.g. daily - possibility to get fast feedback from anyone
Applicable commitment	- use of agile methods needs commitment - also continuous development of own working methods are encouraged
Continuous and innovative learning	- dynamical job rotation - switching competences between groups - helpfulness to solve problems with colleague
Dynamical resource allocation	- resource pools for e.g. CR work (also parallel release programs should be noticed) and maintenance (fault corrections included) work - innovative learning, unexpected work and prioritization must be noticed when resource allocations are done - dynamically adjustable iterations concerning resources and length of iterations
Dynamical SW development	- dynamical changing of content of release programs - no adding of features during an iteration - (re)moving of feature is possible also during iteration
Dynamical SW process improvement	- open SW process improvement - continuous and dynamical SW process improvement through R&D organizations - improvement suggestions could be collected into e.g. electronic bulletin board in intranet
Minimum bureaucracy	- decisions should be made by low-level management without waste trivia - bureaucracy concerning documentation and reviews is recommended to minimize
Modularization	- high cohesion and low coupling - versioning could follow same basic rule during (e.g. feature specific) release program, i.e. 1.1-0 -> 1.2-0 -> 1.3-0 -> 1.n-0 during release program x and 2.1-0 -> 2.2-0 -> 2.3-0 -> 2.n-0 during release program x+1 if there are major differences between them - due to above, e.g. 1.1-1 etc. version are not necessarily needed except for fault corrections case by case
No parallel tasks	- dedicated persons for e.g. CR and maintenance work - dynamical resource allocation etc.

Adequate communication

Firstly, to enable adequate communication through R&D organizations, designers and testing personnel are encouraged to co-operate with each other. Marketing, owners (e.g. Product Owner), managers (e.g. product manager) are also encouraged to communicate with each other. Every personnel group could have some contact persons for enabling fluent communication between different personnel groups. Also multi-site aspect should be noticed concerning arrangements for contact persons. In that case there would be contact persons inside release programs through multi-site. If distance between communicating persons does not enable face to face communication they could communicate e.g. by chat, mail or any appropriate phone. However, it would be better if release programs would be country specific in which case face to face communication can be enabled at least between persons working in same building. On the other hand, e.g. video conference meeting is partially comparable with face to face communication.

Line managers and project managers should communicate with each other. Line managers could be as primary contact between SW and testing groups (ST group if MT and FT are executed in SW groups) and management. Inside SW and testing groups could be named contact persons between those groups and other SW and testing groups. In consequence of above, it is possible to minimize asking of same questions by different management boards twice or more. However, communication could be increased due to above arrangements.

In addition, meetings & workshops could dynamically be arranged only when needed, i.e. the meetings are not necessarily needed to arrange e.g. daily. It is recommended that meetings could be arranged immediately when needed if any matter does not prevent arranging of that meeting. It requires commitment concerning reservation of time for meeting so that all participants can arrange their tasks to participate in these meetings. So called wasted and fixed time meetings are not necessarily appropriate. Instead at same time they could work with daily tasks without participation in wasted meetings. If e.g. some problem should be solved or some decision should be done arranging of meeting is typically good alternative. Length of meeting could be dynamically adjusted until needed actions have been done during meeting. Of course, length of meeting should not typically be many hours. Also time reservation for meeting should be reasonable. Furthermore, different sitting arrangements, communication tools (mail, mobile phone, conference phone, electronic bulletin board, chat etc.) can increase communication into adequate level. If e.g. language barriers decrease communication, a mail is typically easier way to communicate with second language than speaking. On the other hand, language courses are available too.

Applicable commitment

Commitment concerning use of agile methods and continuous development of own working methods with them are encouraged. Therefore, SW development etc. can become more effective according to organization needs.

Continuous and innovative learning

Competence sharing by dynamical job rotation inside team is recommended. Job rotation could be decided by resource pool and prioritization of tasks before iteration starting point. Therefore, dynamical resources can be allocated also for switching competences according to organizations needs. After reaching aimed competence expanded learning about challenges outside team could be one alternative. It is recommended if some person wants to expand learning about issues outside team. Helpfulness to solve problems with colleague is needed so that competence is increased and e.g. SW development becomes more efficient within team. XFRC practices (e.g. pair programming for hard tasks and arranging meetings) can be exploited concerning problem solving. In addition, e.g. SW work, MT and FT work could be combined if competence sharing would be increased. Therefore, different MT and FT groups are not necessarily needed if testing persons are included to SW groups and job rotation would be used.

Job rotation could also be beneficial when it would be expanded to concern management. Every group member could have turn to manage group during e.g. iteration. Line manager of group could work as coach and give examples from management and leadership perspective if manager has enough knowledge about those aspects. Therefore, every group member could derive management experiences in long run. It would be possible also from project management perspective. Estimations and resource allocations could be done by groups and also resource pools could be beneficial. Resource pools are discussed more in "dynamical resource allocation" section in this chapter. If groups would have enough experiences concerning line and project management the groups could be as self-organizing groups. It is possible that self-organized groups will increase in the future. In this case many actual line and project managers would not necessarily be needed in R&D organizations. Instead, also managers could expand their knowledge according to their own and organizations needs. Briefly, job rotation can be utilized through organizations and in spite of current role in organization.

Dynamical resource allocation

Resources could be moved dynamically by using resource pools for e.g. maintenance work and CR work. Resource pools could also be used for other work such as general SW work during release programs, HW work and possibly management work. However, it is essential that parallel tasks are not allocated for anyone. Instead above mentioned dynamical resources should be allocated and person's own predilections should be noticed. In addition, e.g. task as "innovative learning" could be included to be as part of resource planning for release programs. At least 20% of total work of release programs should be reserved for innovative learning. E.g. reading CR related specifications, learning about interested and beneficial issues, SW process development improvements etc. can be included to be as innovative learning. Also unexpected work during release programs should be noticed when resource allocations are done. Both innovative learning and unexpected work could be estimated in rough level when release program is started and with details before start of iterations.

Resource pools can be used when allocated work effort concerning future iterations that are not necessarily periods of certain length, i.e. iterations are dynamical adjustable periods depending on planned work amount for next iterations. Also skills for e.g. splitting tasks and making estimations affect length of iterations. In brief, when planned work for iteration has been completed a next iteration can be started. It is possible if resource pools would be used. It means that person(s) can help colleagues in same or parallel group regarding their tasks if planned tasks have been completed before end of the iteration. Due to that, all planned work for the iteration should typically be completed by group at the latest during few weeks. In other case in which above person(s) are doing last task before end of iteration, other persons could help them to complete their tasks.

Before starting of iterations resources should be taken into consideration also from prioritization perspective when allocating resources for next iteration. Prioritization of work could be decided within meetings that are arranged when tasks for starting iteration are planned. If a lot of e.g. high priority CR work is coming, resources could be allocated by priorities and above predilections. Otherwise, resources can be allocated e.g. by high priority of possible fault corrections. Designer could correct some fault(s) during coming day(s) if the fault(s) have been prioritized with high priority. When completing above fault correction(s) designer could start e.g. CR work according to prioritization of the moment work.

Dynamical SW development

Content of release program can be changed dynamically. However, adding feature or certain part of it is not possible until next iteration is started. Adjustment of content is possible by dropping feature or part of it to next iteration. In this case (re)moved part of feature can be implemented during next iterations.

Iterative and incremental SW development is recommended. Length of iterations could vary from e.g. few days to few weeks. It does not necessarily need to be fixed time such as one month that is typical length of iteration in Scrum. Increment could be e.g. one or more procedures with needed interface(s). Typically it is not illustrated how large entity increment is when agile methods are discussed. However, it can be dynamically adjusted part of task that is part of some feature. Therefore, e.g. one procedure without real functionality but with needed interface(s) could be smallest and suitable increment in real SW development. Real functionality could be developed during iterations and hard-coded MT tested functionality could be as functionality that enables exploiting by Application R&D (in case Platform SW) or FT by Platform R&D in this case.

Such practice as dynamical integration with e.g. CVS or Subversion makes possible to develop same modules at the same time and by many person. In addition, executable files of program blocks etc. are needed for generating SW build. CVS is not necessarily suitable for version control of executable files. It is suitable for version control of code modules and so called main modules for those. Changes regarding main modules could be listened by automatic compiler that could compile all changed program blocks etc. and generate at least unofficial SW build automatically. Main modules should contain needed definitions so that changed code modules could automatically be searched and taken automatically into compilation from Subversion because CVS does not necessarily support such operation. In consequence of above, continuous automatic generation of unofficial SW builds after continuous automatic integration would surely be possible in practice. Therefore, generation of SW build is not needed to synchronize with e.g. ends of iterations but unofficial SW builds would be available continuously until official SW build is generated. On the other hand, also official SW build could be generated at any time automatically if functionality can be proven with needed tests. It means that after generation of SW builds automatically all needed tests should be executed automatically. Also test report should be generated automatically. If some test fails notification to responsible person could be sent automatically. Above is part of automatic and anticipatory testing which enables finding possible faults during regression testing, i.e. although some change concerns specified part of some feature it is possible that some secondary effect fault has appeared too. In addition, extra delays with relation to testing can be minimized by using above recommendations.

If some major changes should be done these changes could be done during next release program. Therefore, not many official SW builds are needed during one release program but continuously automatically generated SW build would be enough for one release program. It could be generated automatically also during maintenance phase. Briefly, due to increased automatic concerning SW development, manual phases could be decreased.

Dynamical SW process improvement

From SW process improvement perspective continuous and dynamical SW process improvement is recommended. It makes possible to modify SW process according to dynamic changes in market and needs of R&D organizations. One part of process can suit for needs of some organization and some other part can be suitable for needs of some other organization. In ideal situation overall process can be adjusted to be suitable for both organizations in this case.

SW process improvement suggestions could be collected e.g. monthly from Platform R&D and Application R&D organizations. Adequate communication could be exploited and contact persons in groups could collect improvement suggestions from groups. List of collected improvement should be place that is visible to everyone in these organizations. Everyone should have rights to update that list. The list can be e.g. on electronic bulletin board in intranet. E.g. Q&P organizations could handle list of improvements e.g. monthly and execute needed actions through R&D organizations. Actions could be communicated to contact persons of groups and contact persons could share actions related information through groups.

Alike open source code can be developed by anyone the XFRC process model can be modified and adapted according to needs of organizations. Use of values and practices etc. depend on an organization needs. In conclusion it can be stated that possibility to modify process model through organizations can be concluded to be as open SW process improvement in brief.

Minimum bureaucracy

Decisions should be made by low-level management without waste trivia. Minimum bureaucracy is needed also from documentation perspective. Extra (heavyweight) documents are not needed. Instead such practice as automatic and lightweight documentation is recommended. E.g. SD documentation is waste if needed design details are added into code and checked during code reviews that should be arranged during iterations.

Reviews of code and documents should be arranged concerning e.g. CR work or maintenance work related changes. Extra review minutes are not needed but minimum bureaucracy concerning reviews is recommended. Reviewed material should be split so that all participants of reviews would not need to look through all reviewed material because it takes time if reviewed material is large. On the other hand, reviewed material should be compact, i.e. it should not be too large. However, if reviewed material is large e.g. couple of review session can be arranged. When the review session is ended it would be enough to mark the "completed" information to check list with a cross. Instead review sessions reviewing could be arranged by pair working. In that case reviewing could be continuous without large reviewed material if reviewing of small part of increment would be done e.g. daily.

Minimum bureaucracy should be noticed perspective of tested functionality too. If some functionality has been tested not much bureaucracy is needed but the "completed" information to check list with a cross would be enough. In addition, managers can learn about different management theories and leadership principles by finding related information in the books etc. They could also ask feedback from SW designers in order to develop their competence regarding management and leadership.

Modularization

Due to commitment to high cohesion and low coupling for SW development e.g. versioning and integration could succeed at least well. Also splitting of specific feature related parts of code could be easier than of inconsistent code. Furthermore, testing and split test cases for specific part of some feature could be easier than e.g. case with relation to high coupling. Overall maintenance of program blocks etc. could probably be easier if modularization perspective is taken into consideration.

No parallel tasks

Dedicated persons for e.g. CR work, maintenance and other responsibilities could be named inside groups. Although some person would have more than one responsibility, daily tasks should not consist of parallel tasks. Instead some new task should not be started until after current task has been completed. If any task is interrupted before it has been completed it takes time to go back to work with that task. By splitting of tasks to suitable entities it does not take a lot of time to complete it. Suitable entity means not more than couple of days work. E.g. split daily tasks enables forming suitable entities. If splitting of tasks seem to be not possible so maybe splitting did not succeeded in the real word. In that case re-estimation is recommended.

Above splitting and avoidance of parallel tasks are recommended to belong to part of daily work. Also iterative and incremental SW development, meetings and team's possibility to make decisions regarding e.g. programming are recommended. Use of above mentioned dynamical resource allocation with resource pools for e.g. CR work and maintenance are recommended too. In addition, if resources are not available enough according to the resource pools many parallel release programs are not recommended to start because it surely causes parallel tasks if resource allocation fails due to some reason. Then release programs should be started only for major changes concerning existing feature or adding of large feature. Adding minor changes etc. could be started to ongoing release program in the beginning of next iteration.

4.3 XFRC practices

Every R&D organization can pick agile methods recommended practices to enable efficient SW development and faster release cycle. These practices are described in Appendix 1: XP practices, Appendix 8: Scrum practices and Appendix 11: Other practices. Platform R&D and Application R&D organizations can also pick recommended proposed solutions for use in daily work from results of interview study. These results are summarized in Appendix 10: Summarized problems with most essential proposed solutions. Also dependencies between proposed solutions should be noticed. Furthermore, adding, removing and modification of above practices and proposed solutions are possible depending on market and needs of organizations, cf. dynamical SW process improvement which is one of the XFRC process model recommended values. However, the research results suggested that at least following practices could be suitable for daily work in R&D organizations:

Table 18. XFRC practices

Practice	**Description**
Incremental and iterative development	- overall (specification, design, implementation and testing) SW development is recommended to execute by incremental and iterative way during dynamical length (e.g. from few days to no longer than few weeks) iterations - prototyping could be used for large features - Use Cases could be used for planning of test cases (probably suitable also for other issues) concerning automatic and anticipatory testing
Small release programs	- at least features, release programs and tasks are split - easier resource allocation and synchronization - freezing content for dynamical length iterations, i.e. adding of feature is not possible during iteration
Meetings & workshops	- beneficial solutions to solve and decide actually almost anything - meetings can be arranged at least at the beginning and at the end of iterations - meetings are recommended to arrange during iterations only if needed, i.e. not necessarily e.g. daily
Automatic and anticipatory testing	- automated generating SW builds - automated generating MT, FT, ST and SW build test reports after automated testing - TDD practices should be used through all testing phases - automated sending of feedback concerning possible fault (e.g. some program block is not compatible to new SW build) to responsible person - regression testing from anticipatory perspective

Practice (continued)	**Description (continued)**
Dynamical integration	- continuous integration with e.g. CVS or Subversion - shared data warehouses (e.g. data types and other definitions etc. included) for Platform R&D and Application R&D) make possible to integrate all definitions etc. that were used in both organizations - possibility to take integrated and at least MT tested Platform SW code for FT by Platform R&D - possibility to take above Platform SW for use in Application R&D - possibility to take integrated and at least MT tested Application SW for FT (and ST) use in Application R&D - possibility to take also late (minor) changes
Pair working	- could be used for problem solving, i.e. mainly for hard tasks → pair programming & pair specification - could be used for testing → pair testing - could be used for coaching → pair coaching - could be used for code and document reviews at least as part of review arrangements with minimum bureaucracy → pair reviewing - enables increasing competence sharing - depending on situation forming pairs is only one alternative, e.g. trio and other alternatives are surely beneficial in most cases
Automatic and adequate documentation	- light and simple but exact adequate documentation with commitment to minimum bureaucracy concerning e.g. reviews - documents could be generated automatically - part of documents could be only temporary, i.e. these non-official documents could be used only for supporting SW development during iterations
Electronic bulletin board	- could be used for collecting SW process improvement related suggestions through organizations - could be used also for other ideas regarding suitable categories - could contain list of further actions too
Backlogs	- backlogs for e.g. product, features, release programs (split features), projects (split feature programs), iterations during projects and tasks during iterations - backlogs could be named according to specific purpose of use in order that backlog is easy to find and associations to specific matter are easy to remember

Above practices have already been discussed partially in chapter 4.2. Descriptions of these applied practices have already been discussed from their original perspective in above appendices too. The practices are also discussed partially in chapters 4.4 and 4.5. Therefore, more discussion about these practices is bypassed in this chapter.

4.4 XFRC roles and responsibilities

The research results suggested that following roles and responsibilities could be suitable from group formation perspective in R&D organizations:

Table 19. XFRC roles and responsibilities

Role	Responsibility
SW designers (general SW work, CR work and MT included)	- two SW designers who would typically participate in general SW work during release programs - from testing perspective they could execute MT inside group - possible CR work can be started in the beginning of next iteration - recommended contact person responsibilities could be split concerning CR work
SW designers (general SW work, maintenance and FT included)	- two SW designers who would typically participate in general SW work during release programs - from testing perspective they could execute FT - possible maintenance work (fault corrections) can be started in the beginning of next iteration - recommended contact person responsibilities could be split concerning maintenance work and FT - one designer could also work as contact person concerning CR work done by the other designers inside group
SW architect (specification work, pair coaching included)	- two SW architects who would typically participate in specification work - they could work also as coach in job rotation case if needed - recommended contact person responsibilities could be split concerning specification work
Line manager	- resource allocation by asking estimations from staff (group members in same groups) and following resource situation from resource pools - recommended contact person responsibilities could be split concerning management and leadership tasks - in spite of above tasks, managers would not have authority regarding group because it would be self-organizing group

SW group could consist of four SW designers, two SW architects and one line manager. Two SW designers would typically participate in general SW work during release programs. From testing perspective they could plan and execute MT. In brief, one (first) designer could plan and execute MT for code implemented by pair or by other (second, third or fourth) designers inside group. If CR work concerns group both designers could interrupt general SW work and start CR work in the beginning of next iteration. One designer could work as contact person between this group and parallel group concerning CR work. The other (second) designer could work as contact person between this group and the other parallel group concerning CR work.

Besides above two designers there could be two SW designers for maintenance work. However, also they would typically participate in general SW work during release programs. If some fault correction should be done to program block from which group is responsible for, they could interrupt general SW work and start maintenance work (fault corrections) in the beginning of next iteration. From testing perspective concerning general SW work and CR work they could also plan and execute FT if resources are not needed for maintenance work at the same time. One (third) designer could work as contact person between this group and parallel group(s) concerning case when same fault requires correction done by parallel group(s). The other (fourth) designer could work as contact person between this group and parallel group(s) concerning related FT executed by parallel group(s). The other designer could work also as contact person inside group concerning CR work done by the other designers.

Two SW architects would participate in specification work. They could work as coach when designers are participating in specification work due to recommended job rotation. On the other hand, SW architects could participate in other SW work than specification in job rotation case if coaching is not necessarily needed after designers have achieved enough core competence regarding specification work. One (first) SW architect could work as Group Owner concerning part of some feature and work also as contact person between persons in similar positions (e.g. Project -, Release - and Feature Owner). The other (second) SW architect could work as a contact person between Platform R&D and Application R&D, i.e. SW with relation to same feature is developed in both groups in different R&D organizations.

Resource allocation by asking estimations from staff (group members in same groups) and following resource situation from resource pools could be done by line manager. Line manager could work as contact person between project management and this group. In addition, line manager could work as contact person between line managers in other groups and this group. Typical management and leadership tasks except authority for self-organizing group concerning SW development practices could be included in line managers responsibilities.

Above roles and responsibilities are one alternative from group formation perspective. All other perspectives concerning efficient SW development should also be noticed in daily work. As conclusion it could be stated that besides above roles and responsibilities all XFRC values and practices are recommended to exploit in daily work. In addition, above responsibilities regarding few examples of job rotation and recommended contact persons between groups are demonstrated in Figure 12:

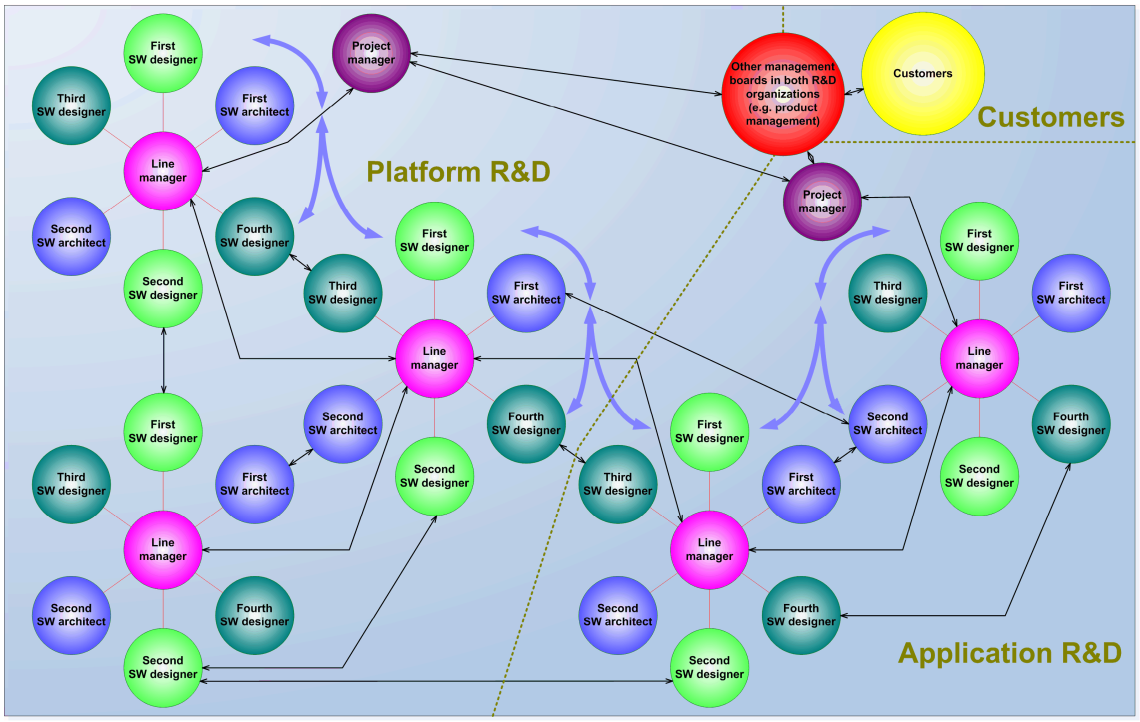


Figure 12. Examples of recommended contact persons and job rotation

Job rotation related examples are illustrated with thick two-way arrows in Figure 12. One alternative is to exploit job rotation after completed tasks or according to other suitable plans of actions as completed iterations or completed release programs. Pair working that was recommended to be one of XFRC practices is certainly one essential aspect concerning the success of job rotation. When job rotation is expanded outside groups, people who form pairs (or e.g. trio) can come from different groups. Also e.g. adequate communication and dynamical resource allocation are necessarily needed.

As above responsibilities described the contact persons could also be named through organizations. In Figure 12 communication between possible contact persons are illustrated with thin two-way arrows. Communication inside groups was not illustrated in Figure 12. However, contact persons could be named also inside group as XFRC roles and responsibilities recommend. These aspects can be decided in detail level by R&D organizations according to their own needs.

4.5 XFRC process model

When connecting illustrated XFRC values, practices, roles and responsibilities the XFRC process model can be visualized as Figure 13 shows:

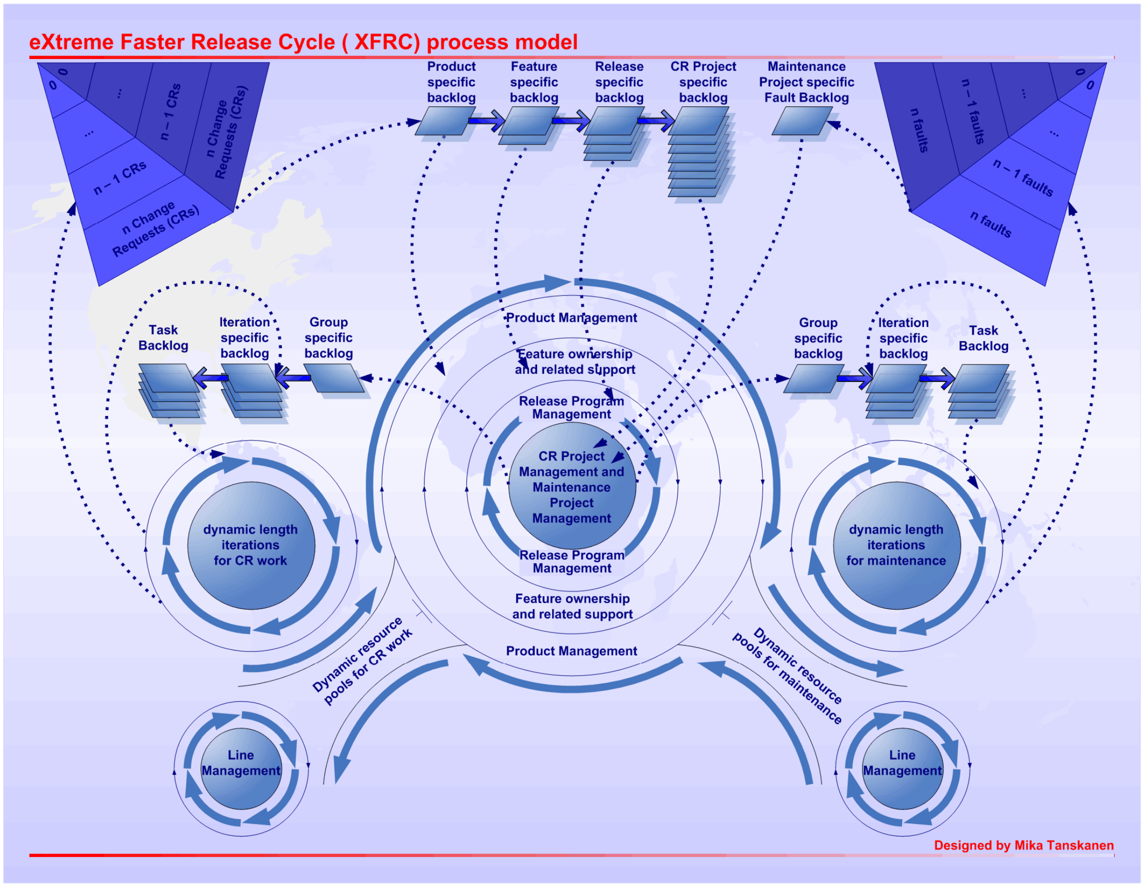

eXtreme Faster Release Cycle (XFRC) process model
0
...
n – 1 CRs
n Change Requests (CRs)
Product specific backlog
Feature specific backlog
Release specific backlog
CR Project specific backlog
Maintenance Project specific Fault Backlog
n faults
n – 1 faults
Task Backlog
Iteration specific backlog
Group specific backlog
Product Management
Feature ownership and related support
Release Program Management
CR Project Management and Maintenance Project Management
dynamic length iterations for CR work
dynamic length iterations for maintenance
Dynamic resource pools for CR work
Dynamic resource pools for maintenance
Line Management
Designed by Mika Tanskanen

Figure 13. eXtreme Faster Release Cycle (XFRC) process model

In the start of the XFRC lifecycle, some feature is required to add to a product or remove from a product. Request concerning a feature is made by a customer. Feature is typically related to some product. Therefore, list of a product related features can be described in backlog that is owned by Product Owners (e.g. product managers) from both Platform R&D and Application R&D organizations. This backlog could be named according to specific product as e.g. RNC, i.e. name of the backlog would be RNC Backlog in this case. Above backlog can be split into e.g. two entities, i.e. one feature specific backlog for Platform R&D and one feature specific backlog for Application R&D because feature specific release programs are illustrated in this case. Feature Owners from both organizations (e.g. Senior SW architect with good knowledge about that feature) could own the feature during its lifecycle and administrate backlog that is named to be feature specific. In this phase different management boards and owners (e.g. Release Owner and Feature Owner) start to plan needed estimations and content for release programs. In any case the feature should be split into smaller entities due to planning of content of release programs.

If feature has effect on SW developed by both Platform R&D and Application R&D as in this case the feature could be split into two release programs. Other release program is for Platform R&D and other is for Application R&D in this case. If feature is very large it is possible to split it into several entities. If feature is small one alternative could be to create a project and attach it to part of ongoing release program. However, typically release programs are recommended to be feature specific in both above R&D organizations. Platform R&D can administrate release program specific backlog for planning content of release programs that are started or have already been started in Platform R&D. Also future release programs related content could be described in release program specific backlog if content is known at least rough level, i.e. some new feature has already been included in feature specific backlog of particular organization. Application R&D can administrate own release specific backlog that contains split features per release programs.

Release Owner (e.g. program manager) could own the organization related part of the feature. Release programs could be split into projects and content of these projects can be described in project specific backlogs. Project Owner (e.g. project manager) could own the project related part of the feature. Faults could be managed by Maintenance Project Managers and faults could be described in Maintenance Project’s specific Fault Backlogs. Faults could be corrected during dynamically length iterations, i.e. fault has been corrected when correction is ready. It is not necessarily ready after fixed period such a day despite plans. It is also possible to include faults into some other backlog(s) that are illustrated in this chapter. Also splitting of faults could be similar as splitting of CR work if effects concern e.g. many program blocks. If backlogs are decided to separate from perspectives of maintenance management and CR work these separated backlogs could be similar for maintenance work as for CR work. However, detailed discussion concerning maintenance management is bypassed because it belongs outside the scope of this research.

Work amount of projects can be split to affected SW groups and testing groups that can administrate their own group specific backlogs concerning part of developed feature. E.g. SW architect in every group could own group related part of the feature. In testing groups some engineer could take principal responsibility for successful testing of the feature. For iterations the group specific backlog could be split into Iteration Backlog. The Iteration Backlog could be generated again after each iteration depending on what has been done and what will be done during next iteration.

Related work that is described in Iteration Backlog can be split into smaller tasks by group members. In ideal situation these tasks can be independent of other tasks. Due to above splitting, every responsible person can administrate their own backlog that could be e.g. Task Backlog and therefore own parts of the feature from task level perspective. In consequence of use of Task Backlog, dividing of tasks can be clear and overlapping between tasks can be decreased especially in case when many persons are responsible for same program block. Dividing related planning could be done during meetings in the beginning of iterations.

Prioritization perspective concerning all above backlogs should be noticed too. Depending on amount of prioritized tasks, iterations could be adjusted to be dynamic length iterations as e.g. from few days to few weeks as dynamical resource allocation recommends. During these iterations SW is developed incrementally by exploiting XFRC values, practices, roles and responsibilities. This can be understood as dynamically length period which contains many dynamically iterations to complete adapted development phases of SW development. One completed task or collection of completed tasks can form increment when IS, (SD), MI and test phases concerning task(s) have been completed during dynamic length iteration. Concept of increment depends on backlog level in the XFRC process model, i.e. one completed small task can form one small increment and many completed small tasks can form many small increments e.g. in task level. In last mentioned case it is also possible to find one or more large increments e.g. in group level and so on.

One example to produce workable and suitable increment is to invite required SW development persons (see chapter 4.4) to participate in meeting. During meeting it could be split responsibilities regarding IS, (SD), MI and at least MT test phases to participants. Cf. job rotation. Duration of meeting could be e.g. few hours including needed breaks. In consequently of above, all participants could participate in any phase to complete workable increment in the real world. If couple of SW architects with one or more designer would specify some part of feature as a small increment of product it enables possibility to make needed test cases concerning the increment. When test cases would be ready for testing of functionality of the increment so possible faults could be found even during same working day. MT with exploiting TDD practices could be used at the same time with MI when specification would have been ready due to group work during above meeting. After first iteration, it would be enough that above would be ready on rough level. More details could be added during next iterations and so on.

From integration perspective implemented and at least MT tested code is integrated dynamically. It can be dynamically integrated after suitable increment is ready for later tests that have already been planned and implemented by exploiting automatic and anticipatory testing. It means that tested code is integrated many times during week because increments consist of small parts of functionality. Depending on compatibility of code regarding other parallel release programs e.g. pre-compiler directives (i.e. compiling switches) such as *#ifdef* and *#endif* can be used so that major changes for versioning is not necessarily needed. In addition, if many persons are responsible for same program block, code should be regularly integrated. Therefore, other responsible persons can take the integrated code into use and make possible changes also to modules that are administrated by other designers.

From automatic and anticipatory testing perspective all (MT, FT and ST) tests should be automated although it takes time to automate them. SW builds should be generated automatically. All essential tests should be automated and TDD should be used. Also updating of test cases should be automated, i.e. if e.g. some data type has been changed in data warehouse it should be possible to update test cases automatically or at least by command which starts automatic updating function. Above makes possible that all automated tests can be executed whenever necessary. Test reports should be generated automatically as the practice automatic and adequate documentation recommends. Also generation of other documents should be automated as much as possible and generating useless documents should be avoided. E.g. code and FT cases could probably be generated at least partially automatically from sequence diagrams that are typically generated during IS phase. SD could be included in code and test plans could be included in test cases.

If some fault is noticed during tests automatically feedback and request to correct some fault should be transferred to right responsible person(s) in a team. Notification could be sent automatically if location of fault could be proven automatically, albeit it takes time to prove direct location of fault that can be located in one or more program block. From anticipatory testing perspective, regression testing should be executed during iterations if possible. It is better to find faults as soon as possible in the beginning of release program rather than in late phase of that program. Therefore, it is essential to try to emphasize the importance of testing at early stage. In addition, small release programs enable above.

When SW has dynamically been integrated and tested, Application R&D could take at least MT tested Platform SW into use during their own tests. That SW could be implemented and tested more detailed in Platform R&D during iterations. Release programs could be started somewhat earlier in Platform R&D than Application R&D. It enables that above SW would be developed and MT tested before starting developing by Application R&D. Therefore, probably few increments developed and MT tested by Platform R&D would be ready before start of release program in Application R&D.

At the end of iterations demo sessions could be arranged. For example, one group in Platform R&D has developed one part (increment) of the feature during iteration. That increment could be utilized by customer as e.g. couple groups from Application R&D in this case. Therefore, few participants from related groups could arrange meeting and group from Platform R&D could demonstrate functionality of increment. On the other hand, many increments maybe developed by many groups during iteration of release program. It depends on size of increments and length of iterations. On the other hand, iteration of release program can also include sub-iterations if size of increment is e.g. only one procedure and needed interface(s).

At the beginning of next iterations the related backlogs should be updated and needed plans for next iterations should be done. If some minor changes are needed to implement in late phase or any phase of release program those changes can be taken into starting iteration. Above changes should not be taken to ongoing iteration because it causes waste interruptions, parallel tasks and other waste work concerning ongoing development. When some minor change is taken into release program at the beginning of the next iteration, it does not cause too much waste work because e.g. planning is done then in any case.

(Re)moving related changes concerning release programs can be take into consideration in case when some minor feature should be dropped from ongoing release program. On the other hand, if (re)moving is not done due to unconditional necessity (e.g. request by customer) so e.g. dynamic iterations make possible that (re)moving is not necessary. If only one major feature is developed during release program and that major feature should be (re)moved it means that release program should be ended. In this case above major feature could be taken into future release program if it is decided to implement later.

For major changes new release program is recommended to be established if those changes are not with high priority. If those major changes are with high priority the changes can be split and prioritized. After that those changes can be done during next iterations. In this case designers who are working with maintenance tasks could interrupt maintenance work after they have completed ongoing incomplete corrections. Therefore, they could participate in CR work during next iterations until high prioritized CR work is completed. If priority of some fault is occasionally higher than ongoing CR work some designers could start to correct those faults after ongoing tasks have been completed and next iteration is started. Designers could take fault corrections into SW build along with dynamic integration. They could participate in CR work after completion of those tasks.

Priorities concerning CR work can be followed by resource pools. Resource pools and all other recommendations that were discussed in this chapter could be suitable for general SW work and maintenance work too. Also parallel release programs should be noticed when resource pools are used but tasks done by e.g. designer should not be parallel. Either many parallel release programs are not recommended to start because it typically causes parallel tasks from SW development and resource allocation perspectives. In conclusion, it can be stated that above recommendations that were discussed in this chapter concerning exploit of the XFRC process model are profitable when overall SW development is made more effective and along with it release cycle is made faster through organizations.

4.6 Summary of XFRC process model

For constructing the XFRC process model many beneficial experiences such as well-tried practices and problems with proposed solutions concerning efficient SW development were collected from the literature. In addition, these experiences were collected during interviews in Platform R&D and Application R&D organizations. Due to above, it was possible to construct the XFRC process model. In consequence of constructing the XFRC process model (chapter 4.5), answers to research main question were found. Overall, exploiting the XFRC process model with recommended values (chapter 4.2), practices (chapter 4.3), roles and responsibilities (chapter 4.4) the SW development could be made more efficient than in current situation of Platform R&D and Application R&D organizations. More efficient SW development enables also faster release cycle that was initiated as research main question of this research. In conclusion, recommended XFRC values are stated as follows:

1. Adequate communication
2. Applicable commitment
3. Continuous and innovative learning
4. Dynamical resource allocation
5. Dynamical SW development
6. Dynamical SW process improvement
7. Minimum bureaucracy
8. Modularization
9. No parallel tasks

Recommended practices are stated as follows and other practices can be added according to the needs of organizations:

1. Incremental and iterative development
2. Small release programs
3. Meetings & workshops
4. Automatic and anticipatory testing
5. Dynamical integration
6. Pair working
7. Automatic and adequate documentation
8. Electronic bulletin board
9. Backlogs

Different roles by the XFRC process model can consist of mainly SW designers, SW architects and line manager. Depending on prioritization of tasks, designers in group could participate in CR work, MT, general SW work, maintenance work or FT during iterations. Also contact persons between groups are recommended and owners (e.g. Product Owners and Feature Owners) in different roles in Platform R&D and Application R&D organizations.

From perspective of appraisal of benefits it can be stated that the XFRC process model bring along many beneficial recommended values, practices, roles and responsibilities. These can be exploited through organizations where dynamic behavior and flexible adaptation for continuous changes regarding market needs are needed. In addition, if every group through R&D organizations commits itself to follow and exploit the XFRC process model in their daily work release cycle can become faster in the real world.

5. CONCLUSIONS

We can't solve problems by using the same kind of thinking we used when we created them.
Albert Einstein

Although agile methods alone are not necessarily enough for faster release cycle, the agile methods could be next step to an avenue of success. The literature review proved that e.g. Scrum and XP practices combining best practices of other agile methods could be suitable combination. Overall, many principles were found from the literature and these principles were discussed in literature review. The principles consist of overview to agile methods and related experiences of other corporations.

The literature review created basis for research of research problem related aspects. It also brought along answers to first research sub-question and related conclusions are discussed in chapter 5.1. The answers found during interview study were compared with the results from the literature review. The results from interview study consist of well-tried practices and quite many problems with proposed solutions in Platform R&D and Application R&D organizations. The results from interview study are based on interview results and collecting experiences of Scrum piloting. Due to interview study, the answers to second (chapter 5.2) and third research sub-questions (chapter 5.3) were found.

The answers to all research sub-questions were used as part for constructing the XFRC process model. Therefore, answers to research main question were found. Conclusions concerning answers to research main question are partially discussed in chapter 5.4. Detailed conclusions regarding the XFRC process model were already discussed in chapter 4. There is illustrated how thinking and attitude concerning overall SW development would be worthwhile to change through organizations so that discovered problems could be solved in the real world.

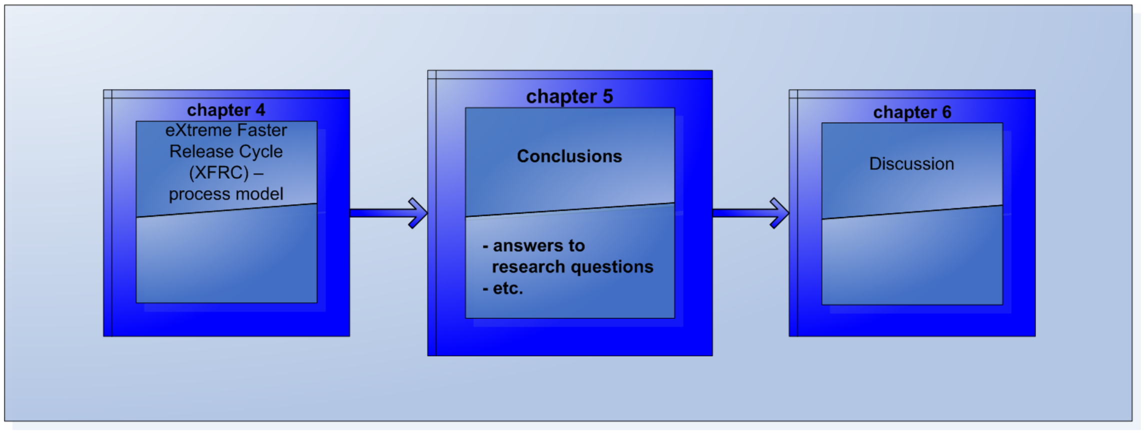

chapter 4
eXtreme Faster Release Cycle (XFRC) – process model
chapter 5
Conclusions
- answers to research questions
- etc.
chapter 6
Discussion

Figure 14. Introduction to chapter 5

5.1 Answers to first research sub-question

In consequence of the literature review (chapter 2), theoretical basis of agile SW development was created and answers to first research sub-question were found. First research sub-question was as follows:

Which are the basic principles for constructing the process model that enables faster release cycle?

Basic principles for constructing the XFRC process model that enables faster release cycle were found due to research overview to agile methods and related experiences of other corporations. In this research mainly XP and Scrum were selected for scientific study because many experiences from other corporations, which have piloted agile method(s) as among others both above methods for improving their SW process in their organizations, were available. In addition, Scrum was already under pilot process in one SW development group in Platform R&D organization. Furthermore, the hypothesis was that e.g. combination of XP and Scrum might be suitable for organizing of faster release cycle. In spite of emphasis of XP and Scrum, about few other agile methods were discussed too. Overall, many agile methods suggested practices were found and those seemed to be beneficial and applicable also for constructing XFRC process model. Also agile methods encouraged values that can be exploited for constructing the process model were discussed. Such values are e.g. openness, commitment, simplicity, feedback and courage.

Briefly, XP focuses mainly on engineering practices for agile SW development. Especially collective ownership, on-site customer, pair programming for mostly hard tasks, Planning Game, simple design, small release programs and TDD have been proven to be beneficial practices of XP. Also Spike-time for e.g. reading and learning about various issues, code reviews, coding standards, on-site coach, re-factoring, sitting arrangements, and "Zero Feature Iteration" practices were discussed in XP cases. Also those practices seemed to mainly be beneficial.

Scrum focuses mainly on managerial and organizational aspects regarding agile SW Development. Scrum seemed to make SW development process visible, controllable and manageable. Most beneficial practices of Scrum seemed to be freezing development scope for a month (sprint), backlogs, Use Cases and adaptable small teams.

Also XP@Scrum and adapted Cycles of Control framework were discussed in the literature review. Both of them consist primarily of above practices. Moreover, few other well-tried practices were discussed. Such practices seemed also to be communication model, reflection workshops for increasing team effectiveness, idea pool for continuous planning of product, prototyping for large features and Red-Flag for unexpected work. Automated and continuous testing (TDD included), continuous integration with e.g. daily builds, incremental and iterative way to develop SW, small release programs, small tasks and various meetings seemed to be common denominator to almost all cases that were discussed in the literature review.

Also problems concerning agile methods suggested practices had been noticed in almost all experiences of other corporations. Typical problems seemed to concern e.g., estimation process, lack of experience with some practices (e.g. TDD), outsourcing, parallel projects, planning workload and unclear common understanding of future release programs. However, many possible solutions for solving those problems were suggested in the discussed cases. According to the cases, corporations had also started some further actions for developing their SW development process in compliance with their piloting results.

All above aspects, values, practices and solutions for noticed problems are essential basic principles for constructing the XFRC process model. These principles were discussed in detail in the interview study when interview results were compared to the literature. Interview related conclusions are briefly discussed in next chapters 5.2 and 5.3. In addition, basic principles with entailed benefits are summarized in appendices 1, 8 and 11. Principles concerning XP practices are summarized in Appendix 1: XP practices. Furthermore, Scrum practices are summarized in Appendix 8: Scrum practices and other practices are summarized in Appendix 11: Other practices. In consequence of avoidance of extra repetition, detailed conclusions concerning basic principles were bypassed in this chapter.

5.2 Answers to second research sub-question

5.2.1 Well-tried practices

Before discussion about answers to second and third research sub-questions the things which appeared as well-tried practices by interviewees are discussed in this chapter. These well-tried practices were discussed in detail in the interview study in chapter 3.2 and used as part of organizing of faster release cycle. It means that most of them are used as proposed solutions for problems that were noticed in Platform R&D and Application R&D organizations. Problems with proposed solutions are discussed in next chapters.

The well-tried practices mean such practices that are not necessarily needed to change in Platform R&D and Application R&D organizations. From Platform R&D organization perspective e.g. split tasks, workshops and use of Scrum had been proven to be good practices. Also independent SW and FT groups seemed to be beneficial arrangements. On the other hand, the literature recommended that SW development and testing could be combined. Interviewees from Application R&D clarified that e.g. small groups, iterative development, code reviews and use of CVS had been good practices in their organization.

Situation regarding above well-tried practices might be partially similar concerning other R&D organizations that were excluded from the scope (chapter 1.4) of this research. Also situation concerning problems in other R&D organizations might be partially similar. On the other hand, also exactly same well-tried practices and problems have been noticed at least in few organizations as the literature review proved.

5.2.2 Discovered problems

In consequence of the interview study (chapter 3), answers to second research sub-question were found. Second research sub-question was as follows:

> *What kind of problem(s) pertain to ongoing product release program when some feature is added to that program in late phase and how to solve the possible problem(s)?*

Results from interview study appeared that a lot of improvements are needed to achieve adequate basis for organizing of faster release cycle. Concerning mainly situation when some feature is added to ongoing release program in late phase the many problems were discovered during interviews.

Discovered problems are described here:

1. Lot of parallel tasks in daily work
2. Lack of competence sharing
3. Bureaucratic decision making
4. Underestimation of needed resources
5. Inadequate co-operation between testing and SW groups
6. Lacking Feature Owner
7. Inadequate communication
8. Late changes to the content of release programs
9. Inconsistent synchronization of release programs
10. Heavyweight documentation and bureaucratic reviews
11. Lack of versioning knowledge of program blocks

All above problems cause that release cycle slows down. According to many interviewees, the biggest problem seemed to be many parallel tasks in SW designers' daily work. Amount of parallel tasks should be decreased through R&D organizations. Problem that concerns late changes to release programs seemed to be big problem too. Also e.g. increasing of communication and increasing of competence sharing through organizations were emphasized to be essential actions in the future. These are essential in order that enabling agile SW development although feature(s) would be added to release program in late phase. Along with agile SW development it could be possible to organize faster release cycle.

5.2.3 Proposed solutions

To solve all above problems that were described in chapter 5.2.2 many proposed solutions were collected from interviewees. In addition, proposed solutions were partially collected from the literature. Combined proposed solutions for above problems are described in detail in chapter 3.3. Because the results were already analyzed during the interview study proposed solutions are described only with few words in next sections. Assorted list of solutions proposed by interviewees vs. literature are unambiguously described in Appendix 10: Summarized problems with most essential proposed solutions.

Lot of parallel tasks in daily work -> decreasing of parallel tasks

For decreasing of parallel tasks one proposed solution was that dedicated persons for each task could be selected inside team. Dedicated persons could be named for SW work, CR work, maintenance etc. possible responsibilities. Decisions with relation to dedicated persons could be done by line management or team could decide their responsibilities themselves.

Names of dedicated persons could be described in a backlog, e.g. iteration specific backlog. Also freezing development scope for e.g. one month has been proven to be one solution to decrease parallel tasks. In addition, specific periods for e.g. planning and implementation etc. could be included to phases of iterations. Furthermore, arranging meetings before iterations seemed to be one good solution if there are many responsibilities in a team. Meetings could separately be arranged for e.g. CR work and fault correction. Due to above, e.g. synchronization of tasks could succeed and delays in ongoing tasks would decrease.

Lack of competence sharing -> increasing of competence sharing

For increasing of competence sharing job rotation could be one alternative. Job rotation could be done e.g. couple of time in a year in which case core competence would increase. Later on job rotation could be extended also to other groups. In this case designer would be in same group but task(s) could concern about other competencies from other group(s).

E.g. pair programming and pair coaching can be considered to partially be as job rotation when pairs are changed. When pairs are sitting in the same room it has been noticed to also be good solution from competence sharing perspective. Typical suggestion according to the results was that pair programming would be used mostly for difficult tasks as problem solving.

Also Use Cases were suggested to be solution to increase competence sharing. Use Cases could be used for planning e.g. FT and ST cases. Due to use of Use Cases for planning of tests, it would be possible to clarify how some CR related tests should be executed within different release programs. Therefore, use of TDD practices would probably succeed too. Use Cases seem to be suitable also for other phases in SW development. In addition, the results suggested few other practices whose use could increase competence sharing through organizations. Such practices are e.g. meetings & workshops, prototyping for large features, Planning Game, training sessions, Zero Feature Iteration and coding standards. E.g. during meetings, tacit and explicit knowledge can be found and perceive better due to knowledge sharing. In consequence of all above agile practices, competence could be increased as knowledge increases.

Bureaucratic decision making -> decreasing bureaucracy

Decision making regarding approval/rejection of CRs should be faster than in current situation. According to the results, decision making could be done by low-level management.

However, it requires that overall effects on a product are known from low-level management perspective too. Adequate communication and competence sharing could increase knowledge of these effects. Therefore, line managers could make decisions concerning CRs at least in case where effects are concentrated on own or parallel SW group.

Bureaucracy could also be decreased with relation to reviews and possibility to e.g. competence sharing should be allowed by management boards. Furthermore, minimum bureaucracy seems to be beneficial aspect when level 2 or higher of CMM is decided to achieve.

Underestimation of needed resources -> continuous resource allocation and prioritizing of tasks etc.

At least 20% of total effort of release programs should be reserved for CR work. In addition, at least 20% of total effort should be reserved for ST personnel for possibility to e.g. reading specification documents or participation in possible meetings as reviews. Time for reading and learning about various issues could be allocated from end of ongoing release program or from the beginning of next program. Also amount of unexpected work should be noticed when estimations are planned.

Resource allocation should be continuous, risks should be noticed and prioritization of the most important features should be done carefully when release programs and projects during them are started. Prioritization should be done e.g. at start of iterations. Thus, some new feature could be added to release program even if in late phase.

Needed resources could be available dynamically and due to right prioritization of tasks. The results also suggested that e.g. Portfolio Management, Product Roadmap and Release Backlog for upcoming release programs seem to be suitable practices from estimation planning perspective. Furthermore, e.g. collective ownership, fixed time and scope concerning changes to release program content, demo-sessions, prototyping, meetings and small release programs seem to be beneficial for making estimation easier according to the results. Moreover, experiences concerning estimations become better after each sprint.

Inadequate co-operation between testing and SW groups -> sitting arrangements and increasing of co-operation

The results discussed about individual SW and (MT & FT) testing groups vs. combined SW and testing groups. Co-operation between them and beyond through ST groups was recommended in order that e.g. testing support and fast feedback would be enabled. Therefore, designers and testing personnel could sit near each other, e.g. in same floor. Also meetings &

workshops could be arranged. In addition, Use Cases could be used for planning test cases as in discussion about competence sharing was described. Above arrangements would increase co-operation between testing and SW groups in R&D organizations.

Increased co-operation makes also competence sharing between above groups possible. Due to competence sharing it would be possible that designers could execute e.g. FT cases if needed. Competence sharing could be succeeded if designer and testing personnel are working as pair, i.e. some kind of pair testing in this case. Also organizing of testing with e.g. TDD practices would be beneficial. Furthermore, continuous integration with suitable version control system (e.g. CVS or Subversion) and continuous automated tests with automatically generated test reports could make testing and release cycle faster. It takes time to automate tests but automated tests are quite fast to execute during e.g. regression testing.

Lacking Feature Owner(s) -> named Feature Owner(s)

Feature Owner could own and coordinate a feature through its lifecycle, also during maintenance phase. Therefore, needed information concerning some feature would be available immediately if named Feature Owner has good core competence. Designers already own part(s) of feature(s) in program block level. There could also be one owner in every SW group and that owner would own part of some feature in group level. Owners in group level could co-operate with each other and communicate with Feature Owner. Also meetings between different participants could be arranged if needed. During those meetings, possible problems etc. could be discussed. In consequence of above, delays regarding availability of feature related information and finding right responsible person of some feature would not necessarily cause slow down of release cycle in the future.

Inadequate communication -> increasing of communication through organizations

Communication is in major role in agile methods. Multi-site organization causes communication problems especially in case when there are split tasks through multi-site. The results suggested that specified tasks would be done in same building or at least in same country. In consequence of above, communication is easier and faster when distance between communicating persons is not too long. Also meetings & workshops increase communication. E.g. problem solving and planning all manner of things seem to usually be easier and faster during meetings than e.g. by mails. On the other hand, if e.g. meetings would be arranged too often so time does not remain enough for real work, albeit meetings are part of actual work too. However, best benefits of meetings could be achieved if persons who have knowledge about the subject of the meeting would be invited and they would participate in those meetings.

In addition, the results proved that due to use of agile practices increased communication can be achieved. E.g. communication model, on-site customer and backlogs are aspects that make communication more effective. Backlogs could increase visibility of coming CRs at least from designers' and testing personnel's perspective. Communication model and compliance with it could minimize same questions by different management boards in matrix organization. Customers could participate in meetings and they could discuss with managers from different boards about content of release program. Therefore, communication could be increased through different management boards and customers. On the other hand, amount of customers is big and resources in R&D organizations are limited. Consequently, arranging meetings with representatives from customers' organizations is not necessarily appropriate alternative in the real world.

Late changes to the content of release programs -> freezing content of release programs for fixed or dynamical length time

Late changes to the content of release programs cause that release content is typically too large and scheduling must be re-planned. Furthermore, major changes cause delays in ongoing release programs and next release programs are delayed too. According to the results, major changes to content of release program should not be done in late phase. It means that new feature should not be added to ongoing release program after E1 milestone. Only fault corrections could be taken into release program in late phase.

However, if any changes to content are done in late phase the results suggested that e.g. freezing content of release programs for fixed time could be beneficial. Small release programs combined with iterative and incremental SW development enable it in practice. Iteration such as e.g. one month sprint is one example of fixed time practice. It seemed to be good solution according to the results because it seems to make possible adding and (re)moving of features also in late phase of release programs. In this case, some high prioritized feature could be taken into coming iteration but not into ongoing iteration. In addition, availability of competent resources should be taken into consideration when some feature is planned to add into coming iteration. For example, if some SW designer is working with low priority fault correction so designer could interrupt fault correction and start working with e.g. new feature when next sprint is started. On the other hand, iterations could be adjusted dynamically depending on planned work amount for iteration. Consequently, length of iterations could be adjusted to vary from e.g. few days to few weeks. If planned work amount concerning the iteration has been done e.g. after few weeks a next iteration could be started after planning and prioritizing assignments.

To manage content of release program and to plan upcoming release programs Product Roadmap and the backlogs seemed to be beneficial. Product Roadmap seemed to be useful for specifying of all planned release programs that are scheduled into it. Features can be described in e.g. Product Backlog. It seemed to be beneficial to collect ideas concerning features too. The features can be split and described in Release Backlog that seemed to be useful for specifying of content of release programs. New features can be added to the Release Backlog, meaning that other features can be excluded due to possible re-prioritization. Each feature in Release Backlog can generate several tasks to Sprint Backlog for each sprint. E.g. planning, evaluation and estimation of work can be done during different meetings as the results proved.

Inconsistent synchronization of release programs -> consistent synchronization

A lot of improvements are needed that release programs could be synchronized and due to this release cycle could be made faster also from synchronization perspective. At least module tested SW could be released incrementally by Platform R&D for Application R&D needs. Application R&D could test their SW with Platform SW and communicate possible changes to Platform R&D in early phase. At the same time SW work would continue in both R&D organizations and after possible changes the SW would be tested by Platform R&D also in FT phase. Iterations could contain phases from IS to FT or at least from IS to MT. Iterative and incremental way to develop and release SW for Application R&D needs enable to react to possible changes in early phase as above was mentioned.

Also Simulation Game with prototyping, prioritized feature list (e.g. backlog), meetings & workshops, continuous integration and small release programs seemed to be beneficial practices to synchronize release programs. In addition, on-site customer practice could be applicable when persons from Application R&D act as a representative concerning requirements for Platform R&D. Representatives from both organizations could participate in e.g. prioritizing of features during meetings. Furthermore, customers could act as an on-site customer concerning requirements for Application R&D if customers know what they really want. Therefore, features related decisions etc. could be done by participants from related organizations.

Heavyweight documentation and bureaucratic reviews -> lightweight documentation

Lightweight documentation and split tasks to small increments concerning documentation seemed to be good practices in order that maintaining different documents would not take too much time. Specific design details can be included into code and also some temporary work documents could be used. Due to that, e.g. SD documentation could be decreased. Furthermore, IS documentation could contain FT scenarios that make possible to start planning FT cases in early phase as TDD encourages. TDD could be used also in other test phases. To enable that updating for documentation would be fast enough also feedback through organizations should be fast. Above incremental way to split documents to small increments supports that perspective too. Iterations make possible to change documentation faster than in current situation.

As the results suggested there is too bureaucratic reviews in Platform R&D and Application R&D organizations. Therefore, formal inspections are not needed for minor changes but reviews are enough so that documentation review process would be faster. Also continuous pair reviewing could be used when minor changes are reviewed. Formal inspections concerning documentation are needed only for major changes. When inspections and reviews are arranged participation of invited persons in those sessions is essential so that possible unclear things could be discussed. Especially reviewing IS documentation is essential according to the results. It requires commitment and time reservation to participate. To enable more active participation in above sessions the Spike-time and Red-Flag practices seemed to be beneficial alternatives in practice. On the other hand, reviewing of IS documentation could be done incrementally by pair or few person e.g. weekly whereupon reviewed material would not be too large. In addition, possible faults could be noticed as early as possible.

Lack of versioning knowledge of program blocks -> increasing knowledge of versioning

High cohesion and low coupling should be noticed and learned within SW development. Therefore, better maintainability and reuse of SW components could be achieved as basic rules of SW development recommend. If many designers are working with same program block the CVS and Subversion seemed to be good solutions concerning continuous integration of tested code incrementally and iteratively. CVS could be used for versioning before freezing of code modules and to integrate code from development branch(es) to main branch. On the other hand, if compatibility of program blocks is assured, high cohesion and low coupling are used so many development branches are probably not needed. Subversion could replace freezing of code modules into separated databases at least partially. In addition, Subversion seems to enable continuous integration better than CVS.

5.2.4 Dependencies between proposed solutions

Dependencies between proposed solutions have been appraised according to the results. These dependencies are based on interpretations of the results. Due to interpretations, the dependencies could be appeared as Figure 15 shows:

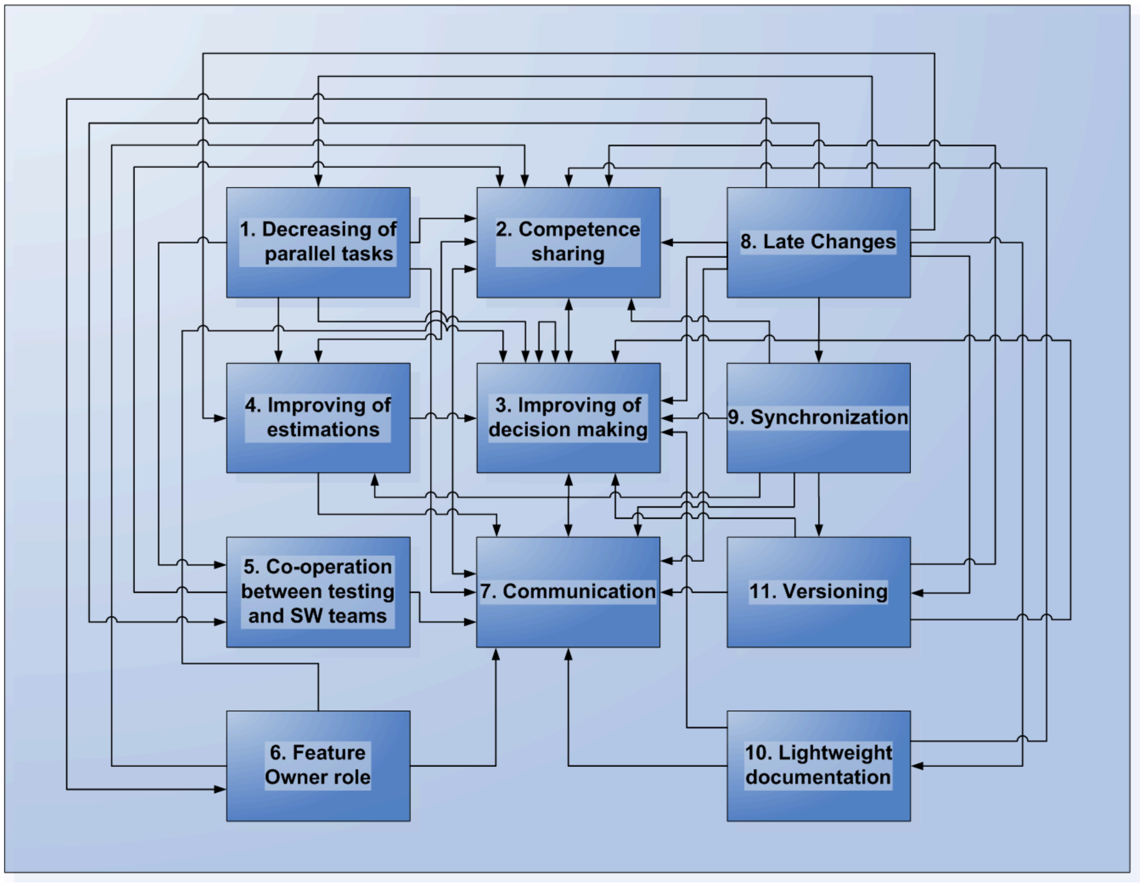

1. Decreasing of parallel tasks
2. Competence sharing
8. Late Changes
4. Improving of estimations
3. Improving of decision making
9. Synchronization
5. Co-operation between testing and SW teams
7. Communication
11. Versioning
6. Feature Owner role
10. Lightweight documentation

Figure 15. Interpreted dependencies between proposed solutions

As above figure shows the arrows point prerequisites that are required for enabling an avenue to success concerning starting point. For example, if arrow from starting point has been drawn to point some solution so last mentioned solution has been interpreted to be as required solution for enabling source solution. According to interpretations, at least following seem to be essential solutions as well as targets for development that affect almost all other solutions and enable them in practice i) decreasing of parallel tasks, ii) increasing of communication, iii) increasing of competence sharing and iv) improving of decision making. Exactly all proposed solutions should be taken into consideration to enable organizing of faster release cycle. E.g. forecasting of late changes to release programs seem to require taking all proposed solutions into consideration so that forecasting would be successful. Late changes are necessarily not required by enabling other proposed solutions that are illustrated in Figure 15. However, at least customers require those changes in practice.

Although dependencies are based on mainly interpretations there are described few dependencies based on suggestions by interviewees. Such dependencies were noticed concerning e.g. competence sharing, decreasing of parallel tasks and forecasting of late changes to content of release programs. Competence sharing is needed that dedicated persons could be named and in consequence of that parallel tasks could be forecasted. Therefore, at least forecasting of late changes requires decreasing of parallel tasks. On the other hand, decreasing of parallel tasks is probably required concerning many other tasks in order that some high prioritized task could be executed.

According to interpretations, to enable decreasing of parallel tasks many other actions are required, e.g. improving of estimations and decision making. Without decision making actually anything can not improved. Also improving of decision making requires itself in order that decision for improving decision making could be done in practice. E.g. improving of communication requires decision making that as well requires improving of communication and so on. Other interpretations by reader are certainly possible.

Although there seem to be many dependencies between proposed solutions all proposed solutions can not be implemented immediately in R&D organizations. Instead improvements could be done incrementally. For example, above mentioned decreasing of parallel tasks requires at least i) increasing of competence sharing, ii) improving of decision making, iii) improving of estimation planning iv) increasing of co-operation between groups and v) increasing of communication. At first, e.g. communication could be increased. If increasing communication seems to be difficult so competence sharing regarding successful communication could be started. When communication is succeeded without major problems competence sharing should be increased. It requires at least decision making etc. that competence could be really shared e.g. inside group. Later on competence could be expanded to parallel groups and e.g. to testing groups and vice versa. Also estimations for release programs should be planned thereby that there would be enough time for increasing of competence sharing in this case. Suitable effort could be e.g. 20% of total time of release programs as the results suggested. Also estimation planning requires competence sharing and decision making and so on. Above mentioned things could be first increment to an avenue of success. Next steps could be specified according to R&D organizations own needs.

5.3 Answers to third research sub-question

Besides answers to second research sub-question, answers to third research sub-question were found during interview study (chapter 3). Third research sub-question was as follows:

What kind of problem(s) pertain to ongoing product release program when some feature is (re)moved from that program in late phase and how to solve the possible problem(s)?

Interview results that relate to (re)moving some feature from ongoing release program in late phase seemed not to be so big problem than adding some feature. However, same problems seem to effect in both cases. Also proposed solutions for solving these problems that are related to (re)moving case are mainly equal to case when adding feature to release program in late phase.

Most essential differences between adding and (re)moving cases are just actions and focus of multiplicative effects. For example, when designer's parallel tasks probably temporarily decrease due to (re)moving some feature from release program it causes more parallel tasks to managers in matrix organization(s). From designer's perspective it causes mainly frustration and only minor changes to e.g. documentation and versioning. On the other hand, removing of feature is infrequent. Moving feature to next release program seemed to be more general according to designers.

In (re)moving case, management have to change estimates, plan resources for coming tasks, plan prioritization of tasks and content of release programs. When some feature is removed testing personnel close related tests for removed feature and remaining functionality is tested in order that compatibility can be ensured. When some feature is moved to next release program the ongoing tasks can be continued.

According to interviewees, mainly above problems were noticed concerning (re)moving case. Moreover, in general terms the solutions that were proposed to solve adding case related problems seem to be applicable to solve (re)moving case related problems. Therefore, detailed discussion about above and other (minor) problems that concern (re)moving case is bypassed.

5.4 Answers to research main question

Due to finding answers to above research sub-questions, finding answers to research main question was possible. Needed answers were found and finding these answers enabled constructing the XFRC process model in practice. Research main question was as follows:

What kind of SW process model would enable the faster release cycle when content of on-going release programs is continuously changing in late phase?

Answers to research main question consist of the XFRC process model with recommended set of values, practices, roles and responsibilities. These all aspects concerning the XFRC process model are illustrated in chapter 4. Therefore, detailed discussion about above is bypassed in this chapter and only other conclusions concerning the XFRC process model are discussed.

The XFRC process model is similar to other process models such as suggested by XP and Scrum agile methods. It contains also similar and same aspects than other process models. Briefly, it is one combination of few agile methods recommended values and practices etc. In addition, the XFRC process model contains combination of suggestions that were found during interview study.

E.g. XP and Scrum contain some similarities although XP consist of mainly engineering aspect and Scrum contains mainly managerial aspect. The XFRC process model has mainly been constructed to be combination of above aspects that have already been proven to be beneficial practices according to the literature. Also aspects concerning few problematical practices have been noticed. These problematical practices mean use of such practices that were noticed to cause few problems according to the literature review. Many problems were discussed during the interview study too. Problems noticed by interviewees are not in direct relation to agile methods. However, with agile methods and proposed solutions by interviewees and the literature these problems can be solved or at least consideration of those possible problems can be done. Therefore, combination of discovered beneficial aspects and consideration of possible problems proved to be good solution for constructing the XFRC process model within this research.

Actually, the XFRC process model does not solve all problems concerning more efficient SW development. In spite of that, it enables more efficient SW development without waste time on trivia. Due to efficient SW development, it describes alternative approach to enable faster release cycle. Furthermore, it could be dynamically scaled and exploited depending on R&D organizations needs. Scalability requires, indeed, e.g. commitment.

6. DISCUSSION

That is what learning is. You suddenly understand something you've understood all your life, but in a new way.

Doris Lessing

This chapter collects aspects concerning this research. Firstly, learning experiences are discussed in chapter 6.1. Learning experiences describe what went well and what could have been done better during this research.

Reliability of research describe that if almost same results have been achieved after collection of data by iterations reliability can be confirmed to be good. Validity refers to whether the research questions that were selected in the beginning have been answered. Briefly, above aspects describe how the aims of research were achieved concerning this research. These aspects are discussed in chapter 6.2.

Further actions for organizing of faster release cycle are suggested in chapter 6.3. Finally, suggestions for further research are discussed in chapter 6.4.

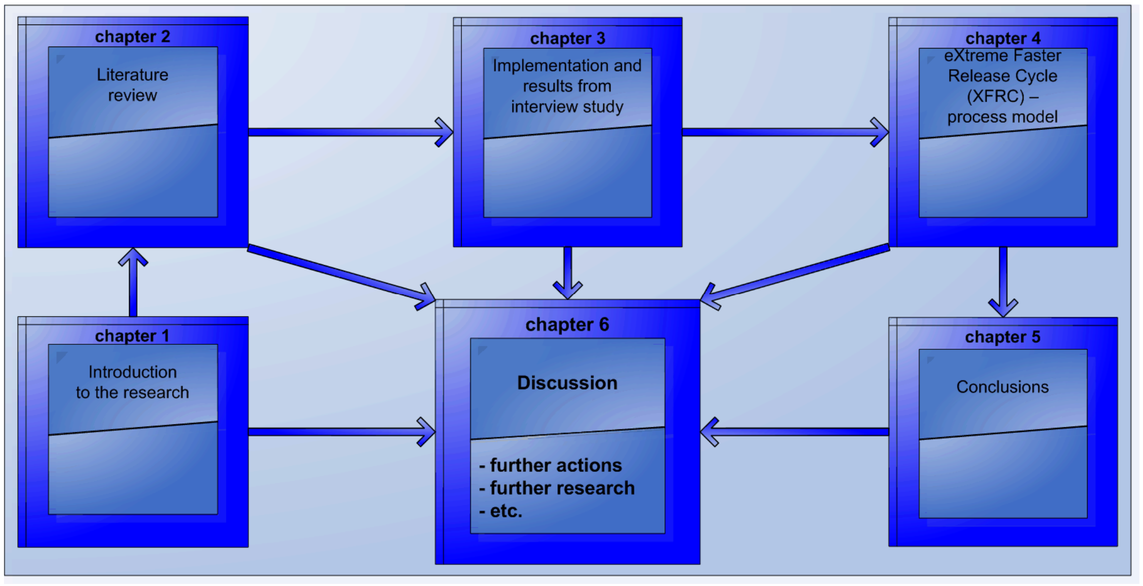


Figure 16. Introduction to chapter 6

6.1 Learning experiences

Research in R&D organizations was fascinating and it also seemed to provide reliable way to solve problems. During this research many new things were learned. One of them was use of constructive research approach in practice. Also knowledge of agile methods advanced and understanding of interview method became deeper as it was utilized in this research.

A fewer amount of interviewees would have been enough because saturation of research was achieved before latest interviews. Also e.g. query could have been suitable for collecting of more research data instead or in addition to many interviews.

Surprisingly lot of discussion arose also about many other issues than just research problem related issues during interview sessions. Thus, interview method seems to be practical for collecting of tangential data besides research problem related data. Probably as much data would not have been able to collect via queries because people do not always want to write long answers within queries and asking of further questions is not so easy within queries than within interviews.

6.2 Reliability and validity of research

Reliability means that almost same results are achieved after collection of data by iterations, i.e. if similar results are achieved for a research with same (measured) characteristics then reliability is good (Järvenpää & Kosonen 2003, pp. 28-30). In this research the literature used for literature review is mostly reliable because the literature has been approved by different committees. In addition, the experiences of corporations were collected concerning different industries, not only corporations in same industry. On the other hand, it is possible that all the problems are not mentioned in the literature which was used to collect experiences of other corporations. However, maybe at least most of (major) problems were mentioned. Furthermore, only "experiences concerning use of agile methods in large organizations" could have been searched. On the other hand, experiences concerning mainly small and medium-sized organizations can be scaled and adapted to large organizations too. The literature seems to contain also claims that some practices have been succeeded and e.g. success of scalability concerning use of agile methods is obviousness. Probably e.g. scalability is possible in small organizations but it is not necessarily easy to adapt through large organizations. However, it is possible that more researches will be done in the future and possible scalability problems will be solved in practice.

Meanwhile interview results can be addressed to be reliable only partially, because it is possible that many people have not actually same opinions than participants of this research. However, most results are similar or in accordance with the literature review results. If interviews would have been arranged through more widely area (i.e. development sites in other countries) so results could have been different because there are different cultures in R&D organizations. In addition, larger amount of interviewees could have produced more different opinions than sampling in this research. Same situation is possible if similar research will be arranged in the future and interviewees would participate from different groups in multi-site organizations. On the other hand, also query could have been dedicated through organizations and it would have been more suitable alternative than increasing the amount of interviewees because the organizations include hundreds of people and answering to queries would probably have been produced enough results from this research aspect.

For example, when interviewee was authorized representative of later phase of SW process so interviewee's general opinion was that previous phases of SW process should be completed before own phase. E.g. ST personnel's opinions were that faults should be found in FT phase and the FT phase should be completed 100% before ST phase and so on. Also adding and removing of features seemed to be possible before own phase but not generally within own phase. Additionally it can not be assumed that every people have same opinion in organizations, although few people would have same opinion about some specified thing – as mentioned above. Thus, reliability of this research seems to be good only partially. In spite of that, the results suit for use in organizations discussed in this research and probably in many other organizations as well.

Validity refers to whether the research questions that were selected in the beginning have been answered (Järvenpää & Kosonen 2003, pp. 30-32). In this research validity seems to be good, because needed answers to research main question and sub-questions were found. Due to above, research problem was solved as aimed at the beginning of this research.

6.3 Suggestions for Further Actions

The XFRC process model has been constructed during this research. It enables efficient approach to develop SW in R&D organizations. Due to above, it enables organizing of faster release cycle. In addition, it is possible to scale the XFRC process model from small product release programs to large product release programs. In addition, projects that are included in release programs are scalable too. Therefore, this process model can be used in all R&D organizations and in other corporations too. The construction might be incipience of a golden opportunity to elaborate appropriate dynamically scaled SW process for many corporations regardless of line of business.

The construction for organizing probably brings along many benefits for R&D organizations if results of possible piloting of the XFRC process model will be beneficial as expected. As incremental development way encourages, thus also the process model can be split to increments and parts of model which might be classified into highest priority group in R&D organizations could be taken into further actions. E.g. increasing of communication, increasing of competence sharing and decreasing of parallel tasks could be implemented at first and the rest of other increments of the process model could be implemented later. If the XFRC process model is taken into consideration during SW Process Improvement (SPI) activities it might cause resistance concerning changes. Furthermore, the factors such as improvement tactics, process complexity, volume of initiative, organizational culture and individual skills affect SPI outcome (Börjesson 2004).

However, e.g. organizing and SW development methods related changes are essential to improve SW process for continuous changes regarding market needs. As initially was discussed it requires dynamical behavior and flexible adaptation for above continuous changes regarding market needs in contemporary growing global market area. Improving processes for integrating performance, competence and knowledge management, managers will be able to find more answers to e.g. organizing related questions and more, clearing the path to the ultimate intelligent organization (Sydänmaanlakka 2002).

If corporation’s mission and vision contain e.g. SPI it is possible to improve process that would be suitable from corporation and customers perspective. SPI takes time and money in the short and of course also in the long term, but it can also be beneficial, fascinating and funny, and bring money to corporation in the future.

One possible point for further actions within SPI could be e.g. IDEAL-model (*Initiating, Diagnosing, Establishing, Acting* and *Learning*) that has successfully been used e.g. at Ericsson corporation (Börjesson 2004). The IDEAL-model (McFeeley 1996) that was developed by Software Engineering Institute (SEI) is illustrated in Figure 17 on next page. Also detailed research results how to enable SPI in organizations would be beneficial to learn as part of further actions. Salo (2006) has researched above perspective at VTT.

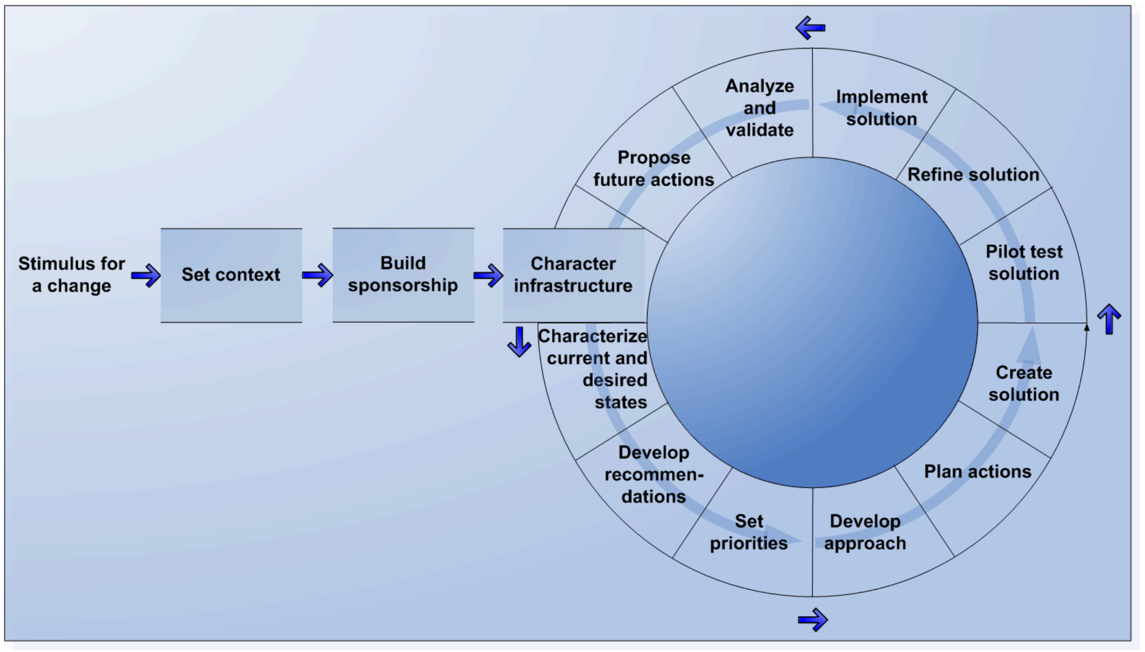


Figure 17. IDEAL model

6.4 Suggestions for Further Research

Besides the selected research problem, also many other improvement needs, opportunities and proposals came up during the research. Organizing problem was only one of them. E.g. prioritizing of customer requirements (in that case customers would be “users” from usability perspective) and improving of product management would be other issues. Organizing related issues with SW development aspect were selected for this research because it seemed to be relevant alternative to both organization and the author of this thesis. Other issues need further research that might be done in the future. Other improvement needs, opportunities and proposals that have been collected during interviews have been archived to corporation’s internal database for confidentiality reasons. Thus, also decision making for possible further research and other issues related details will be discussed outside this research.

When other issues are decided to research or solving of e.g. organizing problem is decided to continue with details one alternative would be to arrange further research for collection of more detailed data. From literature review perspective the different research results are found from e.g. books and conference articles. Because scope of this research was limited to concern mainly XP and Scrum the other agile methods and suitability of them could be researched in the future. E.g. FDD and many other agile methods (chapter 2.1.4) could be taken under detailed research. Also Lean (Poppendieck et al. 2006) SW development could be taken into consideration during possible research. In addition, applicability of e.g. Acceptance Test-Driven Development (ATDD) (Koskela 2007) could be researched. ATDD extends TDD to the overall software development process and recommends that tests for features should be implemented first (Koskela 2007).

Agile methods will probably be developed in the future. Also new agile methods will probably created even increasingly in the future. These methods can be adapted and taken into use through R&D organizations so that SW development could be made more effective and release cycle could become faster. In addition, new experiences and learning portals will probably be available in the future and these experiences could be exploited in R&D organizations. One of those learning portals has been published by VTT and partners in co-operation (Abrahamsson et al. 2007).

From query perspective, focused query could be good solution and it could contain e.g. multiple choices of typical answers that were got in the interviews. The query could be focused to both Platform R&D and Application R&D organizations so more detailed data and opinions would be collected through both multi-site organizations. Finally, due to all suggestions that were described in this thesis, release cycle can be organized to become faster than in current situation in R&D organizations.

LIST OF REFERENCES

Aarnio, L. 2001. *Automated testing as a part of extreme programming methodology*. Helsinki: University of Helsinki, Department of Computer Science. Master's Thesis.

Abrahamsson, P., Warsta, J., Siponen, M. T., and Ronkainen, J. 2003. *New directions on agile methods: A comparative analysis*. Proceedings of the 25th International Conference on Software Engineering (ICSE '03). Oulu: Technical Research Centre of Finland, VTT Electronics. pp. 244-254.

Abrahamsson, P., Hanhineva, A., Hulkko, H., Ihme, T., Jäälinoja, J., Korkala, M., Koskela, J., Kyllönen, P. & Salo, O. 2004. *Mobile-D: An Agile Approach for Mobile Application Development*. Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '04). NY: ACM Press. pp. 174-175.

Abrahamsson, P. et al. 2007. *House of Agile, www-document*. [Cited: 9.12.2007]. Available: http://www.houseofagile.org/

Aho, S. 2006. *Improving the Software Development Process of a Research and Development Team*. Tampere: Tampere University of Technology, Department of Information Technology. 69 p. Master's Thesis.

Ambler, S.W. 2002. *Agile modeling: effective practices for eXtreme Programming and the unified process*. NY: Wiley. 384 p. ISBN: 0-471-20282-7

Ambler, S.W. 2007. *Agile Modeling home Page, www-document*. [Cited: 27.11.2007]. Available: http://www.agilemodeling.com/

Auvinen, J., Back, R., Heidenberg, J., Hirkman, P. & Milovanov, L. 2005. *Improving the Engineering Process Area at Ericsson with Agile Practices – A case study*. Turku: Turku Centre for Computer Science (TUCS), Laboratory of Data Mining and Knowledge Management Software Construction. 24 p. ISBN: 952-12-1616-6

Baskerville, R. & Ramesh, B. 2003. *Is "Internet-speed" software development different?*. Proceedings of the Institute of Electrical and Electronics Engineers (IEEE) Software, Vol. 20, No. 6. pp. 70-77.

Beck, K. 1999. *Embracing Change with Extreme Programming*. Proceedings of the IEEE Computer, Vol. 32, No. 10. pp. 70-77.

Beck, K. 2000. *Extreme Programming Explained: Embrace Change*. MA: Wesley. 190 p. ISBN: 0-20-161641-6

Boelsterli, B. S. 2003. *Iteration Advocate. Small sampling of New agile Techniques Used at a Major Telecommunications Firm*. Proceedings of the Agile Development Conference (ADC '03). CO: WaveFront. pp. 109-113.

Börjesson, A. 2004. *Successful Process Implementation*. Proceedings of the IEEE Software, Vol. 21, No. 4. pp. 36-44.

Cockburn, A. 2004. *Crystal Clear: A Human-Powered Methodology for Small Teams*. MA: Wesley. 336 p. ISBN: 0-201-69947-8

Cusumano, M. A. & Selby, R. W. 1997. *How Microsoft builds software*. NY: ACM Press. pp. 53-61.

Cusumano, M. A. & Yoffie, D. B. 1999. *Software development on Internet time*. Proceedings of the IEEE Computer, Vol. 32, No. 10. pp. 60-69.

Derbier, G. 2003. *Agile Development in the old economy*. Proceedings of the Agile Development Conference (ADC '03). pp. 125-131.

Grossman, F., Bergin, J., Leip, D., Merritt, S. & Gotel, O. 2004. *One XP Experience: Introducing Agile (XP) software Development into a Culture that is willing but not ready*. Proceedings of the conference of the Centre for Advanced Studies on Collaborative research. Canada: IBM Press. pp. 1-13.

Haikala, I. & Märijärvi, J. 1997. *Ohjelmistotuotanto*. Jyväskylä: Gummerus. 357 p. ISBN: 951-762-497-2

Hedin, G., Bendix, L. & Magnusson, B. 2003. *Introducing Software Engineering by means of eXtreme Programming*. Proceedings of the 25th International Conference on Software Engineering (ICSE '03). Lund: Lund Institute of Technology. pp. 586-593.

Highsmith, J. A. 2000. *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. NY: Dorset House. 358 p. ISBN: 0-932633-40-4

Hirsch, M. 2005. *Making RUP Agile*. Proceedings of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA '02). NY: ACM Press. pp. 1-8.

Huczynski, A. & Buchanan, D. 2001. *Organizational Behaviour – An Introductory text*. London: Prentice-Hall. 916 p. ISBN: 0-273-65102-1

Hunt, A. & Thomas, D. 2000. *The Pragmatic Programmer: From journeyman to Master*. MA: Wesley. 352 p. ISBN: 0-201-616222-X

ISO 9241-11. 1998. *Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability*. Switzerland: International Organization for Standardization. 26 p.

Itkonen, J., Sulonen, P. & Vanhanen, J. 2003. *Improving the Interface Between Business and Product Development Using Agile Practices and the Cycles of Control Framework*. Proceedings of the Agile Development Conference (ADC '03). Espoo: Helsinki University of Technology, Department of Computer Science. pp. 71-80.

Järvenpää, E. & Kosonen, K. 2003. *Johdatus tutkimusmenetelmiin ja tutkimuksen tekemiseen*. Espoo: Otamedia Oy. 101 p. ISBN: 951-22-3321-5

Kasanen, E., Lukka, K. & Siitonen, A. 1993. *The Constructive Approach in Management Accounting research*. Journal of Management Accounting Research, Fall.

Koskela, L. 2007. *Test Driven - TDD and Acceptance TDD for Java developers*. Greenwich: Manning Publications Co. 544 p. ISBN: 1-932394-85-0

Larman, C. & Basili, V. R. 2003. *Iterative and incremental developments: A brief history*. Proceedings of the IEEE Computer, Vol. 36, No. 6. pp. 47-56.

Mahnic, V. & Drnovscek, S. 2005. *Agile Software Project Management with Scrum*. Proceedings of the 11th International Conference of European University Information Systems (EUNIS). Manchester: University of Manchester. pp. 1-6.

Mason, M. 2004. *Subversion for CVS users, www-document*. [Cited: 9.12.2007]. Available: http://osdir.com/Article203.phtml

Maximilien, E.M. & Williams, L. 2003. *Assessing Test-Driven Development at IBM*. Proceedings of the 25th International Conference. NC: IBM and North Carolina State University. pp. 564-569.

McFeeley R. 1996. *The Ideal Model, www-document*. [Cited: 9.12.2007]. Available: http://www.sei.cmu.edu/ideal

Mugridge, R. 2003. *Test Driven Development and the Scientific Method*. Proceedings of the Agile Development Conference (ADC ’03). New Zealand: University of Auckland, Department of Computer Science. pp. 47-52.

Nonaka, I. & Takeuchi, H. 1995. *The Knowledge Creating Company: How Japanese Companies Create the Dynamic of Innovation*. Oxford: Oxford University.

Paetsch, F., Eberlein, A. & Maurer, F. 2003. *Requirements Engineering and Agile Software Development*. Proceedings of the 12th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2003), Linz, Austria. Calgary: University of Calgary. pp. 308 -313

Palmer, S. & Felsing, J. 2002. *A practical guide to Feature-Driven Development*. NJ: Prentice Hall. 271 p. ISBN: 0-13-067615-2

Poppendieck, M. & Poppendieck, T. 2006. *Implementing Lean Software Development*: *From Concept to Cash*. MA: Wesley. 304 p. ISBN: 978-0-321-43738-9

Rautiainen, K., Vuornos, L. & Lassenius, C. 2003. *An Experience in Combining Flexibility and Control in a Small Company’s Software Product Development Process*. Proceedings of the International Symposium on Empirical Software Engineering (ISESE ’03). Rome, Italy. Espoo: Helsinki University of Technology, Department of Computer Science. pp. 28-37.

Rautiainen, K. & and Lassenius, C. 2001. *Pacing Software Product Development: A Framework and Practical Implementation Guidelines*. Espoo: Helsinki University of Technology, Software Business and Engineering Institute.17 p.

Raynus, J. 2002. *Software Process Improvement with CMM*. Artec House. 222 p. ISBN: 1-580-53474-0

Rising, L. & Janoff, N. S. 2000. *The Scrum Software Development Process for Small Teams*. Proceedings of the IEEE Software. AZ: AG Communication Systems. pp. 26-32.

Royce, W. 1970. *Managing the Development of Large Software Systems*. Proceedings of the IEEE CS Press (Reprinted from proceedings of the IEEE WESCON, August 1970, pp. 1-9). pp. 328-339.

Salo, O. 2006. *Enabling Software Process Improvement in Agile Software Development Teams and Organisations*. Oulu: University of Oulu and VTT Technical Research Centre of Finland. Dissertation. 149 p. ISBN 951-38-6869-9

Schatz, B. & Abdelshafi, I. 2005. Primavera Gets Agile: A Successful Transition to Agile Development. Proceedings of the IEEE Software, Vol. 22, No. 3. pp. 36-42.

Schoonmaker, S. J. 1997. *ISO 9001 for engineers and designers*. NY: McGraw. 238 p. ISBN: 0-070-57710-2

Schwaber, K. 2003. *Agile project management with SCRUM*. WA: Microsoft Press. 163 p. ISBN: 0-7356-1993-X

Schwaber, K. 2007a. *Scrum home page, www-document*. [Cited: 21.1.2007]. Available: http://www.controlchaos.com/

Schwaber, K. 2007b. *XP@Scrum home page, www-document*. [Cited: 21.1.2007]. Available: http://www.controlchaos.com/about/xp.php

Schwaber, K. & Beedle, M. 2002. *Agile software development with SCRUM*. NJ: Prentice Hall. 158 p. ISBN: 0-13-067634-9

Stapleton, J. 1997. *Dynamic Systems Development Method: The method in practice*. MA: Wesley. 192 p. ISBN: 0-201-17889-3

Sydänmaanlakka, P. 2002. *An Intelligent Organization: integrating performance, competence and knowledge management*. Oxford: Capstone. 234 p. ISBN: 1-84112-048-0

Taskinen, L. T. 2002. *Measuring Change Management in Manufacturing Processes - A Measurement Method for Simulation Game based Process Development.* Espoo: Helsinki University of Technology, Department of Industrial Engineering and Management. Dissertation. 254 p. ISBN: 951-38-6381-6

Vriens, C. 2003. *Certifying for CMM Level 2 and ISO 9001 with XP@Scrum*. Proceedings of the Agile Development Conference (ADC '03). Netherland: Philips Research – Software Engineering Services (SES). pp. 120-124.

APPENDICES

Appendix 1: XP practices

XP practices (Aarnio 2001, pp. 19-24; Beck 1999) are described with entailed benefits in the following Table:

Name of practice	Description, benefits included
Planning Game	- customer decides scope and timing according to estimations by designers - can be used during Planning and Iterations to Release phases
Small release programs	- content of releases is small and releases are made often
Metaphor	- metaphors are shared between customer and designers - helps everyone to understand basic elements and their relationships
Simple Design	- light documentation - better testability and readability etc.
Testing	- Test Driven Development (TDD) - unit and acceptance testing - continuous and automated testing
Re-factoring	- improving structure of code without affecting its external behavior
Pair Programming	- pair programming with each other at a single computer - changing pairs occasionally
Continuous integration	- code is integrated (with e.g. CVS) and tested at least daily
Collective Ownership	- everyone team member is responsible for code
On-Site Customer	- customer is part of XP team and makes business decisions and prioritizes user stories (features) nearby XP team
40-hours weeks	- no overtime work, 40 weekly hours is maximum - maximum can also be e.g. 37.5 hours (typically weekly working period in Platform R&D and Application R&D organizations is 37.5 hours)
Open workspace	- team works in large room with small cubicles - sitting arrangements
Just rules	- team members must follow rules, but they can change them if they agree how they will assess possible effects of the changes

Appendix 2: Scrum's Product Backlog

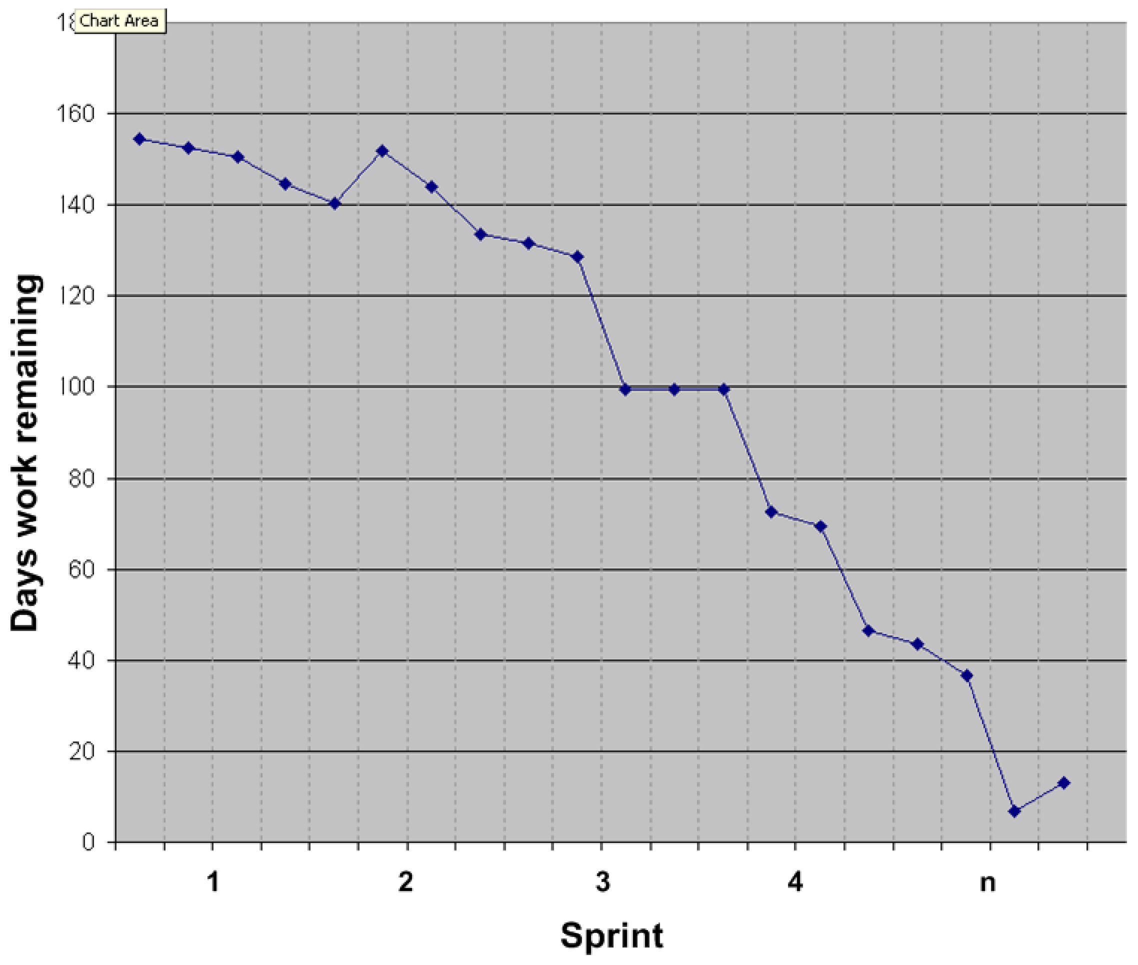
An adapted Product Backlog appears as follows:

Prioritization	**Item #**	**Feature**	**Release**	**Estimation (in days)**	**Estimated by**
Very High	1	<name of feature>	<e.g. name of release in which a feature related functionality is developed>	<estimation>	<name of person>
Very High	2				
High	3				
High	4				
High	5				
High	6				
Medium	7				
Medium	8				
Medium	9				
Medium	10				
Medium	11				
Medium	12				
Medium	13				
Medium	14				

See also Schwaber (2003, p. 10)

Appendix 3: Scrum's Product Burn-Down chart

An example of the Product Burn-Down chart appears as follows:



See also (Schwaber 2003, pp. 11-12).

Appendix 4: Scrum’s Sprint Backlog

An adapted Sprint Backlog appears as follows:

Item #	Task	Responsible person	Status	Hours remaining (week 1)	Hours remaining (week 2)	Hours remaining (week n)
1	<name of task >	<responsible person>	<e.g. not started, ongoing or com-pleted>	<hours>	<hours>	<hours>
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						

See also (Schwaber 2003, p. 13).

Appendix 5: Scrum's Release Backlog

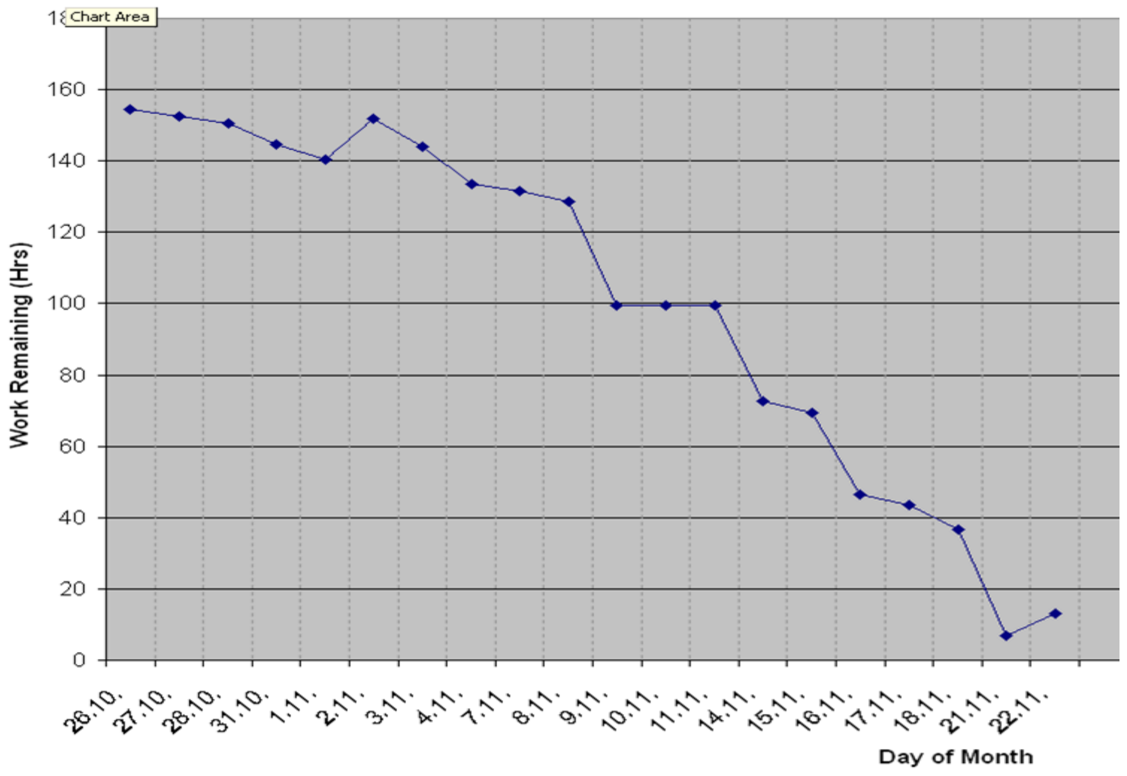
An adapted Release Backlog appears as follows:

Prioritization	Item #	Feature	Estimation (in sprints)	Assigned to team
Very High	1	<name of fea-ture>	<estimation, i.e. how many sprints has been esti-mated to need that feature related functionality would be ready>	<name of team that is responsible for devel-opment work for a fea-ture>
Very High	2			
High	3			
High	4			
High	5			
High	6			
Medium	7			
Medium	8			
Medium	9			
Medium	10			
Medium	11			
Medium	12			
Medium	13			
Medium	14			

See also Schatz & Abdelshafi (2005, p. 39).

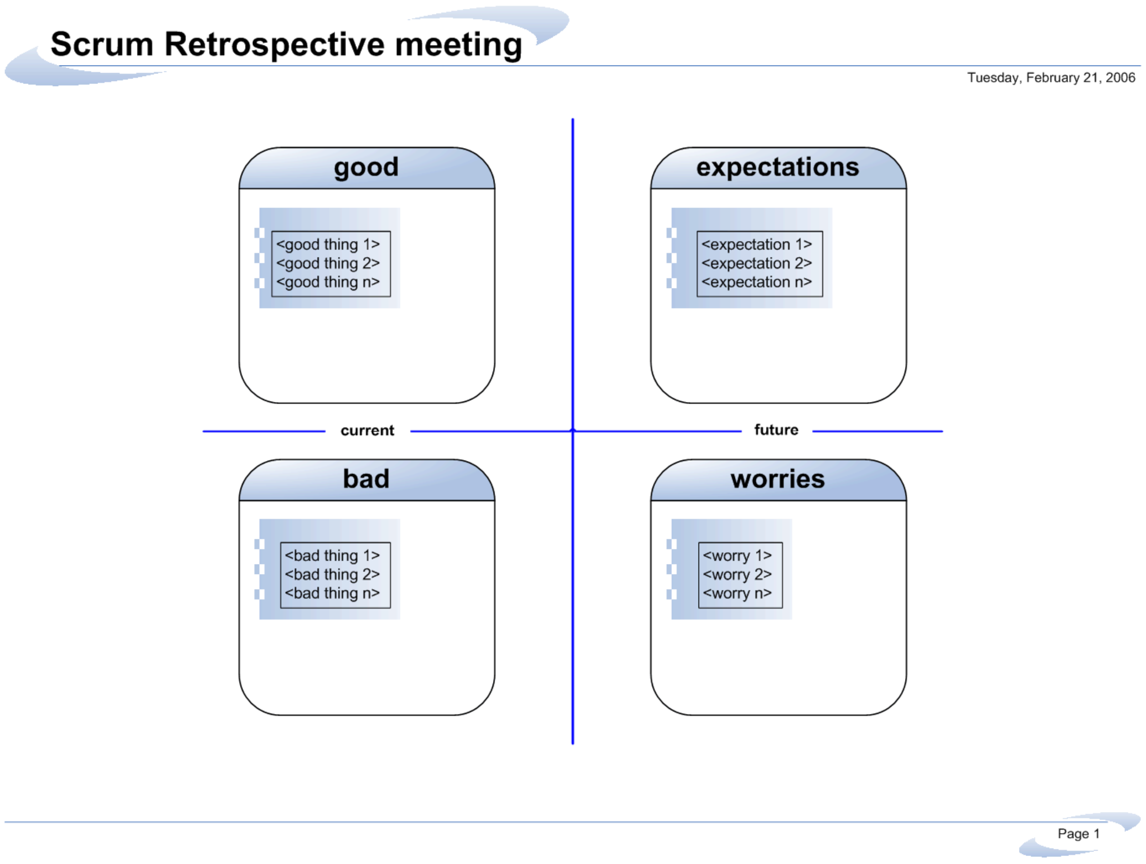
Appendix 6: Scrum's Sprint Burn-Down chart

An example of the Sprint Burn-Down chart appears as following chart that was used during Scrum piloting in Platform R&D organization:



Appendix 7: Scrum Retrospective Meeting chart

An example of the Scrum Retrospective chart appears as follows:



Appendix 8: Scrum practices

Scrum practices that were discussed in chapters 2.1.3 and 2.3 are described with entailed benefits in the following Table:

Name of practice	Description, benefits included
Sprint	- fixed period (30 days iteration) - SW is developed iteratively and incrementally during above period - requirements etc. are as input of sprint - increment of product is as output of sprint - freezing the development scope for sprint, i.e. content of release program can be added until start of next sprint
Small tasks	- e.g. features are split into small tasks - the tasks are prioritized for each sprint and sorted by team members - small tasks are easier to estimate more specific than large tasks
Small teams	- flexible and adaptable alternative from organizing perspective
Sprint Planning Meeting	- first part of meeting: Product Owner presents a Product Backlog to team and team decide what it must to do during sprint - second part of meeting: Sprint Backlog and possible Release Backlog are constructed
Daily Scrum Meeting	- team members tell what they have done since last daily meeting and what they will do before next meeting - also about possible obstacles are discussed during daily meetings - effort estimations are written by team members to Sprint Backlog - also tacit and explicit knowledge can be found and perceived better due to knowledge sharing during daily meetings
Sprint Review Meeting	- developed and (automatically) tested functionality (increment) of product is demonstrated to Product Owner and stakeholders - timely adaptation to the project can be made - end users are able to tell possible elaborations to the product
Sprint Retrospective Meeting	- team discusses about experiences regarding completed sprint and plans how to make things better during next sprint
Meetings (all Scrum related meetings in brief)	- communication and knowledge are increased - problem solving and clearing of obstacles, fast feedback - finding new ideas become possible - volunteerism and motivation can be increased within self-organizing team etc.
Use Cases	- SW is developed incrementally based on e.g. Use Cases that are used e.g. in RUP method - Use Cases can be assigned to someone in the team - can be e.g. sequence of some feature to illustrate what kind of functionality should be in practice - also e.g. test cases can be illustrated to be as Use Cases
Continuous integration	- code is integrated (with e.g. CVS) and tested at least daily
Product Backlog	- list of changes that will be made to the product for future release programs - above changes can be e.g. functions, technologies, enhancements and bug fixes etc. - Product Owner owns Product Backlog
Release Backlog	- subset of Product Backlog that is selected for a release program - Release Backlog is estimated in hours
Sprint Backlog	- estimations and task names for development work to be done during sprint - Sprint Backlog is estimated in hours and team members maintain it
Product Burn-Down	- can be used to indicate days the work remaining during sprint - due to above, managing the anticipated end date and progress of the entire project is easier
Sprint Burn-Down	- can be used to indicate hours the work remaining during sprint - cf. Product Burn-Down

Appendix 9: Interview Questions

The interview questions are described below. The questions have been described in Finnish because the interviews were arranged in Finnish.

Taustaa (*background related questions*) koskevat kysymykset:

- Kuinka kauan olet ollut Nokian palveluksessa?
- Millainen koulutustausta sinulla on?
- Millaisia työtehtäviä olet tehnyt viime aikoina?

Tutkimusongelmaan (*research problem related questions, i.e. questions to collect answers to second and third research sub-questions*) liittyvät kysymykset skenaarioineen:

Ajatellaan seuraavaksi sellaista skenaariota, että johonkin meneillään olevaan release-programmiin on päätetty ottaa loppuvaiheessa mukaan jokin uusi ominaisuus, niin:

- Liittyykö sellaiseen tilanteeseen ongelmia, (ts. hidastaako jokin asia prosessin etenemistä)? Millaisia? Miten kyseisiä ongelmia voitaisiin ratkaista?
- Mahdollisia muita ongelmia? Millaisia? Miten kyseisiä ongelmia voitaisiin ratkaista?
- Minkä tahon toimesta uuden ominaisuuden mukaan ottamista koskeva pyyntö yleensä tulee?
- Päätöksenteko (mielipide asiasta ja mahdolliset kehittämisideat)?
- Ominaisuuksien mukaan ottamiseen liittyvä priorisointi (mielipide asiasta ja mahdolliset kehittämisideat)?
- Mitkä kriteerit vaikuttavat siihen, että otetaanko jokin ominaisuus johonkin meneillään olevaan release-programmiin tai vasta seuraavaan release-programmiin?
- Asiakkaan taholta tulevien muutospyyntöjen käsittelyn sujuvuus (mielipide asiasta ja mahdolliset kehittämisideat)? Entä yrityksen sisältä tulevien muutospyyntöjen käsittelyn sujuvuus (mielipide asiasta ja mahdolliset kehittämisideat)?
- Missä vaiheessa (milestone) uusia ominaisuuksia ei pitäisi enää ottaa mukaan johonkin release-programmiin, vaan siirtää ne seuraavaan?
- Testaushenkilöstölle, projekti- ja programpäälliköille lisäksi kyseisiin toimenkuviin liittyviä lisäkysymyksiä kuten mm. seuraavat:
    - Testauksen organisointi? Mahdolliset ongelmat? Mahdolliset kehittämisideat?
    - Projektin organisointi? Mahdolliset ongelmat? Mahdolliset kehittämisideat?
    - Programmin organisointi? Mahdolliset ongelmat? Mahdolliset kehittämisideat?

Ajatellaan seuraavaksi sellaista skenaariota, että jostakin meneillään olevasta release-programmista päätetään ottaa loppuvaiheessa pois (siirretään esim. seuraavaan release-programmiin) jokin ominaisuus, niin sovelletaan em. kysymyksiä.

Muut (*other questions*) kysymykset:

- Millaiseksi koet nykyisten työtehtäviesi määrän?
- Työympäristön viihtyvyys?
- Motivointitekijät?
- Työtehtäviin liittyvä kommunikointi?
- Muita mahdollisia kehittämisideoita?
- Asioita, jotka on järjestetty organisaatiossa jo riittävän hyvin?
- Mahdollisia muita kysymyksiä.

Appendix 10: Summarized problems with most essential proposed solutions

Assorted list of solutions proposed by interviewees vs. literature are unambiguously described in following Table:

Problem	Proposed solution(s) by interviewees	Proposed solution(s) found from the literature
Lot of parallel tasks in daily work (chapter 3.3.1)	- Dedicated persons for CR work - Dedicated persons for maintenance (e.g. fault correction) work etc. possible responsibilities	- Iterative and incremental SW development: e.g. freezing development scope & fixed time - Team possibility to make decisions regarding e.g. programming - Meetings & workshops - On-site coach - Backlogs etc.
Lack of competence sharing (chapter 3.3.2)	- Job rotation through organization - Use Cases could be used for planning of test cases	- Pair programming - TDD with training sessions - Meetings & workshops - Prototyping for large features - Planning Game etc.
Bureaucratic decision making (chapter 3.3.3)	- Minimum bureaucracy - Decision making could be done by low-level management.	- Minimum bureaucracy - Basics of decision making etc.
Underestimation of needed resources (chapter 3.3.4)	- Time allocations (20% of total effort) for CR work - Continuous resource allocation - Prioritization of CRs should be by both R&D organizations	- Spike-time for e.g. learning - Red-Flag for managing unexpected work - Portfolio Management - Product Raodmap & backlogs - Meetings & workshops etc.
Inadequate co-operation between testing and SW groups (chapter 3.3.5)	- Combined FT and SW work - Sitting arrangements - Structure of teams could be organized	- Pair programming → pair testing - TDD - Automated testing with test reports - Meetings & workshops etc.
Lacking Feature Owner (chapter 3.3.6)	- Feature Owner could own a feature through its lifecycle - Fast feedback by Feature Owner	- Product Owner → Feature Owner - Meetings & workshops etc.
Inadequate communication (chapter 3.3.7)	- Specific tasks would be done in same building - Meetings & workshops - Increasing communication through organizations	- Communication model - On-site customer - Meetings & workshops etc.
Late changes to the content of release programs (chapter 3.3.8)	- Small content of release programs - Major decreasing CRs - No CRs for minor (under 50 hours) changes - No CRs after E1 milestone	- Small release programs - Iterative and incremental SW development - Product Raodmap & backlogs - on-site customer etc.
Inconsistent synchronization of release programs (chapter 3.3.9)	- Iterative and incremental SW development - Prototyping and Simulation Game - Meetings & workshops	- Iterative and incremental SW development - Small release programs - Backlogs etc.
Heavyweight documentation and bureaucratic reviews (chapter 3.3.10)	- Lightweight documentation - Formal inspections only for major changes - Incremental development - Group formation - Design details can be added into code as comments	- TDD - Red-Flag - Spike-time etc.
Lack of versioning knowledge of program blocks (chapter 3.3.11)	- Modularization - CVS	- Iterative and incremental SW development - High cohesion and low coupling - Continuous integration etc.

Appendix 11: Other practices

The practices of other agile methods were discussed in chapters 2.1.4 and 2.4. Part of them already included discussion about XP and Scrum. Such practices have been excluded from the following Table. Only other practices that were found due to the literature review are described with entailed benefits in the following Table:

Name of practice	Description, benefits included
Code reviews	- quality increases - SW design related things can be added to comments in code and therefore no SD documentation is necessarily needed
Coding standards	- consistent way to produce code
Spike-time	- time is allocated for e.g. reading and learning about various issues
Red-Flag	- Red-Flag practice is used to describe e.g. in Sprint Backlog how much effort should be reserved for unexpected work during sprint
Zero Feature Iteration	- makes possible to get early information on the system architecture before real First Iteration - possible to evolve the architecture at early stage before too much time had been used to build functionality
On-site coach	- on-site coach can track what team members are doing and help them solving their possible problems
Prototyping	- prototyping can be used for large features - it is easier to estimate how much effort is needed to implement real feature when prototype is in view
Continuous and automated testing	- automated regression testing is suggested to start as early phase as possible - estimation how much tests have already been automated could be given as percentage and number should be public - testing could be automated - after execution of automatically tests (e.g. so called basic test set) a test report could be generated automatically - see more information also from description of TDD (practice by XP)
Reflection workshops	- similar than meetings - can be arranged to increase maturity and effectiveness of team
Idea pool	- part of Strategic Planning Cycle in Cycles of Control framework, i.e. continuous planning of product - should be in specified place where everyone (e.g. team members) can see it
Portfolio Management	- provides the interface between business and product development - provides guidelines for how product development efforts should be organized in terms of SW development cycle and release cycle - manages product roadmap
Release Project Management	- handles the development of individual product versions
Increment Management	- manages incremental development of product functionality within release programs
Heartbeats	- provides scheduling and monitoring for daily or weekly tasks