HELSINKI UNIVERSITY OF TECHNOLOGY
Department of Electrical and Communications Engineering

Lauri Vuornos

Software Process Improvement in a Small High-Tech Company: Case Smartner Information Systems Ltd.

Master's Thesis

26.5.2002

Supervisor: Professor Casper Lassenius
Instructor: M.Sc. Kristian Rautiainen

Abstract

Author and name of the thesis:
Lauri Vuornos:
Software Process Improvement in a Small High-Tech Company: Case Smartner Information Systems Ltd.

Date: 26.5.2002 **Number of pages:** 105

Departement:	**Professorship:**
Electrical and Communications Engineering	T-76

Supervisor:
Professor Casper Lassenius
Instructor:
M.Sc. Kristian Rautiainen, Helsinki University of Technology

The objective of this thesis is to help the case organization, Smartner Information Systems Ltd., to improve its software development process. The main focus is on defining a new software process model for the case organization and a suggestion for deployment steps of the new model.

The thesis begins with defininition of the research problem and goals, and with presenting motivations and background for the work. It is followed by an introduction of the case organization and its current software process.

Requirements for the new process model are worked out with professionals from the case organization based on current needs and challenges in the software development. Based on a literature study, four existing software process models are evaluated against the requirements.

The central part of the thesis presents the new software process model for the case organization. The new model is based on the requirements and the evaluated existing models. The new model takes selected practices from existing software process models and combines them into a new model.

The last part of the thesis compares the current practices of the case organization to the new model and introduces a suggestion for deploing the new model. The suggestion consists of improvement steps that aim to solve specific problems in the current practices of the case organization by deploying practices of the new model.

Keywords: software development process, software engineering, software process improvement, agile software process

Tiivistelmä

Tekijä ja työn nimi: Lauri Vuornos: Software Process Improvement in a Small High-Tech Company: Case Smartner Information Systems Ltd.	
Päivämäärä: 26.5.2002	**Sivumäärä:** 105
Osasto: Sähkö- ja tietoliikennetekniikan osasto	**Professuuri:** T-76
Työn valvoja: Professori Casper Lassenius **Työn ohjaaja:** DI Kristian Rautiainen, Teknillinen korkeakoulu	

Tämän diplomityön tavoitteena on auttaa case-yritystä, Smartner Information Systems Ltd., parantamaan ohjelmistoprosessiaan. Työn pääpaino on uuden ohjelmistoprosessimallin kehittämisessä case-yritykselle, sekä ohjelmistoprosessin kehitysehdotuksen laatimisessa perustuen uuteen malliin.

Tämä työ alkaa tutkimusongelman ja tavoitteiden määrittelyllä, työn tarpeellisuuden perustelulla, sekä työn taustan kuvaamisella. Tämän jälkeen case-yritys ja sen nykyinen ohjelmistokehitystoprosessi esitellään.

Vaatimukset uudelle ohjelmistoprosessille selvitetän yhdessä case-yrityksen asiantuntijoiden kanssa. Vaatimukset pohjautuvat case-yrityksen ohjelmistokehityksen nykyisille haasteille ja ongelmille. Kirjallisuustutkimuksen perusteella neljä olemassa olevaa ohjelmistoprosessimallia arvioidaan vaatimuksia vasten.

Keskeisimmässä osassa työtä esitellään uusi ohjelmistoprosessimalli case-yritykselle. Uusi malli pohjautuu määritellyille vaatimuksille, sekä olemassa olevien mallien arvioinneille. Uuteen malliin yhdistetään valitut käytännöt ja periaatteet arvioiduista ohjelmistoprosessimalleista.

Työn viimeisessä osassa verrataan case-yrityksen ohjelmistokehityskäytäntöjä uuteen malliin, sekä esitetään ehdotus mallin käyttöönotosta. Ehdotus koostuu parannuskokonaisuuksista, jotka pyrkivät ratkaisemaan tiettyjä ongelmia case-yrityksen nykyisessä ohjelmistokehityksessä uuden mallin avulla.

Avainsanat: ohjelmistoprosessi, ohjelmistotuotanto, ketterä ohjelmistoprosessi

Preface

This thesis has been written for Smartner Information Systems Ltd.

I am very grateful to Smartner Information Systems Ltd., especially my superior Ari Backholm, for providing me the time and resources to write this thesis.

I would like to thank my supervisor, Professor Casper Lassenius for his valuable comments and interest in the work. My instructor Kristian Rautiainen has been very helpful in providing feedback and thoughts about software development process in small companies.

I would also like to thank Miira Heiniö for helping me with the English grammar and linguistics.

Special thanks to Hanna Salakari for patience at home and for encouraging me to finish this thesis.

Helsinki, 26.5.2002

Lauri Vuornos

Contents

List of Terms

.NET is Microsoft's platform for XML Web services, the next generation of software that connects our world of information, devices and people in a unified, personalized way.

4CC stands for four cycles of control. It is a tentative software process management framework developed in SEMS research project.

Ant is tool for building software written in Java.

Application server is a platform for running web applications (for example BEA Weblogic or IBM WebShere).

Brainstroming is a technic for creating innovative ideas and solutions to problems with a small group of people in a very informal meeting.

Bugzilla is a widely used open source defect tracking system.

CEO or Chief Executive Officer directs the whole company and is usually reports to the board of directors.

CMM or Capability Maturity Model for Software developed by Software Engineering Institute (SEI) at Carnegie Mellon University in 1991.

CVS or Concurrent Versions System is a widely used version control system for source code and documents.

Coding standard is a set of rules that developers follow when implementing software.

Core product is a product that has been chosen to be strategically the most important product for the company.

IDE or Integrated Development Environment is an environment supporting software development.

J2EE or Java 2 Enterprises Edition is a standard for developing multitier enterprise applications.

JUnit is a testing framework for unit testing with Java especially ment for XP style development.

Javadoc is a document generated from Java source code. It is a standard way to document implementation done with Java.

LDAP or Ligh-weight Directory Access Protocol is widely used for accessing contact information.

PDA or personal digital assistant is a pocket size personal computer.

PKI or Public Key Infrastructure is a term generally used to describe the laws, policies, standards, and software that regulate or manipulate certificates and public and private keys.

Peer review is a technic where a colleague reviews the work of someone else from the team.

Planning game is a meeting where a customer and a development team define the content of an XP project.

Platform is a software application that offers services to other applications.

Product roadmap is a high-level release project plan for all the products of the company.

Quality gate is checkpoint where some quality attribute is measured.

RUP or Rational Unified Process is a software process framework based on the USDP. RUP is described in more detail in section 3.1.1.

Roadmapping is an interactive process for producing a roadmap. It provides a way to evaluate different strategic alternatives to produce the best possible plan to reach the business objectives of the company.

SEMS research project develops a Software Engineering Management System for small and medium-sized software enterprises. The SEMS project is planned to span over three years (2001-2003) and it is part of the TEKES technology programme SPIN.

SPICE is a major international initiative to support the development of an International Standard for Software Process Assessment (ISO/IEC TR 15504:1998)

Scrum is an agile software process model. Scrum is described in more detail in section 3.1.3.

Synchronize and Stabilize is software process model used in the Microsoft. Synch and stabilize is described in more detail in section 3.1.4.

TLS or Transport Layer Security is a protocol that provides security to application protocols.

Triage is a concept defined by: 1) A process for sorting injured people into groups based on their need for likely to benifit from immediate medical treatment. 2) A system used to allocate a scarce commodity, such as food, only those capable of deriving the greatest benifit from it.

UML is an industry-standard language that allows us to clearly communicate requirements, architectures and designs.

UMTS or Universal Mobile Telecommunications System is a part of the International Telecommunications Union vision of a global family of 'third-generation' (3G) mobile communications systems.

USDP or Unified Software Development Process is a process framework taking advantage of visul modeling with UML.

VP or Vice President of the company.

WAP or Wireless Application Protocol is used in mobile devices mostly in Europe.

WLAN or Wireless Local Area Network is a wireless equivalent of widely used local area network.

Workshop is a interactive and rather informal meeting where a group of people work for a common goal.

XP or Extreme Programming is an agile software process model. XP is described in more detail in section 3.1.2.

Chapter 1

Introduction

In this chapter background for the research is presented. The research problem and goals are defined as well as the scope of the research.

1.1 Background

It is widely understood that an appropriate software development process can considerably improve the effectiveness of software engineering. However, finding an appropriate software process might not be easy. Small high-tech software companies developing software products for new markets often face problems when adapting traditional software process methodologies such as CMM[1]. Judith Brodman (Brodman and Johnson, 1994) argues that small software businesses often face a situation where they try to fund costly software process improvement programs without substantially raising their overhead rates. Needs and requirements that those companies have for an efficient software development process vary considerably from those of bigger companies. On the other hand, continuous lack of resources makes it difficult to maintain and improve the software development process. Small software businesses often end up balancing between an ad-hoc process and a formal software process model. This can easily lead to ineffective software engineering.

The majority of companies developing software are small. According to the US Bureau of Census 1995 County Business patterns, almost 90% of the software companies in the United States have fewer than 50 employees (Laitinen et al., 2000). Respectively, in Finland TAI Research Center reports (Järvenpää and Aalto, 2000) that in the year 1999 approximately 70% of the software companies had twenty or less employees. Studying the software development process in small businesses is a fairly recent phenomenon. Most of the studies on that topic have been done in the past few

[1]Capability Maturity Model for Software developed by Software Engineering Institute (SEI) at Carnegie Mellon University in 1991.

years. The perspective in those studies is most often tailoring an existing and widely accepted model for small software development companies. For example, Judith Brodman has studied how CMM fits for small businesses (Brodman and Johnson, 1994) and Timo Varkoi has studied software process improvement in small software companies based on the SPICE[2] framework (Varkoi and Jaakkola, 1999). However, most of the studies, including the ones mentioned, concentrate more on contract based software development which differs considerably from software product business. The problems caused by combination of smallness and focusing on new software product markets with new and partly unstable technology have not been studied almost at all.

When a small software product company, like the case organization, looks for improvement in its software development process it easily faces a difficult situation. There are many different process models that promise improvements in software engineering but none of them seems a good fit for the company. The impression could be that those models have been developed for larger organizations or for a completely different business environment. Developing software products for new and unstable markets with latest technology raises many challenges for a suitable software development process. Requirements of the software product change continuously during the product development process, which has not been the case in the well-established application domains where the used technology is mature. Also, the first mover advantage is considered even more valuable in the new highly competed markets. In the book *Microsoft Secrets* (Cusumano and Selby, 1998) Michael Cusumano and Richard Selby point out that continuous change and competition has made Microsoft create new software development methods that take changes for granted rather than as exceptional. This software development methodology is called *synchronize-and-stabilize*, and it has proven its efficiency in Microsoft. Another perspective to the problem are the methodologies that have been developed for small groups and companies developing software with vague or rapidly changing requirements. Maybe the best known of these so called agile process models is extreme programming (XP) developed by Kent Beck (Beck, 2000).

Research Background

The combination of smallness and focus in the new markets with latest technology raises the motivation for this research. Uncertain and dynamic describe well the environment of the case organization. Alan MacCormack defines this in a very suitable way in his article *Developing Products on 'Internet Time'*(MacCormack, 2001): *"By uncertain, we mean environments in which future evolutions in markets and/or technologies are hard to pre-*

[2]SPICE is a major international initiative to support the development of an International Standard for Software Process Assessment (ISO/IEC TR 15504:1998)

dicts. By dynamic, we mean environments in which these evolutions occur rapidly". This research aims to solve problems that the case company has with its software development process. Basically, the most important challenge in the case organization is to make the software development process more effective, maximizing the value produced for customers. Challenges that the uncertain and dynamic environment of the case organization introduces, need to be answered. The software process should provide more predictability, visibility to product development and have a better connection to the business strategy of the company. These challenges are the reason why the case organization is looking for improvement in its software process.

Research is done in a very problem centric manner. Literature is studied and analyzed from the viewpoint of the research problem which is defined in detail with the case organization. Knowledge gathered from literature is combined with findings from a case and synthesized into a process model and a improvement plan for the case organization.

Case organization Smartner Information Systems Ltd. is a software product and professional services company enabling mobile business services. Smartner offers its mobile technology for operators and application service providers who need tools for building mobile services and solutions for enterprises. At present, the Smartner team consists of about 30 professionals working in the fields of mobile solutions, software development and business management. Smartner is based in Helsinki, Finland.

1.2 Research Problem

The research problem can be stated as a question: How can the software development process be improved in the case organization, a small software product company?

Terminology used in the field of software process development is not well-established. The same terms may have different definitions depending on the focus of the research or even depending on the author of the paper. Software development process has many definitions in literature such as definitions by CMM (Paulk et al., 1991), SPICE (Dorling et al., 1996) and USDP (Jacobson et al., 1999), just to mention a few. Fortunately, the key issues seem to be quite similar in all of them. A software development process defines who is doing what, when and how to reach the goal: a new software product or a change in an existing one. In this research, a widely accepted definition is adopted from Capability Maturity Model (Paulk et al., 1991): *"Software process is a set of activities, methods, practices and transformations that people use to develop and maintain software and the associated products".*

To be able to answer the research question both the organizational factors and existing software process models need to be understood. Existing software development process models need to be evaluated against the needs and requirements of the organization. This information is the basis for defining the new software development process model for the case organization. For making concrete suggestions for the improvement, the current state of the software development process must be known in the case organization and it needs to be compared to the new model. Studying these aspects provides the essential information for answering the research question. In short we need to know:

- What are the requirements for the software development process considering the needs of the case organization?
- How do existing process models and software development methodologies support these requirements?
- What is an adequate software development process for the case organization?
- What is the current situation of the software process in the case organization compared to the new model?
- What steps should the case organization take to deploy the new software development process model?

These questions also define the phases of the research and the structure of this document. The research is done in close cooperation with the case organization to ensure concrete results.

1.3 Goals and Objectives

The main objective of this research is to help the case organization improve its software development process.

The effectiveness of a software development process depends on various factors. It is not well understood which factors actually help to improve software engineering in different situations. This emphasizes that understanding the requirements for the software development process in a certain situation is essential. It also makes it hard to find one process model that provides answers to all questions. The solution could be in using a set of practices from different methodologies rather than just one process model. McCormack summarizes this idea in his article *Programming Extremism* (McCormick, 2001): *"What's needed is not a single software methodology, but a rich toolkit of process patterns and 'methodology components' (deliverables, techniques, process flows, and so forth) along with guidelines for how to plug them together to customize a methodology for any given project"*.

Considering McCormack's citation it is obvious that information about the case organization and understanding existing software process models are vital issues for the research. To reach a more concrete level in research goals the research objective must be divided to goals that turn the research questions into actions. In order to achieve the objective we need to:

- Recognize the requirements for the software development process in the case organization.
- Evaluate existing software process models against the requirements.
- Define software development process model for the case organization.
- Recognise the similarities and differences between the current state of the software process in the case organization and the new model.
- Define steps that case organization should take to improve its software development process.

These goals define the milestones of the research and form a structured way to answer the research problem. Relations between research problems and goals are visualized in figure 1.1.

1.4 Research Scope

This research concentrates on the software development process of the case organization. Business processes are not discussed from an improvement point of view but are taken as given border conditions. This includes defining the business and technology strategy of the company.

The delivery process of different software products to various customers is a research area of its own. Providing customers with product customization, integration to existing systems and other professional services have different characteristics than product development. It is remarkably closer to traditional contract based software engineering than product development. In this research, focus is on the product development process hence the delivery process is not studied in detail. Only the aspects relevant to software product development are discussed.

Deployment of the new process model in terms of the methodology used to communicate the process to the organization and how process information is managed is not examined in detail in this research. These activities are commonly referred to as software process support, which forms a research area of its own and it is not in the scope of this research. The importance of software process support is considerable but deploying it requires the software process to be defined at least to some extent. Importance of the software process support is described well by Timo Kaltio and Atte Kinnula in their study *Deploying the Defined Software Process* about process deployment and process asset management at Nokia Mobile Phones (Kaltio and Kinnula, 1998). They stress the importance of a well organized support

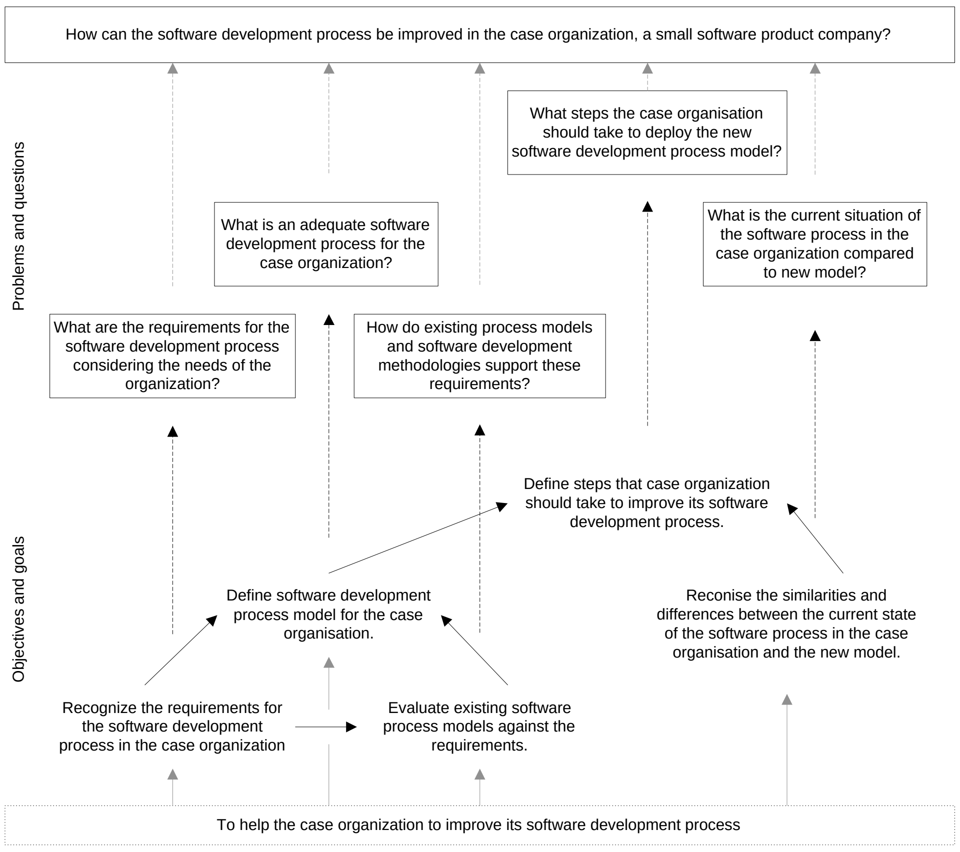

How can the software development process be improved in the case organization, a small software product company?
Problems and questions
What steps the case organisation should take to deploy the new software development process model?
What is an adequate software development process for the case organization?
What is the current situation of the software process in the case organization compared to new model?
What are the requirements for the software development process considering the needs of the organization?
How do existing process models and software development methodologies support these requirements?
Objectives and goals
Define steps that case organization should take to improve its software development process.
Define software development process model for the case organisation.
Reconise the similarities and differences between the current state of the software process in the case organisation and the new model.
Recognize the requirements for the software development process in the case organization
Evaluate existing software process models against the requirements.
To help the case organization to improve its software development process

Figure 1.1: Research problems and goals

infrastructure, having the right product (for accessing process assets) and effective promotion. Studying these aspects would be a natural follow-up for this research.

Research does not produce *lessons learned* from implementing the improvement plan since deploying a new process model is a time consuming activity and does not fit in to the time frame of this research.

This research is focused on software development process in the case organization. However, generalization of the research results is briefly discussed in the last chapter 6.

1.5 Research Structure

The research problems and goals form the structure of the research. The chapters of this document are structured respectively:

Chapter 1 Introduces the research by defining the research problem and goal.

Chapter 2 Defines the requirements for the new process model.

Chapter 3 Evaluates existing process models.

Chapter 4 Presents the new software process model.

Chapter 5 Compares the current situation in the case organization to the new model and introducing improvement steps for the case organization.

Chapter 6 Conclusions about the results.

How each chapter relates to the others is described together with the research methodology in section 1.6.

1.6 Research Methodology

A literature study is used to gather relevant information about existing software development processes and previous research results in the same field. Workshops are used when information is gathered from the case organization. Some of the information about the case organization comes from personal experiences, since the author of this research is part of the case organization. Information from the case organization was gathered in two phases: when defining the requirements and when describing the current state of the software process in the case organization. The author's experience was used in the later phase, and other professionals from the case organization were heard in the first phase.

The basis for the whole research is the first goal of recognizing the requirements for the software development process. The case organization's needs for the product development and its current challenges were worked out in a workshop with professionals from the case organization. Experiences from the same kind of situations found from the literature were combined with the results. Finally requirements for the software development process were produced from the needs found in the workshop and the literature.

Selected existing software development process models were evaluated against the requirements to form a compact picture how they support each of the requirements.

Suitable elements from different models are combined with practical knowledge to form a new software development process model that supports the defined requirements as well as possible.

The new software process model was compared to the current practises of the case organization using author's experiences as a source for information. The reason for this approach was that author has been participating in product development of the case organization at all the levels from roadmapping to coding and testing the products. The goal of this comparison was to point out the shortcomings of the current software development practices in the case organization compared to the new model.

A suggestion for improvement steps for the case organization is introduced. The improvement steps are based on the differences found from comparing the current practices in the case organization and the new model. Since deploying a new software process is not a trivial thing to do, and deployment not being within the scope of this research, the improvement steps form a loose framework rather than a detailed project plan for improvements.

Contribution and reliability of the research results are discussed briefly in the last chapter with suggestions for further research.

Chapter 2

Requirements for the Software Process

In this chapter the requirements for the software development process in the case organization are defined. To give background for the study, the case organization, Smartner Information Systems Ltd., is briefly introduced and the current software process of the case organization is presented. A workshop for defining the needs for the product development is described and the results are analyzed against the literature. Finally, requirements for the software development process are defined based on workshop results and literature.

2.1 Case Organization

The case organization, Smartner Information Systems Ltd is a software product and professional services company enabling mobile business services. Smartner offers its mobile technology competence to operators and application service providers who need tools for building mobile services and solutions for enterprises. Smartner is developing its business towards international software markets. At the moment, Smartner targets European markets.

At present, the Smartner team consists of about 30 relatively young professionals having expertise in the fields of mobile solutions, software development and business management. Smartner was founded in 1999 and is based in Helsinki, Finland. Smartner is privately owned by personnel, four partners and three investors: EQVITEC Partners Oy[1], Sitra[2] and IT

[1]EQVITEC Partners Oy is a leading Finnish venture capital firm focusing on technology investments (http://www.eqvitec.com).

[2]Sitra, Finnish National Fund for Research and Development, is a leading governmental fund dedicated to promising early stage technology investments (http://www.sitra.fi).

Provider Adviser 1 AB[3]. Smartner's second financing round was completed in fall of 2001.

2.1.1 Customers and Products

Smartner offers products to wireless application service providers, who need tools and applications to enable wireless services targeted for business users. The leading idea in Smartner's product development is to provide technology that integrates mobility into existing enterprise information systems. With Smartner's solution wireless service provides, Smartner's target customers can provide their enterprise customers with value-added wireless services.

Smartner products are divided into wireless technology and wireless applications. Wireless technology enables rapid application development. Wireless applications are ready-to-use solutions taking advantage of the Smartner's wireless technology. Smartner's currently shipping products are Smartner Engine 2.1, which is a wireless application platform, Smartner Messaging Server 2.0, which is a multi-network messaging platform, Smartner Enterprise Gateway 1.2, which enables a secure connection to enterprise systems and Smartner Office Extender 1.2 bringing office solutions into end users WAP, PDA and SMS enabled devices.

The business environment where Smartner operates is a highly competed emerging market. Many big players such as Microsoft have just released their competing solution for mobile office market as well as some smaller players like LPG Innovations Oy from Finland and Fenestrae from Netherlands. According to Goldman Sachs 2000 (Sachs, 2001) this market is going to grow extremely fast: the number of mobile internet users will increase from the present 5 million to 49 million by year 2002. It is estimated that by year 2004, companies have spent a total of 154 billion dollars on mobile services.

2.1.2 Units and Teams

Smartner's organization is divided into three units: Sales, Research and Development (R&D) and Administration. Sales and R&D both have about 45% of the personnel and administration copes with remaining 10%. The management team consists of persons that are members of some operational unit as well, except for the CEO who leads the management team.

R&D forms a single operational team whereas sales is split up into three teams with slightly different functions: sales, professional services, and marketing. The cooperation between teams and units is considerable. Delivery projects consists of people from R&D, professional services and sales. Especially R&D and Professional Services work closely together in delivery and

[3]IT Provider Adviser 1 AB is a leading Swedish venture capital advisory firm advising three different investment funds focusing on technology investments (www at http://www.itprovider.com).

integration projects. Responsibilities of the teams are in short:

Sales is responsible for negotiating and closing the deals and customer relations.

Marketing is responsible for public relations and communicating Smartner's message to the partners and customers.

Professional Services is responsible for consultation, product support and management of delivery and integration projects.

Research and Development is responsible for developing and maintaining the software products and implementing customer specific product tailoring.

Administration is responsible for the infrastructure of the company and company well being in general.

Implementing the software development process is mainly done by the R&D team who gets the product requirements and customer feedback mostly from Professional Services, Sales and Marketing. The head of the R&D, VP of business development, is the most important link between R&D, sales and customers.

2.2 Current Software Process

In this section, the current software development process of the case organization is presented. The software process is described as it is defined in the case organization. How well the current model relates to the reality is not discussed here but is left for chapter 5, where current practices are compared to the new software process model.

2.2.1 Background in Software Process Development

Software process development has been considered important since the start of the company. Management has seen it as a way of creating competitive advantage for the company. In practice, the young professionals have been open to developing their way of working towards more efficient software engineering.

Smartner has participated in SEMS[4] research project led by researchers from SoberIT (Software Business and Engineering Institute), during the year 2001. The goal of the research is to develop software engineering management system for small and medium-sized software enterprices.

[4]SEMS research project develops a Software Engineering Management System for small and medium-sized software enterprises. The SEMS project is planned to span over three years (2001-2003) and it is part of the TEKES technology programme SPIN (http://www.soberit.hut.fi/sems/english).

2.2.2 Current Software Process

The software development process has evolved significantly from the start of the company. The direction has been towards a light-weight and flexible but deadline oriented process. The key factors of the process have been:

- Flexibility
- Fixed deadlines
- Customer orientation
- Minimizing overhead

The current software process model[5] in the case organization is based on the first draft of the 4CC model (Rautiainen et al., 2002) and practices that have evolved during the years of existence of the company. Some effects of this research can also be found from the model such as the approach of combining existing process models.

The structure of the current model is based on the concepts of the 4CC model (see figure 2.1). It is a relatively high-level description of how software development is done in the company and lacks details about concrete development work. The introduction to the current software process model is based on the case organization's document *"Smartner Product Development Process in Short"* (Backholm and Vuornos, 2001), that is directed to customers and partners. There is no other process documentation in the case organization, but the common development methods are still used based on verbal agreements.

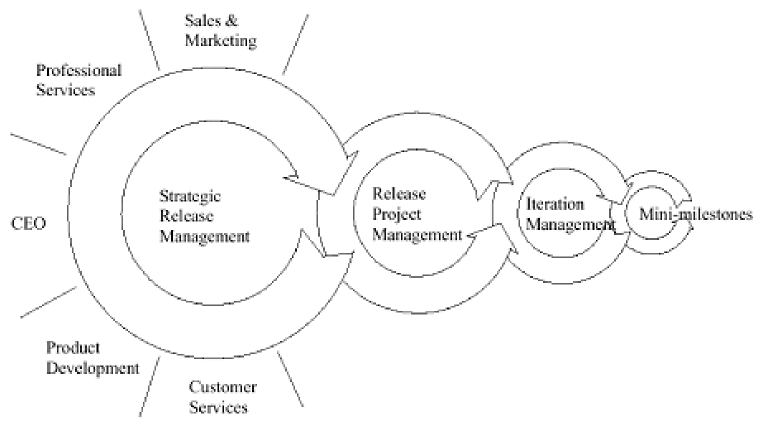


Figure 2.1: Overview of the 4CC model (Rautiainen et al, 2002)

[5]The situation at the end of the year 2001

The five corner stones of the current model are: time pacing, prioritization, ability to react to changes, iterative development and daily builds.

Strategic Release Management

Strategic release management has a one-year time horizon. It consists of setting a release schedule, roadmapping the high-level product visions, and continuous collection of requirements.

The roadmap is reviewed quarterly to ensure that needed changes in technological and business environment are taken into account. It sets the strategic vision of product development.

The requirements are collected by Product Managers. Customers, partners, other stakeholders, reports, and internal sources are inputs to the requirements collection.

All parts of the organization, especially the professionals at the customer interface, contribute to this phase, which is lead by the acting Head of Product Management.

The outputs of strategic release management are the roadmap document one year forward including the release schedule, and preprocessed requirements and ideas for forthcoming product releases. The release cycle is three months.

Release Project Management

The release project runs for each product release. A release project starts every three months to match the release cycle. The process consists of five phases:

Feature prioritization: In an internal workshop, the Smartner Product Management team will go through the preprocessed requirements and ideas, project them against the roadmap to come up with an initial prioritized list of features for the next release. This workshop considers the input gathered from various sources during the previous release project. The workshop is held two weeks before the date of Initial Feature Prioritization Freeze.

Feature iteration: The acting Head of Product Management will arrange workshops with customers and partners to go through the initial prioritized list of features for the next release. The iteration takes place to ensure that the releases respond to the needs of the customers. Customers and partners can separately agree on features that Smartner would guarantee to be implemented. Without a separate agreement, Smartner does not guarantee the implementation of any feature.

Initial feature prioritization freeze: The Smartner Product Management team will set the prioritization of features for the first implementation iteration in an internal workshop, based on the input from the iteration round. Reprioritization of features is done during the implementation iterations to ensure fast reactions to changes.

Scheduling iterations: All iterations that are to be implemented are scheduled. Iteration end products are alpha, beta and full releases.

Setting up early access: For the interests of Smartner, customers, partners, and other stakeholders, schemas are set up for early access to information about new releases, including trialing, joint testing, and early customization. The output of the release project is a fully functional release, which matches the customer needs and is deployable to customers as early as possible. The acting Head of Product Management has the overall responsibility over the release project.

Iteration Management

Iteration management has a time-frame of about one month. In this phase, one phase of the release project is planned, designed, implemented and tested. With three iterations for (1) alpha, (2) beta, and (3) full releases, the rounds sum up to the three-month release project cycle.

Each iteration starts with reprioritizing the features. This is done in cooperation with product management and project teams. This XP-style 'planning game' defines which features are implemented first and estimations for needed development time is made.

From the technical point of view, the high-level design is made to lay down a common starting point for the development. The first features to be developed need to define the architecture for the whole release.

The beta release is available for key customers to have early access to new features and for ability to further influence the development decisions.

The full release iteration of the release project is devoted to testing. New features are generally not added to the product. Testing in the last iteration concentrates on functional, usability and load testing.

The output of each iteration is a new version of the software. The new version of the software is always an installable package including the latest features. Product and project managers are in charge of iteration management.

Mini-milestones

The mini milestones have a time-frame of one week. In these mini-iterations the product is designed, implemented and tested. Mini-milestones define very concrete goals and tasks for individual developers at the weekly level. In this phase, visibility of the progress plays a key role.

Implementation and unit testing are done simultaneously in an XP type style. Also component level load testing is done side by side with implementation. Informal code reviews are used for critical components.

The concept of daily build has an important role throughout the whole release project. The product is built and tested with automated unit testers daily, using automated scripts. This ensures that modifications and new features do not break previously implemented components.

The output of each mini-milestone is a new version of the software. Mini-milestones are validated in weekly project meetings and new goals are set. The project manager is in charge of the mini-milestones.

2.3 Workshop for Defining Requirements

In this section definitions of the requirements for the software development process are presented. Initial needs and requirements for product development are worked out in a workshop with the professionals from the case organization. The results are compared to findings from literature and requirements for the software development process are introduced based on the workshop results and literature.

2.3.1 Workshop Setup

The Workshop was held as one two hour session with three professionals from the case organization. The participants in the workshop were:

The head of product development, VP of Business Development who has an important role in defining the strategy for the company, he is responsible for scheduling and defining new product releases and his viewpoint to the product is considerably non-technical.

Team leader of the R&D, Director of Product Development who is responsible of software engineering and process improvement in product development and he has good knowledge about the products in a technical sense.

Director of Business Development, Central Europe who has worked extensively in the customer interface and represents customers' and sales people's point of view.

The agenda and workshop goals were sent to the participants before the actual workshop (see the appendix A.1). The objective of the workshop was to work out the factors in the product development that are important especially for the sales and company's strategy. The goal of the workshop was to define needs for the product development in the case organization. In order to find those requirements we were seeking answers to the questions:

- What are the essential things that define Smartner's operational environment?
- What needs for product development do these factors generate?
- What are the reasons behind the current practices? Are there more needs related to those reasons? And what are the current challenges?

The operational environment was analyzed to get the right context and to outline reasons behind the needs. Needs were derived from the environment to ensure trace between need and the reason. Current practices and challeges were reverse engineered to make sure that important needs from the current situation would not be forgotten. Also new needs that might arise from the same reasons were thought over.

The agenda for the workshop was simple. The problem at hand and the workshop goals were first presented to everyone. The approach that needs from the company's strategy and sales should guide software development process improvement was explained. The essential things in environment and the needs that they introduce for product development were introduced in a brainstorming session. This method was used to maximize the discussion and the amount of ideas. The last point in the workshop was to check that no important need was missing. This was done by taking the current important practices and comparing them to the needs and by every participant thinking of his everyday work and what needs could rise from there for the product development.

The documentation throughout the workshop was done on the flip chart and on a white board. The results were written down in electronic format after the workshop from the workshop documents.

2.3.2 Overview of the Workshop Results

The general opinion of the participants was that the company is in a turning point moving from a few customers acting partly as development partners to many customers that want ready-to-use products. This naturally reflects to the needs for the product development and it is likely to introduce changes to the current practices. Additionally, participants stated that flexibility, fast responses to customers needs and ability to show the product were key issues in previously won deals. On the other hand, in the current customer cases, a ready-to-use product with well-defined features and documentation would have more value. The most important reason for being in this turning point was seen to be working with a partner who actually makes a delivery and making a deal outside of Finland where company does not have presence. In this situation professional skills and efficiency of the company's own employees is not enough but ability to transfer that needed information to the partner organization starts to play the key role.

The most important challenges, or current problem areas, were related to being in a turning point as well. More predictable development and better defined products were seen important, but at the same time, flexibility and fast reactions to changes and customer needs were considered essential. Combining these points of view was seen challenging but important. Other current problem areas that were mentioned were quality of the products and collecting feedback from customers and end-users and using it in product development.

2.3.3 Description of the Environment

The workshop agreed on three categories for operational environment: company strategy, customers and technology. Also the organization of the company was mentioned but with less importance. Even if formal definitions for these categories are not needed here short informal definitions are given to form a right idea of what they are about:

Strategy can be understood as a long term plan created by management and the organization to reach the objectives of the company.

Customers include the actual customers as well as partners.

Technology covers the application domain specific factors and technical factors related to programming languages, design technics, protocols and software architectures.

It is good to point out that customers and technology categories are actually defined by the strategy. Here they are however treated as separate categories to make the structure simple.

More specific factors that describe the characteristics of each category were worked out by the participants. Each category had four to five major factors relate to it. Naturally some of the factors can be related to more than one of the categories. Relations with factors and categories are visualized in a mind map style figure, which is close to the way they were presented in the workshop (see figure 2.2).

Strategy

Strategy was described by five factors in the workshop: Product focus, enterprise software, global competitors, development for the current markets and future proof solution. Next these factors are explained shortly.

Software product focus differs significantly from contract based software project focus. The same software is sold to multiple customers without or with only little customization. Profits come from duplicating the once developed product to many customers.

Enterprise software means the type of software product the company develops. Hoch Detlev divides software into three different categories in the book *Secrets of Software Success*(Detlev et al., 1999): mass market software, enterprise software and professional services. Software developed in the case organization fits well into the definition of enterprise software. Naturally, this type of software leads to few bigger customers. And this leads to the situation where value of every customer is considerable.

Global competitors means simply that when it chose to go for international markets the case organization decided to challenge the global players as well as the local ones in each country.

Development for the current markets summarizes the basic decision about how far to the future development is done. For the case organization, the effect is that products are developed for current markets, step-by-step with the customers. The alternative would be that products would be developed for future markets for example with technology that is currently under development. The choise made minimizes risks and enables immediate revenue from the customers.

Future proof solution is another view to the previous factor. Even if product is developed for the current markets it has to be future proof. The new technology and other future needs must be easily added to the product or at least they should not make the product useless. This point is naturally essential for the venture capitalists behind the company.

Customers

The customers category was described by five main factors in the workshop: Large compared to the case organization, as a development partner, need for ready-to-use product, time related decision and high security demands. Contradictions can be found between factors dealing with the need for ready-to-use products and customers acting as a development partner. These factors both exist at the moment and it is very likely that there will be customers with both motivations in the future as well. Naturally, this not just problem for the organization but a chance to develop the product with some of the customers and provide others with a better defined product a little later.

The *size of the customers* in terms of turnover or number of employees is large compared to the case organization. This causes a situation were the case organization's credibility is continuously evaluated by the customer. The case organization really needs to gain the respect in every customer case, which would not quite be the case if the customer and the supplier were equal in size. A big customer like a national telecommunication operator easily finds it easier to believe in a big player like Microsoft than a small company like the case organization.

Some of the customers like to be *extensively involved* in the product development. Their motivation is the early access to new product versions and possibility to affect features that a new version has. These customers might want to explore the potential of a new market with a new product or a product version. Their demands for exact version definitions are not very high, they would rather have as much new features as possible.

The opposite of the previous customer type are the customers that need the product with exact documentation and description about the features of that product version. Their motivation is mainly to get the product, unpack and install it and make money. The attitude is not to play developer partner with the supplier.

Time related decisions describe the way of dealing with the customers. When a customer is planning to launch a new service for its customers the delivery and product release schedule start to play a very important role. On the other hand, sales persons from the case organization need exact dates about product and product version releases to be used as sales arguments and basis for offers.

High security demands are actually related to both customers and technology categories. It is usually with highest priority in customer's demands for a software product and it has central role in the technology based on the internet as well.

Technology

The technology category was described by four important factors: High rate of change, not well established, no prior experience and complex environment plus need for integration.

High rate of change describes the most important factor of the technology that is related to the case organization and its products. Technology develops fast in the field of programming and architectures including J2EE[6], Microft's .NET[7] and application servers[8], just to mention a few. Development of different network technologies and protocols such as UMTS[9], WLAN[10] and WAP[11] is fast and naturally significantly effects the development of new devices such as mobile phones, PDAs[12] and portable computers.

[6]Java 2 Enterprises Edition (J2EE) is a standard for developing multitier enterprise applications.

[7].NET is Microsoft's platform for XML Web services, the next generation of software that connects our world of information, devices and people in a unified, personalized way.

[8]Application server (for example BEA Weblogic or IBM WebShere) is a platform for running web applications.

[9]UMTS stands for Universal Mobile Telecommunications System. It is a part of the International Telecommunications Union vision of a global family of 'third-generation' (3G) mobile communications systems

[10]WLAN is a wireless local area network, wireless equivalent of widely used local area network.

[11]WAP is a wireless application protocol used in mobile devices mostly in Europe

[12]PDA stands for personal digital assistant, which actually is nothing but pocket size

Not well established describes the situation with the current solutions in the field where the case organization offers its products. The environment where software is run is still very heterogeneous and way in which these services should be operated is not clear. This means that customers do not necessarily quite know from start what they want and how they want it to work. This and the next factor are due to the fact that information technology and telecommunication technology and business are emerging and creating new possibilities and challenges.

No prior experience is available about the technology used for the case organization's application domain. There are no or just a few experience reports on how new technologies should be used in different situations. On the other hand there are not even professionals that would have long experience with these technologies available. The difference between for example an accounting software on PC and the case organization's software product is remarkable when comparing the knowledge available on the application domain and the technologies that can be used to implement it.

Complex environment and need for integration describes the demands for communication with the other systems. Since the case organization's products connect to customers' server environment, end-user organizations' information systems and end-user mobile devices, it is evident that this has a great effect on the software. Customers' server environment itself provides great variance in supporting systems like databases and network management systems. Integration needs arise when customers want to use the new service with their existing customer care and billing systems and when the customer already has a service concept into which it wants to integrate the new service.

Organization

Organization was not discussed too widely in the workshop. However, the three most important factors about the organization were clear: continuous lack of resources, relative young and well educated personnel and large portion of part timer workers in product development.

Continuous lack of resources is probably not a unique problem in this field of business. To use all the resources efficiently, balancing between different tasks and projects is everyday life in most companies nowadays.

Relative young and well educated personnel in the case organization means that the average age is well under 30 years. Almost all of the employees have or are in the middle of getting their master's degree.

Large portion of the product development team works only *part time.* About 35 % of the team works only fifteen to twenty hours per week.

computer.

2.3.4 Needs for the Product Development

Needs related to the case organization's operational environment were worked out in the workshop simultaneously with the environment factors. In this document they have been separated to a different section to make the structure clear. The needs are presented in the relation with a specific environment factors category by category.

After the workshop it was evident that needs for product development worked out in the workshop could not be used as requirements for software development process without modifications. There were two main reasons for this: too many detailed needs and needs concerning both products and their development. Needs that consider more products than development process are translated into needs for development.

Strategy Related Needs

Strategy related needs and their relation to different factors are visualized in figure 2.3. Next, every need related to strategy is briefly introduced with its relation to a environment factor.

That type of the software results in three needs: customizable product, easy management and need for usability. When targeting enterprises that will provide service to their customers using a product, they most likely want to brand for example the user interfaces to fit their own public image. Easy management of the products is essential for the enterprise to provide its customers with good quality of service and to manage its products efficiently. Usability is naturally important to all products that are used by people. Here usability is a key need since customers seem to compare competitive products' usability from their customers' point of view and use the results as decision making arguments.

Product focus introduces three needs: minimized delivery projects, 'standard' integration and partners for integration and support. When selling products, the profit comes from duplication of the product to multiple customers. Essential in this model is that delivering the product to the customer needs to be easy. It has been seen that integration needs of the customers are similar. This brings up a need for 'standard', productized integration package. To be able to concentrate on product development partners are necessary for making the integration and providing the first level support.

The same global competitors are present in almost all cases where proposals are made. Thus knowing the competitors product gives valuable information about your own product and its strengths and weaknesses. Benchmarking against the competitors helps to learn from the own product as well. Sun Tzu (Sun-Tzu, 1983) once said: *"Know your enemy and know yourself; in a hundred battles, you will never be defeated"*, which is still a valid attitude towards the competitors.

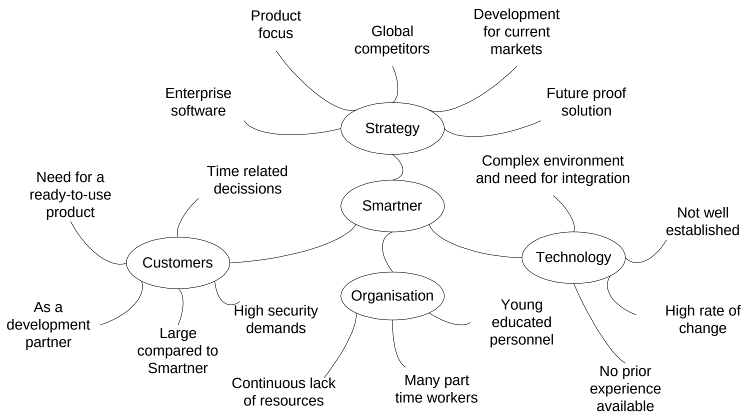


Figure 2.2: Operational environment of the case organization

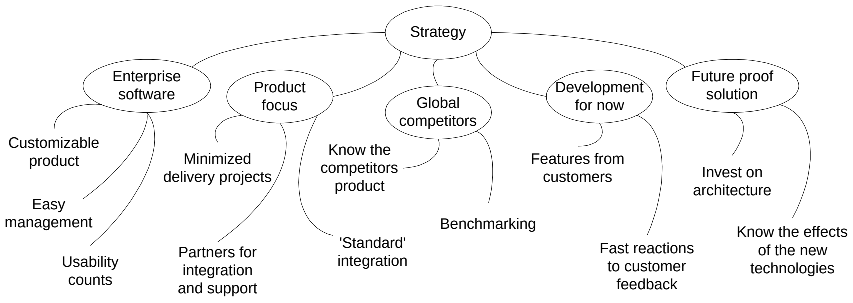


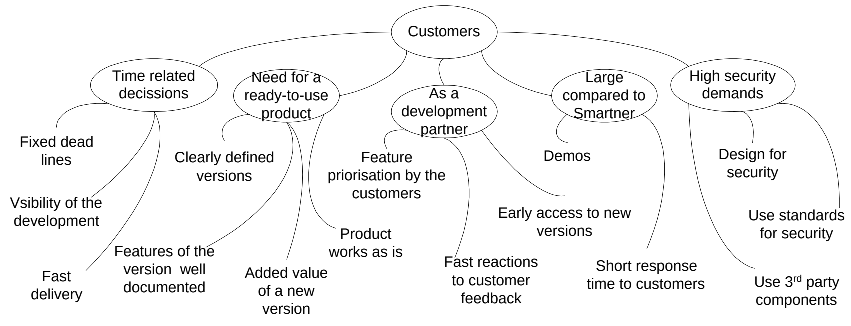
Figure 2.3: Strategy related needs

Development for now and not for the future raises two important needs: features come from customers and fast reactions to customer feedback is essential. Who would better know what the customers want in the close future than the customers themselves? So cooperation with the customers to define new product versions and fast reactions to customers' feedback about the features under development makes current customers happy. This will most likely help in getting new customers.

A future-proof solution needs two main things. Investing on architecture that is able to absorb new technologies and products is essential for fast development and continuity. Secondly, knowing how new technologies will effect the product is important. This is valuable in finding possible problematic areas with a new technology and in exploring the possibilities a new technology introduces.

Customer Related Needs

Customer related needs and their relation to different factors are visualized in figure 2.4. Each of the factors related to customers introduced two to four needs for product development.



Customers
Time related decissions
Need for a ready-to-use product
As a development partner
Large compared to Smartner
High security demands
Fixed dead lines
Vsibility of the development
Fast delivery
Features of the version well documented
Clearly defined versions
Added value of a new version
Product works as is
Feature priorisation by the customers
Fast reactions to customer feedback
Early access to new versions
Demos
Short response time to customers
Design for security
Use standards for security
Use 3rd party components

Figure 2.4: Needs related to customers

Time related decisions brought up three concrete needs: fixed dead lines, visible development and fast delivery. These things had been seen valuable in getting deals already and there had been no sign of them still not being important. Fixed deadlines in development projects makes it easier for sales to communicate with the customers about the dates of delivery. The features included in a specific version was seen flexible if the deadlines are met. Visible development makes it easier for managers to see where development projects are. Seeing a new version every three months is not enough, concrete visual state of the project is needed at the weekly or at least the monthly level. When it comes to making a deal it seems that the customer is always in a hurry. Fast delivery of the product can be the key factor in getting a deal. This actually leads to demands for resourcing product development and the delivery project.

The need for a ready-to-use product raised four needs for the product development: clearly defined versions, well documented features by version, value added by a new version is clear and product works as it is. It is important for a customer to know the version they have. When they buy a product they expect a stable set of features that do not change. This has been one of the problems in the case organization with recent customer cases. Features of a version need to be documented clearly. It must be clearly seen if a certain feature is part of a version or not. This is especially important in situations where new features are creeping into product after product version release. For Sales the difference between versions is important. Value added by a new version has to be easily seen to make an argument for a customer to upgrade their product. The product also needs to work as it is. Customer must be able to set it up and run it without extensive configuring and integration. This is valuable especially with customer pilots.

Some customers act as development partners. Needs related to these customers differ substantially from the customers that want a ready-to-use product. Needs related to development partner type of customers are: feature prioritization by customer, fast reaction to customer feedback, early access to new versions. The customers that are actually taking part in the development are valuable in prioritizing features for new product versions. Reactions to feedback from the customers is essential to keep them involved in the development. Development needs to be done in such way that selected customers can have early access to a new product version. This early access is widely called beta testing.

Customers being large compared to the case organization leads to some needs. Key points in earning credibility and trust in the eyes of the customer are short response times to the customers and demonstrating a working product fast. Response time has proved valuable in the previous cases and is a good way to prove that the company really wants to do this case and is going to do its best. Demonstrating the product is a chance to prove superiority of own product as well.

High security demands result in three main needs: Design for security, use standards for security and use 3rd party components. The most important need is to keep security demands in mind always when designing software. Using standards for security is important because one needs to rely on well known and accepted technologies like PKI[13] and crypting algorithms like TLS[14] to be credible. When the organization's core competence

[13]Public Key Infrastructure (PKI) is the term generally used to describe the laws, policies, standards, and software that regulate or manipulate certificates and public and private keys.

[14]TLS stands for transport layer security, which is a protocol that provides security to application protocols.

is not implementing security components, 3rd party component should be used as widely as possible. This makes the solution more credible and saves resources to core product development.

Technology Related Needs

Technology related needs and their relation to different factors are visualized in the figure 2.5. The most important and most stressed factor out of four technology implicated factors was the high rate of change. The other three factors are different views to the same thing: new and old technology and systems are challenging to combine.

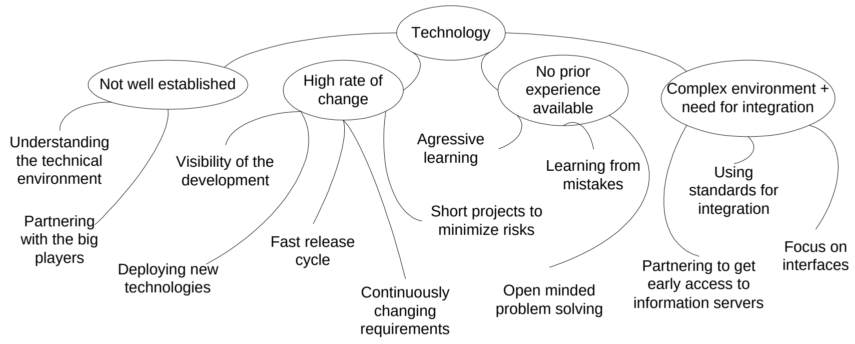

Technology
Not well established
High rate of change
No prior experience available
Complex environment + need for integration
Understanding the technical environment
Partnering with the big players
Visibility of the development
Deploying new technologies
Fast release cycle
Continuously changing requirements
Agressive learning
Short projects to minimize risks
Learning from mistakes
Open minded problem solving
Using standards for integration
Partnering to get early access to information servers
Focus on interfaces

Figure 2.5: Technology related needs

High rate of change in technology was considered one of the most important factors affecting the case organization. It introduced five main needs: fast release cycle, visibility of the development, deploying new technologies, continuously changing requirements and short development projects. Fast release cycle was needed for the following reasons: When the release cycle is short enough the main product vision does not probably need to be changed during the project and on the other hand new features possibly taking advantage of new technology can be delivered relatively fast to the customers. Visibility of the development becomes important in a dynamic environment where requirements change continuously and development projects need to be steered according to those changes. Product development needs to be able to continuously handle changing requirements efficiently to stay productive. Short development projects are needed to minimize risk. Short projects are easier to manage and adding features incrementally seems to help the product better together. The importance of managing this high rate of change is well summarized in the article *Bringing Descipline to Project Management* (Elton and Roe, 1998): *"When it comes to products for the internet and other fast-emerging technologies, customers' needs are changing so rapidly that traditional, sequential product-development processes run the* *risk of generating products that are obsolete by the time they are released.".*

"Not well established" introduced two needs. Understanding the technical environment, 'big picture' so to say, is essential for developing a product that can be used in different environments without other modifications than configuring. Secondly, there is a need for partnering with the big players who have provided customers with hardware, software, support and consultancy. Partners whose core competence is to know the environment with its challenges and possibilities can remarkably help with installation and integration projects.

No prior experience available on application domain and technologies raised a few obvious needs: aggressive learning, open-minded problem solving and learning from mistakes. These needs are common to all situations where not enough knowledge is available on the subject in hand. Aggressive learning means effective communication within the team and encouraging everyone to study new subjects. Open-minded problem solving is needed to solve problems that do not have well known and documented solution. Learning from mistakes is one of the corner stones in an effective organization. As the CEO of the case organization once said in his speech to the personnel: *"We are moving fast and that causes us to make mistakes. We all make mistakes but w'd better learn from them".*

Complex environment and need for integration were summarized in three needs: using standards for integration, focus on the interfaces, and partnering to get early access to information systems. Using standards for integration makes it easier to communicate with different systems. For example directory services regardless of the supplier support LDAP[15], the standard way of accessing contact information. Within the products design focus needs to be on interfaces especially in the parts that might be integrated to external systems. Partnering with the information server suppliers to get early access to new product versions is important to be able to take advantage of their new features as early as possible.

Organization Related Needs

Organization related needs and their relation to different factors are visualized in the figure 2.6. Each of the factors resulted in only few needs. This was probably because the focus of the workshop was not in the organizational factors.

Continuous lack of resources naturally leads to a situation where everything just cannot be done at the same time. Prioritizing tasks together with a schedule centric attitude gives tools for surviving with less people. Prioritizing needs to include all the activities that the product development team

[15]LDAP stands for ligh-weight directory access protocol and it is widely used for accessing contact information

does: different product development projects, integration projects with customer and support for older versions. Schedule centric attitude focuses on being on time.

Relatively young and well-educated personnel raises a few needs for product development: flat team organization, divided responsibility and big challenges. Team organization needs to be as flat as possible to delegate the responsibility as low as possible. The development team has been motivated by big challenges.

The organization having many part time workers introduces needs for good communication. Tasks need to be clear and separate for part timers to be able to be effective and on the other hand communication about changes must reach all the team members.

2.3.5 Comparing Needs to Literature

In this section the most important areas found in the workshop are briefly compared to the literature. If a high-level look is taken to the workshop results, two main areas can be pointed out: an unstable and dynamic environment (see definition in the chapter 1.1) and time dependence. The area of unstable and dynamic environment is compared to Ed Yourdon's thoughts in his book *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects* (Yourdon, 1999). The area of time dependence is discussed with Kathleen Eisenhardt's article *Time Pacing: Competing in Markets That Won't Stand Still* (Eisenhardt and Brown, 1998).

Unstable and Dynamic Environment

Ed Yourdon deals with software projects that take place in very uncertain and dynamic environments in his book called *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects* (Yourdon, 1999). Yourdon defines projects he calls 'death march' projects, explains reasons behind them and gives tips to survive those projects. Even if his main assumption is that death march projects are exceptions in orga-

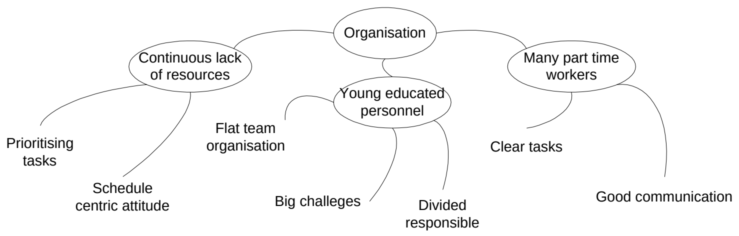


Figure 2.6: Needs related to organization

nization's project culture there could be organizations where death march projects are a way of life. If Yourdon's definition of a death march project are compared to the environment of the case organisation many links between them can be found (see table 2.1).

Table 2.1: Comparison of 'death march' project definition and case organisation's environment

Death march project definition	**Case organization**
Schedule compressed to less than half the amount of time estimated by a rational estimating processes because of the pressures of business competition in today's global marketplace.	Schedule is defined but no rational estimations are made.
The staff has been reduced to less than half of the number of people that would normally be assigned to a project this size and scope.	Situation in many projects when part of the project team is needed for performing other tasks in customer projects and support.
The budget and assosiated resources have been cut in half.	Does not happen too often.
The functionality, features, performance requirements, or other technical aspects of the project are twice what they would be under normal circumstances.	Frequent situation when customers' wishes and requests for features creep into the requirements.

Death march project definition fits well to the case organisation. Also some of the reasons behind the death march projects, such as intense competition caused by globalization of markets and appearance of new technologies, naive optimism of youth and start-up mentality, apply well to the case organisation (see appendix B.1 for all the reasons behind the death march projects). After comparison it is quite fair to say that Yourdon's advice for death march projects can be considered when setting requirements for software development process in the case organisation.

Yourdon introduces the concept of *triage*[16] as the most important thing in the process that helps to succeed death march type projects. In short, triage means prioritizing things so that the most important things are done first and making sure that most important things really are the most important things. This aligns well with the needs that were discovered in the

[16]Triage: 1) A process for sorting injured people into groups based on their need for likely to benifit from immediate medical treatment. 2) A system used to allocate a scarce commodity, such as food, only those capable of deriving the greatest benifit from it.

workshop. Other principal best practices Yourdon offers for death march projects are[17]:

- Formal risk management.
- Agreement on interfaces.
- Peer reviews.
- Metric-based sceduling and management.
- Binary quality gates at the 'inch-pebble' level.
- Project-wide visibility of project plan and progress.
- Defect tracking against quality targets.
- Configuration management.
- People aware of management accountability.

Based on the workshop results we can take almost all of these best practises as requirements for the software development process for the case organization.

The need for visibility (see figures 2.4 and 2.5) relates straight to the binary quality gates and project wide visibility of progress. Binary quality gates mean that the project should have weekly or even daily 'binary' indications of progress. Status of the project must be seen and understood by the whole team, which can be done by making the progress visible.

Good support can make even tightly scheduled project easier. According to Yourdon, configuration management (source code version control) is an essential practice in high-pressure projects. Defect tracking can support projects in scheduling and prioritizing activities. Risk management with defect tracking can act as a powerfull tool to steer a project to the right direction.

Learning from mistakes (see figure 2.5) can be done in many ways. Yourdon suggests using metrics for doing it. This should be considered when defining the software development process for the case organization as well. The importance of the metrics was put in words by DeMarco: *"You cannot control what you cannot measure"*(Demarco and Boehm, 1982).

Peer reviews can be considered important since they are widely agreed to find defects in the early phase of development and on the other hand peer reviews in the form of pair programming forms one of the main principles of extreme programming.

In short, it is reasonable to consider almost all of the key practices listed by Yourdon when setting requirements for the software development process.

[17]Actually Yourdon has adopted best practices from SPMN that is an organization under USA Departement of Defense.

Dependence on Time

Kathleen Eisenhardt describes the importance of managing time and change in her article *Time Pacing: Competing in Markets That Won't Stand Still* (Eisenhardt and Brown, 1998). She describes time pacing as *"a strategy for competing in fast-changing, unpredictable markets by scheduling change at predictable time intervals "*. She claims that companies, no matter how big or small, high or low tech can benifit from time pacing especially in the markets that will not stand still. Even from the description of time pacing it is clear that time pacing could be very useful for the case organization. When comparing the main points of the time pacing and the needs of the case organization the suitability of the time pacing becomes even more evident (see table 2.2).

Table 2.2: Comparison of time pacing and case organisation's needs

Time pacing	**Case organization needs**
Time pacing creates predictable rhythm for change in the company.	High rate of change in the environment of the case organisation.
Transitions can be used for learning, changing direction and for acconmplishing other goals.	Learning from mistakes and changing direction effectively.
Rhythm helps people plan ahead and synchronize their activities.	Organization with many part time workers needs clear task planning and communication.
Time pacing helps managers to avert the danger of changing too infrequently or too frequently.	Short development projects and fast release cycle.

Besides creating a sense of urgency, time pacing stresses two essential processes for succeeding in changing markets: managing transitions (shifts from one activity to the next) and managing the rhythm. Managing transitions is vital because major transitions are periods when the organization is likely to stumbe. Well managed rhythm is in synhronization with the rhythms of suppliers and customers. Rhythm creates the momentum for the organization and helps people to predict the changes. The main points of the time pacing are valuable to consider from the view point of the case organization.

Managing transitions well in the product development can improve the efficiency of the shifting from one product development project to the next (minimizing the 'downtime' between the projects) and shifting from development to the maintenance and support of a product.

Managing rhythm so that organization is synchronized with any of those rhythms that create new opportunities (like release rhythm of the information server suplier) can make organisation to improve its performance.

In short, time pacing seems to fit well to the needs of the case oprganization. Time related needs from workshop got support from the literature and some of the ideas from time pacing can be concidered when defining the requirements for the software development process.

2.4 Summary of the Requirements

This section summarizes results from the workshop and makes comparisons with literature to create one list of requirements for the software development process of the case organization.

Due to the large number of needs derived from the workshop material, needs must be grouped to form the actual requirements. Same kinds of needs from different viewpoints can be found related to different environmental factors. For example needs related to the environmental factors *time related decisions* and *high rate of change* produced same kind of needs. For this reason needs are grouped here according to theme. Needs that belong to a same theme form a requirement for the software development process.

The list of requirements is presented in the tables 2.3 and 2.4 where it has been made clear which of the needs are related to each of the requirements. These requirements are the basis of the rest for the research.

Table 2.3: Requirements for the software development process

Requirement	**Need**
Architecture centric	Customizable product
	Easy management of product
	Invest in architecture
	Divided responsibility
	Design for security
	Use standars for security
	Use 3rd party components for security
	Focus on interfaces
	Using standards for integration
	Deploying new technologies
Time oriented	Fixed deadlines
	Schedule centric attitude
	Fast release cycle
	Short projects to minimize risks
	Partnering to get early access to information servers
	Big challenges
	Flat team organization
	Managing transitions
	Right rhythm
Fast reactions to change	Continuously changing requirements
	Prioiritizing tasks
	Risk management
Customer oriented	Features from customers
	Feature priorization by customers
	Fast reactions to customer feedback
	Fast delivery
	Short response times to customers
	Early access to new product versions
Managed requirements	Clearly defined versions
	Features of the version well documented
	Added value of a new version
	Configuration management
Visible development	Demos
	Visibility of the development
	Clear tasks
	Binary quality gates as the 'inch-pebble' level
	Peer reviews

Table 2.4: Requirements for the software development process

Requirement	**Need**
End-user orientation	Usability is important
Focus on core product	Minimized delivery projects
	Partners for integration
	Standard integration
	Partnering with the big players
Extensive testing	Product work without modifications
	Understanding the technical environment
	Defect tracking
Learning organization	Agresssive learning
	Learning from mistakes
	Open-minded problem solving
	Good communication
Research for the future	Knowing the competitors
	Benchmarking
	Knowing effects of new technologies

Chapter 3

Evaluating Existing Process Models

In this chapter existing process models and frameworks are evaluated against the requirements defined in the section 2.4 (see tables 2.3 and 2.4). First, relevant process models and frameworks are chosen for evaluation. The chosen models are introduced briefly and then more specific evaluation is done. The result of the evaluation is finally summarized into a table where the process models are rated against the requirements. The process models are not described in detail in this research. Only the relevant points are explained to give the reader adequate background. This chapter forms the basis for constructing the new software development model for the case organization.

At this point it is good to note that process models and methodologies are not trivial to compare. Some of the models are more like loose high-level frameworks when the others might be very detailed and concrete. Since models have different levels of abstraction they could even be parts of each other so that one model actually implements a part of the other model. Because models are evaluated against the requirements, not against each other, that should not be a big issue in this research.

3.1 Choosing Models to Evaluate

Software process maturity models and standards, such as CMM (Paulk et al., 1991), SPICE (Dorling et al., 1996), and BOOTSTRAP (Kuvaja et al., 1994), have a slightly different point of view to the software development process than it is needed in this research. They do not necessarily provide concrete enough actions that could be evaluated against the requirements. The other aspect is that these frameworks have been developed at least partly for analyzing the maturity of the software process, not to provide concrete ways of working.

Four software process models were chosen for evaluation: Rational Unified Process (RUP) (Kruchten, 1999), Extreme Programming (XP) (Beck, 2000), Scrum (Schwaber and Beedle, 2002), Synchronize and stabilize (synch-n-stabilize) (Cusumano and Selby, 1998).

RUP was chosen to represent a more traditional view of software development. Even if it is a framework by its nature, it has a great practical value compared to its base framework USDP (Jacobson et al., 1999) [1]. XP was chosen because it is maybe the most hyped and popular agile process model at the moment. Series of XP books have been published after Kent Beck's initial definition of XP (Beck, 2000) in a relatively short time frame. This gives an indication that XP is found interesting by the software developer community. Scrum, another agile process model, was chosen to give XP a point of comparison. Synchronize and stabilize was chosen because it have been seen to work fairly well in the very unstable and dynamic environment of Microsoft.

3.1.1 Rational Unified Process

RUP is a commercial product that is actually a specific and detailed instance of a more generic process framework USDP introduced by Ivar Jacobson (Jacobson et al., 1999). RUP is supported by a large toolset provided by Rational Software Corporation.

In the product white paper (Rational, 1998), RUP is defined to be *"a software engineering process that provides a disciplined approach to assigning tasks and responsibilities within a development organization"*. RUP takes UML[2] in extensive use. Rather than producing large amount of paper documents RUP uses UML models to represent the software product from its definition to installation. RUP is targeted to both small and large teams and organizations. It is configurable for different purposes.

RUP concentrates on deploying commercially proven *best practices* into the software development. Six best practices are listed as being the most important of them:

1. Develop software iteratively
2. Manage requirements
3. Use component-based architectures
4. Visually model software
5. Verify software quality

[1]USDP stands for Unified Software Development Process

[2]The UML is an industry-standard language that allows us to clearly communicate requirements, architectures and designs.

6. Control changes to software

RUP can be described in two dimensions: time and process workflows as visualized in the figure 3.1. In the figure, the weight of the process workflow in each phase is visualized with shaded area.

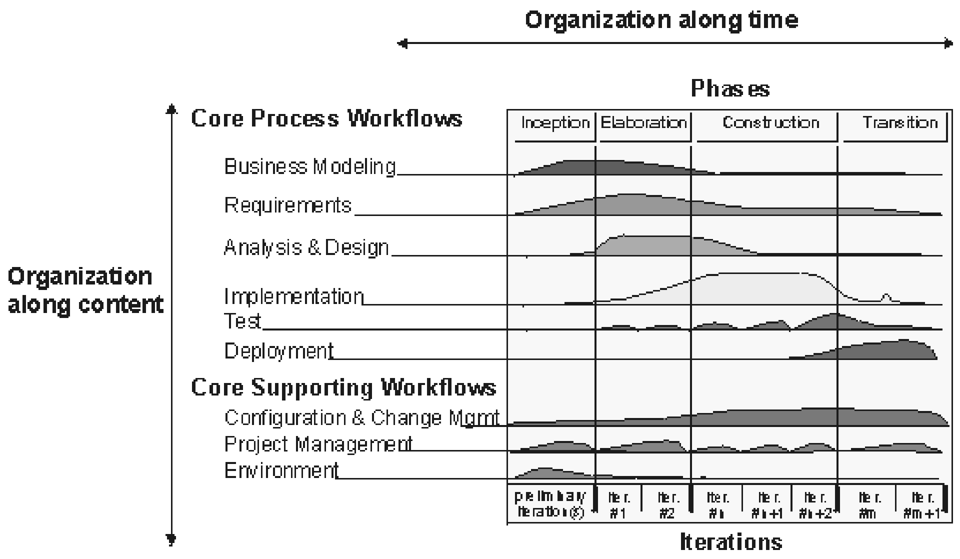


Figure 3.1: The Iterative Model Graph of RUP

The time dimension defines phases and iterations. The phases are called inception, elaboration, construction and transition phase. Each phase consists of number of iterations and ends with a major milestone, a point where key goals have been achieved.

The process workflow dimension defines nine core process workflows. Six of them are considered "engineering" workflows: business modeling, requirements, analysis & design, implementation, test and deployment workflow. Other three of them are "supporting" workflows: project management, configuration & change management and environment workflow.

3.1.2 Extreme Programming

Kent Beck first introduced XP in his book *Extreme Programming Explained: Embrace Change* (Beck, 2000). Since then XP has gained popularity and number of books and articles have been written about it[3]. XP process currently has no tool support but activities carried out in XP are currently supported by number of open source tools.

Kent Beck defines XP to be *"a light-weight methodology for small-to-medium-sized teams developing software in the face of vague or rapidly*

[3]see http://www.extremeprogramming.org for a list of books about XP.

changing requirements". XP's motto is *traveling light and maximizing the produced value to the customer.* Its flexible project scope provides means to control cost, time and quality of the software development project and product.

XP can be described by its values and practices (see figure 3.2). The values of XP are communication, simplicity, feedback and courage. Communication is encouraged by employing practices that cannot be done without communication: pair programming, unit-testing and task estimation in "planning games". System metaphor communicates the whole development team as well as the customer the main story about the software to guide all development. Practices like simple design and refactoring strive for simplicity. The question to ask in XP-style design is *"What is the simplest thing that could possibly work?"* Feedback works in different time scales. Developers get minute-by-minute feedback from their programming pairs and on the other hand from the system by running unit tests. In a time frame of weeks, the planning game gives customer immediate feedback from developers in the form of task estimations. Small releases give the customer and the development team frequently feedback about the quality of the work. Courage follows the other three values. When the first three are in place developers have the courage to strive for simpler and better design by refactoring even if it might result in the software to break for a while. Practices like collective code ownership, pair programming, coding standard and testing give courage to make even radical changes to any part of the code when it is needed. Practices of XP (visualized in the figure 3.2) are done iteratively so that planning games are played once in about three weeks, iteration planning is done few times between the planning games to derive tasks out of the user stories which have been developed in the planning game. Stand up meetings occur about once a day to solve urgent problems of coding, testing and integration.

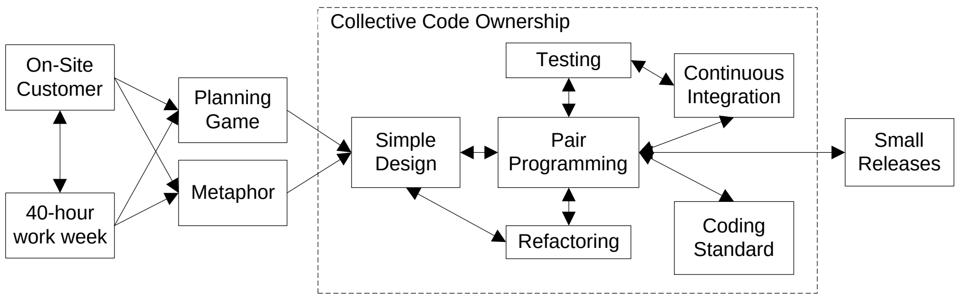


Figure 3.2: Practices of the XP

XP project is managed by coaching, tracking and metrics. Coaching is about communicating and keeping the focus on relevant things such as keeping the design simple, refactoring when needed and running and writing unit testers that run with 100% success. Tracking using metrics helps to keep the project in schedule and makes the team learn from the past. Metrics are to be provided to those who can benefit from them. For example, a metric describing the planned effort against realized effort needs to be seen by the developers who make the task estimations.

Even if XP is considered to be a light-weight method it surely is also a deliberate and disciplined approach to software development when exercised strictly.

3.1.3 Scrum

The term scrum was first used in the field of software development by Hirotaka Takeuchi and Ikujiro Nonaka in their article *The New New Product Development Game* (Takeuchi and Nonaka, 1986) to describe hyper-productive software development in Japan in the year of 1987. Later, the term scrum was adapted by Ken Schwaber and Mike Beedle to give a name to a software development process they were using to build software products and that they used to help other organizations to build systems. To the larger audience Scrum was introduced in their book *Agile Software Development with Scrum* (Schwaber and Beedle, 2002). Originally scrum is a rugby term and Schwaber argues that the name fits for the software process model well since *"Both are adaptive, quick, self-organizing and have few rests"*. Scrum belongs to same category of agile software process models as XP.

Ken Schwaber defines Scrum as *"a management and control process that cuts through complexity to focus on building software that meets business needs"*. Scrum primarily works on the team level so the ideal size of a development team is seven (plus or minus two) members. Scrum advices to divide larger organizations into smaller Scrum teams that may work independently. Scrum being introduced quite recently to the general public it doesn't have tool support.

Scrum concentrates on management and control of development work. It does not define its own engineering practices but is more like a wrapper for existing practices. For example XP practices could be used in a Scrum project. Scrum defines a set of concepts and workflows which are visualized in figure 3.3. Everything starts with *product backlog* that is a prioritized list of all product requirements. The *Product owner* controls the product backlog by prioritizing its requirements that can be originated from various sources (customers, user, sales people, engineers, etc). Development is divided into one-month iterations called *sprints.* A sprint starts with a *sprint planning session* where *sprint backlog* is constructed from the most important items of the product backlog. The sprint ends with a *sprint demonstration* or sprint review meeting that has one important purpose: to demonstrate all new functionalities that have been developed in the sprint.

The importance of the demonstration is stressed in Scrum's motto: *"demo or die"*.

Sprint backlog items are expanded by the team into concrete task list with work estimations. The estimations for task describe how many hours of work are still needed for a specific task. The sum of the estimated remaining working hours is visualized against calendar time in the *project burn down graph*. This graph gives a real-time estimate for sprint completion date (see appendix B.2 for an example and more information about the project burndown graph). Everyday work in a sprint is steered by *daily scrum* lead by *the scrum master*. The scrum master is responsible for the success of the scrum. The scrum master's main responsibility is to help the team to work as efficiently as possible by removing any impediments the team might have and by making decisions. The daily scrum is a short meeting with a standard agenda. Everyone is asked the same three questions to communicate the progress and the possible problems: What did you do since the last scrum meeting? Do you have any obstacles? What will you do before the next meeting?

Scrum assigns great responsibility to the development team: the team is responsible for delivering the new functionality and on the other hand team is free to use any means to make it happen.

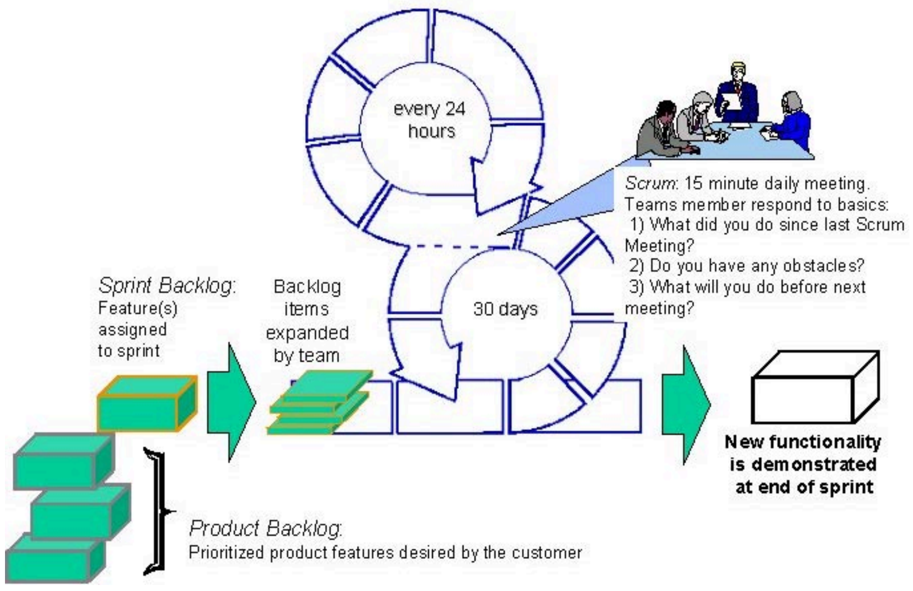

every 24 hours
Scrum: 15 minute daily meeting.
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any obstacles?
3) What will you do before next meeting?
Sprint Backlog: Feature(s) assigned to sprint
Backlog items expanded by team
30 days
New functionality is demonstrated at end of sprint
Product Backlog: Prioritized product features desired by the customer

Figure 3.3: Scrum concepts and workflow

3.1.4 Synchronize and Stabilize

Michael Cusumano and Richard Selby introduced the synch-and-stabilize development approach in their book *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People* (Cusumano and Selby, 1998). Actually synch-and-stabilize describes the software development process used in Microsoft starting from the early 90's. Synch-and-stabilize does not have a similar development community in the World Wide Web promoting the model as XP or Scrum, neither is it a commercial product like RUP. Anyway, Cusumano's and Selby's book has been cited in a number of articles in the software development literature.

Synch-and-stabilize does not have a clear definition. In Cusumano's words *"synch-and-stabilize is Microsoft's approach to product development"*. Even if synch-and-stabilize has been developed for large development organization of Microsoft, one of its principles is to make large organization work like small teams.

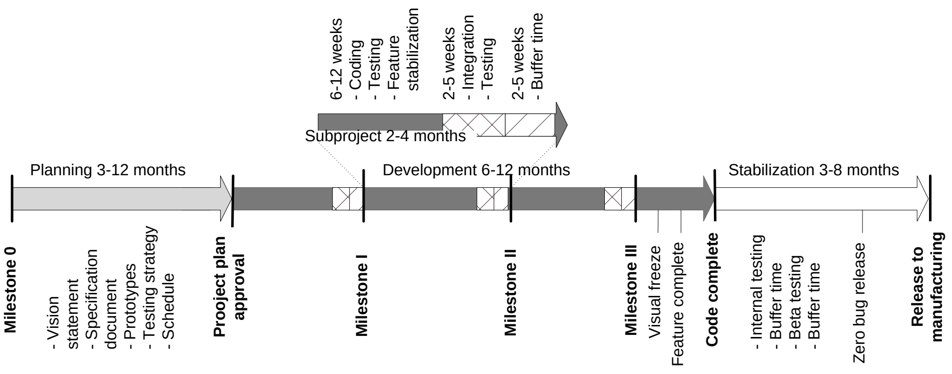


Figure 3.4: Overview of the Synch-and-stabilize development approach

The basic idea of the synch-and-stabilize approach is to continually synchronize the work of parallel working teams and to periodically stabilize the product increments. The overview of the synch-and-stabilize approach (see figure 3.4) visualizes its five principles about defining products and development process:

1. Divide large projects into multiple milestone cycles with buffers and no separate product maintenance.
2. Use a vision statement and outline specifications of features to guide projects.
3. Base feature selection and priorization on user activities and data.

4. Evolve a modular and horizontal design architecture, with the product structure mirrored in the project structure.

5. Control by individual commitments to small tasks and 'fixed' project resources.

The life cycle of product development is formed by three phases: planning, development and stabilization. In the planning phase, the vision statement and specifications including architecture plans are made and features are prioritized. The development phase is divided into three subproject separated by milestones and a stabilization period. In the subprojects the large organization is divided into small feature teams consisting of program manager, 3-8 developers and 3-8 testers (who work parallel 1:1 with developers). Feature teams go through the complete development cycle (implementation, integration, testing and fixing) in each of the subprojects. Subprojects have buffer time for unexpected problems to balance the schedule. Development is governed by the five principles of developing and shipping products:

1. Work in parallel teams but 'synch up' and debug daily.

2. Always have a product you can theoretically ship, with a version for every major platform and market.

3. Speak a common language on a single development site.

4. Continuously test the product as you build it.

5. Use metric data to determine milestone completion and product release.

In synch-and-stabilize approach the project resources are fixed, so that the amount of new features is flexible. The most critical features and shared components are developed in the first subproject and the rest are developed in the next two subprojects in the order of priority. Features may also change from the original set of features during the development phase.

3.2 Evaluation of the Software Process Models

In this section the software process models that were presented in the previous section are evaluated against the requirements. The purpose of the evaluation is to find out how these models support the requirements and to find out the strong areas of each model.

Evaluation of the models against the requirements is done fairly theoretically (see the summary of requirements from section 2.4). The features of the models are taken as they are presented in the literature and evaluation is done based on that information. The reason for this is that objective experience reports are not available equally for the four models under evaluation.

If reference is not explicitly mentioned in this evaluation the information is from the following books

- The Rational Unified Process: An Introduction (Kruchten, 1999)
- Extreme Programming Explained. Embrace Change (Beck, 2000)
- Agile Software Development with Scrum (Schwaber and Beedle, 2002)
- Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People (Cusumano and Selby, 1998)

3.2.1 Architecture Centric

By definition RUP is an architecture centric process model. It stresses the importance of early development of robust executable architecture using component-based technologies. This supports well requirements for focusing on interfaces, dividing responsibilities and gives good basis for using standards and ready-made components.

Defining component interdependency and basic architecture is a very important part of the synch-and-stabilize approach to be able to divide work between smaller feature teams. The details of developing the architecture are not discussed but the main effort for it is done in the first development subproject when common components are implemented and tested.

The architectural skeleton is to be developed in the first iteration of an XP project. The system metaphor should be good enough to define needed border conditions for the architecture. It is evident that the quality of the first architecture skeleton depends on the case and it is very likely that it will be changed even dramatically during the project. Demand for making the simplest possible architecture however gives a good chance to use standards and ready-made components.

Being more a management model, Scrum does not comment how architecture should be considered.

3.2.2 Time Oriented

All evaluated models except RUP introduce fixed deadlines for the development projects. The release cycle naturally depends on the scope of the project but XP and Scrum guide development to fairly short development projects of three to six months. Synch-and-stabilize, on the other hand, aims at one to two year development projects. All these three models not only have a fixed dead line for the whole development project but also for the iterations inside the project. Iterations inside the development project give a chance to refine and modify the requirements of the product under development. This naturally lowers the risk of building something that is not usable by the time the project is finished. Also RUP is based on iterative development but its development approach can better be described as feature oriented than schedule oriented.

Especially Scrum and XP are very time and schedule oriented models from day-to-day practices to whole development project. Scrum even has a daily schedule of steering the project in the scrum meetings. In XP, teams are encouraged to hold daily *stand up meetings* where the direction of the project is corrected if needed. Synch-and-stabilize introduces a concept of *buffer time*. This contingency time is part of each major milestone during development and stabilization. Having a buffer time minimizes the risk that the release date must be delayed because of unexpected problems. This of course gives sales more confidence about promising a delivery of the product to customer at a specific date. Synch-and-stabilize and XP require developers to make daily builds to keep the state of the software visible at all times.

Synch-and-stabilize, and especially XP and Scrum, push the responsibility as low as possible. Organization in XP and Scrum projects is considerably low, giving every team member a great chance to affect the project result. This creates a challenging environment for the whole team since the goal is clear but the means to achieve it are up to the team and individuals. Scrum promotes self-organizing teams which are said to maximize the value produced by the team by giving the team a possibility to divide tasks in the best possible way for that particular group of people. In RUP the structure of the team depends on the way RUP is implemented for that specific project. Anyway, the basic attitude seems to be top-down management of projects that might limit the challenges given to an individual programmer.

The right release schedule, the rhythm, is essential in synch-and-stabilize. New product releases must be in synch with other products as well as with new hardware releases. Of course, Microsoft has much better possibilities to affect other software and hardware vendors' schedules than a small software company. In Scrum the rhythm is clear and predictable, development goes on sprint by sprint in fixed schedule. If this schedule is synchronized with relevant events like exhibitions and other software releases this can be an effective approach. But Scrum does not actually assume one to do this synchronizing.

Transitions are considered at least in some way in Scrum and synch-and-stabilize. In both models development of the product does not stop when a release is made but it continues smoothly to development of the next version. Transition is a moment of looking back, gathering feedback from the project and starting planning for the next version. Synch-and-stabilize introduces *milestone 0* which is the time between finishing one version and official start of coding the next version. During this time code is cleaned up from things that developers felt were wrong. In the larger picture where one product is not developed to the next version or when the development of a completely new product is started is not discussed in detail even in these two models.

XP and RUP have a more or less project view to rhythm and transactions. They consider the development project, not the time between the projects or how to place the project in the calendar.

3.2.3 Fast Reactions to Changes

All the models accept the fact that requirements will evolve during a development project. Iterative development in all models gives a chance to change requirements periodically. RUP has clearly more formal and time-consuming approach to the change management than XP, Scrum or even synch-and-stabilize.

The vision statement in synch-and-stabilize, system metaphor in XP and sprint goal in Scrum are means of setting the direction for the development project and basis for making decisions about changes in requirements. Requirements can be changed if it leads the project to the right direction. Especially if requirements change very frequently adopting the changes is not the only important matter. The development team needs to have a certain period of stability to be able to be effective in their work. If tasks are changing on a daily basis nothing will be completed but lots of things are left unfinished. Fortunately all the models advice projects to concentrate on a stable set of requirements for a certain period of time and to apply changes in the beginning of the development period.

Prioritizing features and tasks is an essential point in all the models. RUP prioritizes high-risk components to the top for every iteration while synch-and-stabilize prioritizes the features based on user activity and data (how well a feature supports important or frequent user activities based on research data). XP and Scrum prioritize features and tasks according to the value that these features produced to the customer. All the models reprioritize features for each iteration. The time frame for doing it is shorter in Scrum and XP than it is in RUP and synch-and-stabilize.

Risk management, i.e., identifying and attacking the highest risk items early in the project, is one of the ten most essential precepts of the RUP as described in the Rational Software white paper *The Ten Essentials of RUP* (Probasco, 2000). The other three models do not address formal risk management but manage risks by very frequent project reviews and short project steering cycles.

3.2.4 Customer Oriented

The requirement of on-site customer in XP makes it the best model when customer orientation is considered. It might be unrealistic to assume that the customer really is available for the development team at all times. If the on-site customer concept works the commitment of the customer and possibility to affect the project and its priorities are at their maximum level. It also gives the project team the possibility to react fast to customer feedback.

The source of the features is explicitly defined in XP and synch-and-stabilize. The on-site customer defines features and requirements in XP while user researches, feedback and usage data is the basis of the feature selection in synch-and-stabilize. Scrum states that features and requirements that are gathered to product backlog are from various sources like customer support and feedback, sales and technology people. RUP does not give a clear picture of how features and requirements are defined when developing a product but in a tailored project they are defined at the start of the project with the customer.

The customer is explicitly involved in feature prioritization in XP and synch-and-stabilize while Scrum involves the customer implicitly through the product owner who does the actual priorization. The amount of customer involvement in prioritization of product backlog in Scrum is largely depended on the attitude of the product owner. Synch-and-stabilize bases prioritization on quantitative user activity data which is a natural approach with mass market products.

Fast delivery is not explicitly discussed in any of the models. In synch-and-stabilize products go through extensive testing including internal and external (beta testing) testing in the stabilization phase. This strives at *zero bug release* and complete product packaging that makes the fast delivery to the customer easier. Naturally, this approach is important for shrink-wrapped products that are available from almost any software store. Continuous integration, early production and presence of the customer should make the delivery time short in XP as well.

3.2.5 Managed Requirements

Requirements management is one of the core process workflows in RUP. It introduces a formal way to capture, organize, document and track requirements from specifications to delivered product. Scrum on the other hand has a very light-weight way of managing requirements: backlogs have short requirement descriptions and are the basis of all work. They are kept up-to-date throughout the project. Synch-and-stabize uses a similar requirement specification documentation that has feature specification in a little more detail. This can make the specification fairly large with complex programs like Microsoft Office (Excel 5.0 had initial specification of 1500 pages). XP manages requirements in user stories in a fairly informal way. In short, one can say that only RUP has extensive requirement management support that takes different product versions and variants into account.

Communicating the added value of a new version to the customer is not discussed in any of the models. In XP the value produced for the customer is the main argument of prioritization so that it is assumed the that value added is clear. In Scrum and synch-and-stabilize prioritization should make the value added clear but communicating the real points to the customer is left unclear.

Configuration management is recognized as one of the core supporting workflows in RUP. It provides guidelines for managing multiple product variants and parallel development of a product. Daily builds and tests in synch-and-stabilize and XP rely on effective configuration management. However, a clear picture of how to perform configuration management is not presented.

3.2.6 Visible Development

Visibility of the development is one of the key issues in XP and Scrum: both encourage communicating the state of the development daily within the project and at least weekly with the customer. Synch-and-stabilize and XP provide valuable insight to the state of the development by daily builds and automated daily tests. These tests and builds can be used as 'inch-pebble' level quality gates that ensure the project to go to the right direction.

The iterative approach to the development adopted by all the four models give good visibility to the development. At the end of the iteration all models stress that software needs to be basically ready for shipping. At least new features developed in the previous iteration must be demonstrated as a part of the whole software product. The most considerable difference between the models from this point of view is the length of the iteration that varies from XP's and Scrum's few weeks to few months of synch-and-stabilize. RUP, instead, gives freedom to choose the length of the iteration and does not stress the importance of frequent project status updates.

Clearly defined and estimated tasks are important in XP and Scrum. Day to day work is defined by the tasks and the progress of the project is estimated through the tasks. Tasks are defined and estimated in both models, as well as in synch-and-stabilize, by the developers who actually implement the tasks. RUP, instead, as a framework, does not describe how tasks should be defined or estimated but leaves that for the one who implements the model to decide.

Peer reviews are taken as one of the corner stones of XP. Actually work is reviewed by a peer continuously in pair programming. Other models do not explicitly comment if peer reviews should be used or not. In synch-and-stabilize limited peer reviews are done on design and code. However, it is not enforced and in practice the amount of reviews depends on the project schedule. Scrum and RUP do not instruct whether to use peer reviews or not.

3.2.7 End-user Oriented

Synch-and-stabilize is the only model out of the evaluated models that addresses usability testing as part of the development process. Usability is built into the product during the whole development project. In the planning phase usability goals are set and exploratory testing is done. In the development phase usability tests are done iteratively to provide feedback for development. Stabilization phase includes wider testing whose results are used in the next version of the product.

Even if usability testing is not an actual part of the other models there is not anything that prevents one to include them as a part of the development or testing tasks.

3.2.8 Focus on the Product

Focusing on the chosen products is a higher-level product development strategy decision and it is not really discussed in the models evaluated here. Actually, this kind of focus is planning the product releases of all the products in a longer time scale. Only synch-and-stabilize takes this long-term product planning into account.

Some other things that may affect to the core product focus by the process are the simplicity of the product, usage of component architecture (and 3rd party components), product documentation and using standards in development. RUP and synch-and-stabilize have their strengths in architecture and component centric design (see section 3.2.1) as well as in documentation.

3.2.9 Extensive Testing

Testing plays a key role in especially XP but in synch-and-stabilize as well. XP's principles of *"test first"* and *"unit test must run 100% successfully"* reflect well its attitude towards testing. Automated and frequently run unit tests control the development project in both XP and synch-and-stabilize. Successfully done testing on all major platforms (hardware and software) is also a goal of iteration in synch-and-stabilize. Also principle of having a 1:1 ratio of developers and testers in the development team stresses the importance of testing. RUP has testing as one of its core process workflows. Testing is done throughout the development project in a systematic way.

Defect tracking guides development in synch-and-stabilize. Defect tracking tools are used for recording and tracking the status of the bugs. As a guideline, a developer that has more than ten open bug reports is not allowed to start developing a new feature. In XP failed unit tests are visible for the whole team but how defects should be managed when found in acceptance testing or production is not described. RUP treats defects similarly as change requests, they are recorded and managed with a defect tracking tools which RUP supports well as can be expected from a commercial process suite.

Understanding the technical environment is necessity for performing the right kind of tests. This is assumed in all three models that focus on testing. Scrum does not give any guidelines for testing.

3.2.10 Learning Organization

Learning from the past and present projects and products is one of the principles in synch-and-stabilize. It is done by arranging project postmortem sessions and writing postmortem document on basis of it. Feedback and improvements are also encouraged by using metrics about the projects and benchmarks about the best practices and project experiences. Scrum introduces sprint review session after each sprint to ensure that the project team communicates to the customer and to the rest of the organization the new features and the status of the project as well as to ensure that the project team gets feedback from them. This feeback is not even semi-formal and it is not recorded for later like the information gathered in synch-and-stabilize process.

Extensive communication and problem solving in the development team is a fundamental thing in Scrum and XP. Teams have very little guidance but challenging problems to solve. This, with adequate resources (like supporting software and hardware) encourages individual developers to face the problems with an open mind. Also the team working closely together maximizes the information flow within the team and helps to find good solutions to problems.

Learning and communication are not the main themes in RUP but the framework is surely adaptive for introducing for example project postmortems.

3.2.11 Research for the Future

Research for the future is not considered in any of the models. The focus in the models is more on the situation in hand and on how to develop a new product or version. Knowing the competitors and benchmarking the product is considered to be more the job of the business intelligence than product development.

Long-term effects caused by the new technologies are not considered either. Microsoft's general strategy towards the future technologies is described in *Microsoft Secrets* (Cusumano and Selby, 1998) but it does not strictly relate to the development process. Even if Microsoft tries to affect and make the standards of new technologies themselves that is not probably an option for small company.

On the other hand, successful usage of standards and component architecture is more likely to help adapting new technologies than an ad hock architecture. From this point of view RUP and synch-and-stabilize (see section 3.2.1) provide a chance to success with new technology adaptation.

3.3 Summary of the Evaluation

This section summarizes results from evaluation of the existing software process models based on previous sections (see sections 3.2.1 - 3.2.11). The scale used for comparing the models based on the evaluation has four levels. The levels give a general picture of how well the model supports a specific requirement.

• • •	Indicates that requirement is one of the main focus areas of the process model and it is described in detail.
• •	Indicates that the process model supports that requirement.
•	Indicates that requirement is described shortly in the process model but it lacks at least some relevant information.
-	Requirement is not supported or even mentioned in the process model.

It is important to notice that level given to a model is not unambiguous. Since one requirement is a collection of needs that have a same theme the model might support some of the needs but not all of them. This leads to the fact that levels give a qualitative estimate, not an exact value for comparing how well the models support a specific requirement. However, this qualitative estimation is sufficient for this research since no decisions are based only on these estimations and it gives a visual overview of the models compared to the requirements.

To summarize, it is clear that none of the models is perfect if all the requirements are considered. The biggest lack of support is related to the requirements that cope with long-term planning and research as well as business intelligence and partnering (*focus on core product* and *research for the future*). Product customization and delivery was not discussed in any of the models either. On the other hand, almost every requirement but the ones just discussed has at least one best rating (• • •). This leads to an interesting conclusion that combining the evaluated models could possibly support these requirements if the models themselves support additive utility. Combining the models is discussed more in the next chapter where the new software process model is introduced.

Table 3.1: Evaluation of the process models

Requirement	**RUP**	**Synch**	**XP**	**Scrum**
Architecture centric	● ● ●	● ● ●	●	-
Time oriented	●	● ●	● ●	● ● ●
Fast reactions to change	●	●	● ● ●	● ● ●
Customer oriented	●	● ● ●	● ● ●	●
Managed requirements	● ● ●	●	●	●
Visible development	●	● ● ●	● ● ●	● ●
End user oriented	-	● ● ●	-	-
Focus on core product	●	● ●	-	-
Extensive testing	● ●	● ● ●	● ● ●	
Learning organization	-	● ●	● ●	● ●
Research for the future	●	● ●	-	-
Average	● ◔	● ● ◔	● ◐	●

Chapter 4

The New Software Process Model

In this chapter the new process model is introduced. The new process model is based on the requirements derived in chapter 2 and results from evaluation of existing process models against those requirements (see the summary of requirements and evaluation from sections 2.4 and 3.3 respectively). Besides these border conditions some findings and experiences from literature are used. Especially, early results from SEMS[1] research project are used since participation of the case organization to that project has had a great influence on the resulting framework called 4CC[2] (Rautiainen et al., 2002).

The 4CC framework aims at combining business and process management through four cycles of control. This approach is a valuable complement to the evaluated models that have their weakest points in including the business and strategic view to the product development. The four cycles of control in the 4CC framework are:

Strategic release management the main purpose of which is to plan release cycles and content, role and timing of individual release project.

Release project management the main purpose of which is to make sure that assigned product release gets done.

Iteration management the main purpose of which is to build a stable, working product in increments.

Mini-milestones the main purpose of which is to integrate and synchronize the efforts of individuals and teams in the development project.

[1]SEMS research project develops a Software Engineering Management System for small and medium-sized software enterprises. The SEMS project is planned to span over three years (2001-2003) and it is part of the TEKES technology programme SPIN (http://www.soberit.hut.fi/sems/english).

[2]4CC stands for four cycles of control.

At the time being, the 4CC is still a tentative framework lacking some details and important issues like usage of metrics.

Combining existing process models to achieve synergy in certain situations is not a new idea. Rational provides two whitepapers (Smith, 2001) (Pollice, 2001) describing how RUP can be implement so that it uses engineering practices from XP. This approach is not too relevant for this research as it is, but some of the ideas might be valuable. The scrum community promotes combining scrum and XP so that engineering practices of XP are wrapped with the management practices of Scrum[3]. This approach is interesting from the viewpoint of this research but articles or experiences about it are not available.

The new software process model is described by first presenting the overall picture of the model and then concentrating on details. The evaluated models are frequently referenced to in this chapter. If the reference is not explicitly mentioned information is from these references:

- The Rational Unified Process: An Introduction (Kruchten, 1999)
- Extreme Programming Explained. Embrace Change (Beck, 2000)
- Agile Software Development with Scrum (Schwaber and Beedle, 2002)
- Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People (Cusumano and Selby, 1998)
- A Tentative Framework for Managing Software Product Development in Small Companies (4CC) (Rautiainen et al., 2002)

4.1 Overview

All the models evaluated plus the 4CC model have an iterative and sequential approach to product development. This approach has also been found to be good for developing a high-quality product in very uncertain and dynamic environments (MacCormack, 2001). Considering the fact that this approach was also found to support the requirements in the evaluation of the models (see section 3.2.2), it is reasonable to take this approach as a basis for the new software process model.

4.1.1 Leadership and Management

Leadership and management are two different aspects of getting a team or a company to reach its goal. John Kotter describes their fundamental difference in his article *What Leaders Really Do* (Kotter, 2001): *"Management*

[3]See http://www.controlchaos.com/xpScrum.htm for more information.

is about coping with complexity. Leadership, by contrast, is about coping with change". In the uncertain and dynamic environment of the case organization, the role of leadership needs to be understood. Since change is a fundamental part of the product development it cannot just be managed but also lead.

Kotter points out differences in activities of management and leadership. At a high level, companies are managed by *planning and budgeting* while leading an organization starts by *setting a direction*, a vision for the future. Management enables achieving the plan by *organizing and staffing* while *aligning people* towards the vision is essential in leadership. When it comes to everyday tasks, management ensures accomplishment of the plan by *controlling and problem solving*. On the other hand, leadership requires *motivating and inspiring* in achieving the vision. Especially in an unstable environment, a well-led organization has much greater chances to succeed than a strictly managed one. Kotter argues strongly for possibilities of an organization that has an active leadership culture.

The new model lays down a basis for product development that is both led and managed. Empowering the individuals to work for the common goal is its main purpose. From the leadership point of view strategic release management sets the direction for the whole development. Release management cycle and sprints are about aligning people towards that common vision and after all, daily rhythm is about motivating and inspiring.

Product development is creative work in a constantly changing environment. Strict control structures cannot work in a place where individual decisions and new ways of thinking are needed for surviving. Kotter illustrates this situation with an apposite military analogy: *"A peacetime army can usually survive with good administration and management up and down the hierarchy, coupled with good leadership concentrated at the very top. A wartime army, however, needs competent leadership at all levels. No one yet has figured out how to manage people efficiently into battle; they must be led"*.

Combining management and leadership is essential since, as Kotter says, strong leadership with weak management can even be worse than the reverse.

4.1.2 Periodical Approach

Different activities in software product development have a different time scale. Strategic planning of product releases has obviously a far longer cycle than for example daily builds. Nevertheless, all these activities are sequential: they start at one point, they go on for a period of time, they end at some point and after that start all over again. The activities might have a little different content every time but the fundamental goals are fairly stable. For example, the content of a release project surely changes from one project to another but the goal stays the same: high-quality software. This sequential flow of development also gives a good chance to improve working by learning from the previous cycles. Different development activities need to be aligned conveniently to get the right rhythm for the development and to reach the business goals of the company.

To illustrate the different periods of activities and their relations one can think in mathematical terms. Activities can be described with $sin(ft)$ functions with different periods. Aligning these functions so that they intersect at the start of the period illustrates the main point of interaction between these development actions. By bounding a function with a shorter period with a function having longer period we can illustrate that the activity with longer period governs the other activity. Naturally, the

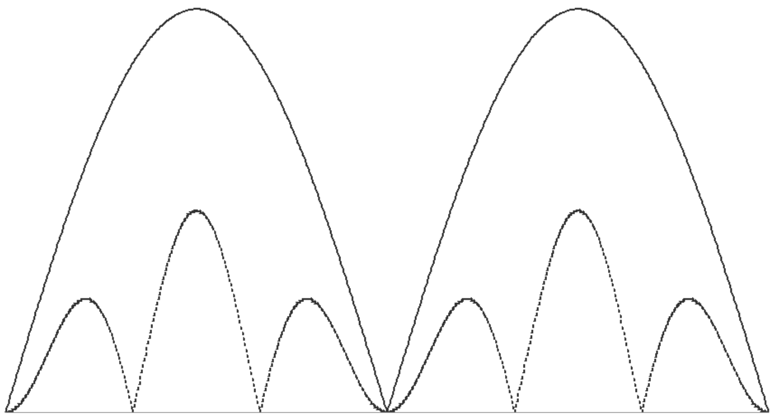


Figure 4.1: Illustration of two development activities with different periods

points where the function hits the time-axis illustrate the start (as well as the end) of the activity. This illustration is visualized in figure 4.1 which could for example represent a time period of six months having two release projects each having three iterations. Functions are defined by $f(t) = |0.5sin(t/3)|, g(t) = |sin(t)|f(t)$ where the absolute value has been used to simplify the figure.

The purpose of the periodical approach focuses on getting the right rhythm for the development, aligning different activities and continuous improvements. This approach has mainly been adopted from Kathleen M. Eisenhardt's article *"Time Pacing: Competing in Markets That Won't Stand Still"* (Eisenhardt and Brown, 1998) that was discussed in section 2.3.5. It steers the model to fulfil the requirement for *time oriented* process.

4.1.3 Main Concepts and Alignment

The main concepts of the new process model are cycles of different development activities. The cycles have different period, purpose and amount of details. Efficient alignment of the cycles is important to enable fast steering of the development and extensive learning from the previous experiences.

4CC model is used as a high level framework for the new model. Terminology is slightly changed to stress the involvement of the other models even if the main cycles are similar to the ones in the 4CC model. On the other hand, elements of the synch-and-stabilize model can easily be mapped to these cycles as well. The management concepts of Scrum are relevant in shorter-term cycles where terminology has been adapted from it. The main concepts and their purposes are:

Strategic Release Management is for setting the direction for the product development. It is about how to align product development with the business and technology strategy by setting the release project schedule for products and the main issues involved in each of them. In this cycle the focus is on long-term issues like starting to develop a completely new product or dropping a non-core product out of development.

Release Project Cycle is for developing a new product or a new product version. Product vision guides development done in the sprints. Product vision is a synthesis from customer feedback, market researches, product roadmap and possibly other things. The schedule and main themes are also defined for the sprints in this cycle. Release Project Cycle produces a new product or product version.

Sprints produce a stable increment for the software product. Sprints have different themes such as architecture and stabilization that clarify the main target of the sprint. Sprint always ends with a sprint review session where the new features are demonstrated and feedback is gathered from the results and from the sprint project. Respectively, sprint always starts with a sprint planning session where features to be developed are selected from a prioritized list and development tasks with work estimations that are derived from the features.

Daily Rhythm is for steering and motivating the development towards the sprint goal. Work of different team members is synchronized and the product is integrated and tested constantly.

Supporting Actions are continuous activities that support and guide development. Their main purpose is to provide efficient methods and tools for configuration and requirement management, defect tracking and daily builds and tests.

Figure 4.2 visualizes the main concepts of the new model and their alignment. The visualization of the model has been produced in five steps:

1. Drawing different cycles of the model on a straight time axis as in figure 4.1.
2. Aligning the phases of the cycles.
3. Cutting a 12-month period out of the time axis and making it a circle.
4. Changing the curve of strategic release management to two circles placed on the starting points of that phase to simplify the picture.
5. Adding spokes to make the model represent a wheel.

Supporting actions keep "the wheel of development" running from one phase to another in a continuous feedback and steering loop.

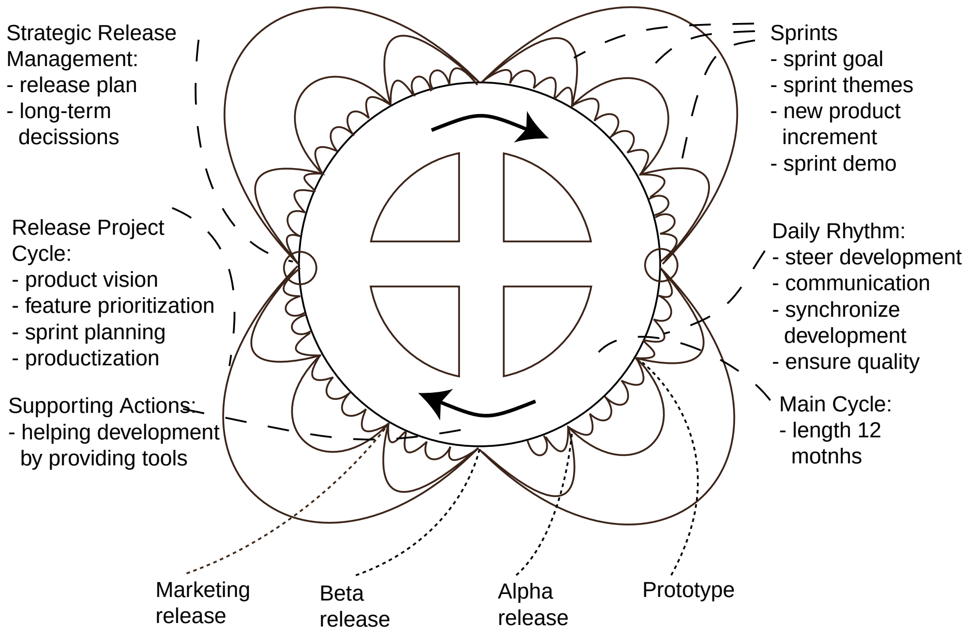


Figure 4.2: Overview of the new process model

The main idea of the alignment is that the cycle with the longer period defines goals for its inner cycle and the inner cycle gives feedback to the outer one. Strategic release management has the longest period. Its output (plans for product releases) is essentially important to take into consideration immediately in the release project cycle. Therefore it is natural to align these two cycles so that every six months they start at the same time. Similarly, the same output affects development sprints and make it evident that release project cycle and sprints are aligned. This alignment also makes it easy to connect a feedback loop from sprints to strategic release management. In practice, all these three cycles do not start at the same second but within a short period of time that is used for communicating the results and the plans between the people involved in different the cycles.

Different releases are presented for one release project in figure 4.2. For the other three release projects delivered releases are the same, but to keep the presentation clear they are not included in the figure. Different releases correspond to dífferent themes of the sprints starting from an architecture prototype and ending after the stabilization period with a marketing release.

Sales needs a demonstration version of the new product as soon as possible and the delivery project needs a product which has gone through a stabilization phase (Cusumano and Selby, 1998) it is reasonable to align these cycles so that they have a phase difference of one sprint. This way the period for ready-for-delivery releases stays the same but it is one sprint behind the marketing releases. The time of the phase difference is used for demonstrating and selling, providing selected customers with early access to the new product, as well as for stabilization of the new product version.

The only thing that figure 4.2 does not visualize is alignment of the whole model to the calendar. Since the strategic release management cycle is about company level issues within long time frame, it needs to be aligned with strategy updates.

If a company develops more than one product or product family, development of these products should be aligned with strategic release management. A release project should concentrate on one product or a product family. If there is a need for releasing more product versions in a release cycle, work should be divided into separate release projects. However, these separate projects should have fairly similar schedules to match the rhythm of strategic release management.

All the main concepts are described in detail in the following sections. The same structure is used for representing the essential points of each cycle. For every cycle goals, the main activities and participants are defined and listed in a similar manner.

4.2 Strategic Release Management

Strategic release management steers the whole product development by setting the direction for it. It is a cycle where business strategy and management are combined with the technology strategy to produce a vision for product development. The main goals, activities and participants of the strategic release management cycle are summarized into table 4.1. The length of each strategic release management cycle is six months. Even if the cyclic nature of the strategic release management is not as evident as it is in release project cycle, it has clear points of redirecting the development. Redirecting is caused by changes in company business and technology strategy, which are the main inputs for the strategic release management.

A six-month period reflects biannual strategy updates and it also ensures fast feedback between business management and product development. The roadmap is updated twice in the cycle to reflect the results from release project cycle. One person, who is usually the head of product development, is responsible of managing this cycle.

Table 4.1: Strategic Release Management

Goal	• Schedule product releases concerning the business and technology strategy • Define the main issues for product releases • Resource development projects
Activities	• Collecting feature ideas and feedback • Updating product roadmap • Resource allocation for the release projects
Participants	• Head of product development • Management team • Product managers • Customers and partners

The goals related to the release schedule and content of the releases can best be explained by using the concept of product roadmap. A product roadmap is actually simply a high-level release project plan for all the products of the company. This implies that the goals of strategic release management can be divided into two areas: roadmapping and resource allocation.

4.2.1 Roadmapping

Roadmapping is an interactive process for producing a roadmap. It provides a way to evaluate different strategic alternatives to produce the best possible plan to reach the business objectives of the company. In short, the roadmap answers the questions when and what, while roadmapping tells how to find those things out.

The question of when to make a product release is always challenging. Good timing might have a great business effect and, respectively, bad timing might make new product release even worthless. The requirement of a *time oriented* process asks for fixed release deadlines, fast release cycle and the right rhythm. Since the products of the case organization are not mass-market products where promoting them at a certain exhibition is vital, regular and fast paced release schedule plays a more important role. Salespeople can be more confident when promising a customer a new release of a product when the release cycle is predictable. When existing customers know that new versions are released periodically they have a better chance to plan their activities, like promoting a new service including the product. Customers and partners also have a chance to affect product features regularly and when the release periods are short, they get to evaluate the potential of the new features fast. This is especially relevant in markets that will not stand still. Releases of different products have to be scheduled so that the development team has a chance to concentrate on one specific problem field at a time. If there is more than one development team, multiple release projects can be done in parallel.

The new model introduces a continuous three-month release cycle. This gives predictability, rhythm and possibility to react fast to customer requests. Only one release project should be scheduled for the same time window since there is only one development team in the case organization. An example of a compact product roadmap is presented in table 4.2 (the product vision is not completely shown).

Table 4.2: A simple roadmap example

Product	**Q1**	**Q2**	**Q3**	**Q4**
Mobile Product		2.0 enables voice over GSM		2.1 wireless short messages
Network Product	0.1 is prototype for network monitoring		1.0 network monitoring agent	
.NET research				New possibilities and threats

The question 'what' includes two major things: long-term strategic decisions and defining mid-term product release contents. Long-term decision about adapting (or not adapting) new technology into the products needs to be included into the roadmap. This gives a tool to fulfil the requirement of *research for the future* if these items in the roadmap include needed research and competitor analysis. Long-term decisions also include decisions whether the development of a specific product should be ended due to the market or to strategic reasons. Mid-term decisions about the content of product releases require feedback and information from customers and various other sources. This makes the planning process more complicated.

Each product release that is included into the roadmap needs to have a scope for the content. For the new model, the concept of product vision is applied from synch-and-stabilize and RUP combined with Scrum's concept of product backlog. The purpose of the product vision is to crystallize the main things about the release and initially select the features from product backlog to release backlog. The feature ideas must primarily come from customers as the requirement of *customer oriented* presumes, but opinions of other stakeholders are considered as well. The primary inputs for product features and requirements are:

Customers and partners are heard by asking for feedback from possible previous products and by holding a special meeting for brainstorming new feature and product ideas.

End-users are considered by arranging customer (end user) satisfaction questionnaires with the actual customer.

Results from previous projects are taken into account especially when something important has been found out, for example in a product benchmark.

Employees being professionals in the field are asked for ideas and feedback and they are given a chance to add ideas straight to the product backlog.

Reports including articles in relevant magazines are studied for ideas.

The challenging task of prioritizing the product backlog that has all the collected feature ideas is done by the head of product development. He is not to do it by himself but in cooperation with selected customers and partners and product managers. After grouping the prioritized features into preliminary release backlogs they are ready to be scheduled into the product roadmap.

Managing requirements and feature ideas from various sources needs to be effective in order to really help roadmapping. Supporting this action with a proper tool can remarkably improve the complicated process, as is pointed out in RUP. This is one of the reasons why requirement management is part of the supporting actions of the new process model. This was also a requirement for the new process model.

4.2.2 Resource allocation

Even if resource allocation in a small company is sometimes done even on a daily basis, the fundamental decisions about allocating resources need to be done in the strategic release management cycle. The human resources of the company are divided between different projects such as development, delivery, product customization and support projects. To be able to plan product releases beforehand by roadmapping, development projects need to have proper resources. The amount of resources available naturally affects the amount of new features possible to develop per release project. At this point, a decision needs to be made about how the resources of the company are to be divided and whether new employees are to be recruited. The proportions may differ from time to time but it is important for a development project that the guidelines have been decided. To achieve more flexibility, outsourcing is an option, especially in customer projects.

The new model suggests allocating a certain percentage of the available effort to development projects. This effort would preferably be from persons that have no or little tasks and responsibilities in other projects. This approach is adopted from Scrum and makes managing release projects easier and the team more efficient.

4.3 Release Project Cycle

The main purpose of the release project cycle is to make sure that release projects are done as they are defined in the product roadmap. The main goals, activities and participants of the release project cycle are summarized into table 4.3. The length of release project cycle is three months to align efficiently with strategic release management cycle. The Product manager in cooperation with the head of product development is responsible for managing this cycle.

Table 4.3: Release Project Cycle

Goals	• To schedule sprints • To update the product vision and refine the release backlog • To produce a fully-functional and tested product or a product version
Activities	• Scheduling sprints inside the release project • Reprioritizing and selecting features for the release • Technical planning session for feature estimation and architecture planning • Creating testing strategy for the release project
Participants	• Head of product development • Customers and partners • Product managers • Development team

The goals and activities of the release project cycle can be divided into two parts: scheduling the sprints and refining the release content. The goal of producing a fully-functional and tested product or product version is taken care of in the sprints. If properly planned, the last sprint produces a ready and tested product.

Selected customers are given a possibility for early access to the new version. This might include trialing, joint testing and early customization of the product. In synch-and-stabilize, this phase of external testing is considered an important source of feedback about the product. Feedback is also to be collected from the customers using the early access possibility. It is used mainly for improving the next product versions but some minor things can be included into the current version as well.

The release project cycle gives feedback to strategic release management by refining the release descriptions in the product roadmap and by introducing new product releases.

4.3.1 Scheduling the Sprints

Sprints are scheduled evenly inside the release project. The length of the sprint is advised to be 30 days in Scrum while iteration length is advised to be 1-3 weeks in XP. The new model adapts the approximately one month sprint length to align sprints with the release project and to fit development well to management approach of Scrum. The most important single thing about the sprint schedules is the dates and times for sprint demonstrations. These dates are to be published for the whole company so that people can include them into their personal schedules.

4.3.2 Refining the Release Content

Refining the release content aims to crystallize the product vision and to refine the release backlog that have been initially defined in the strategic release management cycle. The main thing in refining the release backlog is reprioritizing the features. At this point it is also possible to add features that are not included in the initial release backlog.

Scrum does not tell how to prioritize the items in the release backlog. It just tells that only one person should be responsible for doing that. In RUP features are prioritized by risk and with the *"architecture first"* approach. XP suggests prioritizing by the value produced to the customer while synch-and-stabilize prioritizes features by user activities and data. The new model suggests applying a combination of these practices in a following manner:

- The product manager is responsible for feature prioritization but does it with cooperation with experts and customers.
- Collecting usage data of the products from customers.
- Using internal experts to make initial prioritization based on usage data and their experience.
- Holding sessions with customers and partners to prioritize features.
- Holding technical planning session where architectural issues are considered and added to prioritizations.

- Holding session for defining a testing strategy for the release.

Prioritization is done mainly by the customer as it is required from the model (requirement of *customer oriented*). To also fulfil the requirement of *architecture centric* development the architectural issues are also included in the prioritization. The testing strategy for the release covers testing issues that are not taken care of in the daily development and need to be included in the sprint planning. These tests are, for example, performance test at partners' premises, usability tests, testing on different platforms and testing of different localizations. Work needed to implement the testing strategy of the product is included in the release backlog.

The first sprint has an architecture theme. This means that in the first sprint architecture is implemented for the most important features. Usually, the main themes of the sprints are:

1. Architecture and stabilization
2. Feature implementation
3. Finalizing

Stabilization theme in the same sprint as architecture is actually about the previous release. Stabilization period of one sprint takes place parallel to architecture development, since all the development resources are not bound to new product release and stabilization of the product is needed for more efficient deliveries and less support. The feature implementation theme is the main sprint for new feature development even if some features might be developed in other sprints as well. The sprint with finalizing theme concentrates on integration testing and packaging. Released product increments from sprints with different themes are called *prototype*- (architecture), *alpha*- (feature implementation), *beta*- (finalizing) and *release*-versions (stabilization).

In the finalization sprint product features are frozen and documented. A data sheet describing the features of the product and differences to the previous version is written together with the people responsible for marketing them. Value added and the benefits provided by the new version are described in this document as well.

Items in the release and product backlogs consist of all work that can be foreseen for the product or a release. As it is defined in Scrum, backlog consists of *"product features, functionality, infrastructure, architecture and technology"*. Items are described with short textual expressions, as done in synch-and-stabilize product vision document.

4.4 Sprints

The main purpose of sprints is to produce stable increments to the product. The concept of sprint is adapted from Scrum because it was found to support well requirements related to short-term management of product development (requirements of *time oriented* and *visible development* and *fast reaction to changes*). The main goals, activities and participants of the sprints are summarized in table 4.4. The length of a sprint is defined by the release project cycle and it is approximately one month. The scrum master, a concept adopted from Scrum, is responsible of the success of the sprint with the whole development team. The role of scrum master is close to the traditional role of project manager.

Table 4.4: Sprints

Goals	• To define sprint goal and features that are to be implemented in the sprint • To produce a stable increment for the product as defined in the sprint goal • To demonstrate the product with the new features • To gather feedback from demonstration and sprint project
Activities	• Sprint planning session where sprint goal is defined and sprint backlog is created • Sprint review session where new features are demonstrated
Participants	• Product managers • Scrum master • Development team

Goals and activities of the sprints can be divided into planning, management and reviewing. The goal of producing a stable product increment is partly considered here but its creation is described in detail by the daily rhythm in section 4.5.

Sprints provide release project cycle feedback in the sprint review sessions where new the functionality is presented and possible problems are discussed.

During the sprints, the development team has great responsibility over its doings. When the sprint goal has been set, the team is free to organize itself in the best possible way to reach that goal. The scrum master's main responsibility is to help the team to achieve the goal, not to control the team. Besides helping, scrum master has an important role in communicating and encouraging communication within the team to get people aligned towards the right direction. This low organization and divided responsibility empowers the team to challenge the old results and habits.

4.4.1 Sprint Planning

Planning a sprint takes place in a sprint planning session. The participants of the session are the product manager and the development team including the scrum master. Everyone from the development team needs to be present because decisions affecting the whole team are done and because everyone's expertise is needed. The sprint planning session has two goals: defining the sprint goal and creating a sprint backlog.

The product manager presents the release backlog and other relevant background information, such as upcoming customer cases that include some new features, to the development team. Working together, top priority items, including architectural and testing issues, are identified from the release backlog. Having selected the subset of release backlog items, the sprint goal is crafted based on them. The sprint goal is an objective that will be met if the selected subset of release backlog items are implemented. The sprint goal is like a small product vision that guides the development of the sprint. Sprint goal is a destination for the development team but the path to get there is not defined. That gives the team room for innovative ideas.

The second part of the sprint planning session concentrates on defining a sprint backlog that is a list of tasks to be completed in order to achieve the sprint goal. At this point, the development team takes responsibility of the planning. If the product manager participates in this part of the meeting, he is to take a very passive role. The only time he should be involved is when the sprint goal is considered. Tasks in the sprint backlog are derived from the subset of release backlog items that are to be implemented in the sprint. The amount of work needed to accomplish a task is estimated by the team and recorded to the sprint backlog. The size of a task should be approximately from eight to twenty hours. After all tasks are initially estimated, the total amount of available working hours per day is estimated for the whole development team. The first estimation of working days needed to accomplish the sprint goal is calculated by dividing the sum of the task estimations by available working hours per day. This estimation is then mapped to the calendar to get the first estimation for delivery date. If this date is too far from the planned review date of the sprint, the content of the sprint has to be adjusted. Adjusting is done by modifying the sprint goal so that tasks can be dropped out or added. Sprint goal and tasks are adjusted until the estimated delivery date is close enough to the planned review date.

An important thing in creating the sprint goal and the sprint backlog is that the development team creates them, so that they are committed to reach the goal.

Items from the release backlog that are to be implemented during the sprint are defined in more detail. Functionality that has been described in a few words needs to written out so that it is understood in the same way by the developers and by the customer (or the product manager). RUP suggests using use cases for defining functionality in detail. This approach is also suggested by the new model to take advantage of the end-user perspective of the use cases.

4.4.2 Managing the Sprint

The sprint is managed using *project burn down graph* and *daily scrums* as tools. The project burn down graph is a function of task estimations over calendar time plus a linear slope from the current date to estimated finish date (see an example graph in the figure 4.3). The daily scrum is discussed in section 4.5

Work estimations for tasks are updated constantly during the sprint by the developers. The estimation is actually the amount of work left for a specific task. In an ideal case this estimate would go down linearly as developers work on tasks. However that is not always the case. Estimation may go up when something unpredictable happens. If additional work is discovered during the sprint it is added to the sprint backlog. An example of a project burndown graph is presented in figure 4.3. In this example, the

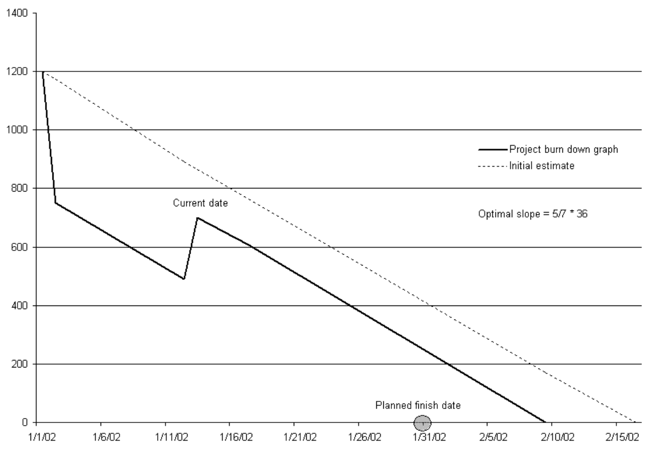


Figure 4.3: An example of a project burn down graph

initial amount of tasks was too large so that the sprint goal needed to be adjusted before starting. Then work proceeded as planned until something unpredictable came up and the graph went up. After that work continued as planned but from it was evident that the planned finish date would not be reached. At this point, the product manager and development team hold a meeting where decisions about actions are made. When the sprint review date and amount of available working hours per day (the size of the development team) are fixed, the only variable is the sum of task estimations. Tasks need to dropped out by adjusting the sprint goal. The effect of adjustment and optimal situation can described by a mathematical equation

$$\left| \sum_{i=1}^{n} x_i - \overline{w}\frac{5}{7}(t_{finish} - t_{current}) \right| < \Delta, \tag{4.1}$$

where x_i is an estimate for a specific task, $\overline{w}$ is average amount of working hours available per day, n is number of tasks and Δ is an acceptable deviation from the planned finish date. Current date and planned finish date of the sprint are represented by $t_{current}$ and t_{finish} and the actual working days are taken into account by multiplying the amount of working hours available by $\frac{5}{7}$ to exclude weekends.

Project burn down graph provides a real-time picture of the project's state if task estimations are updated frequently. This is not a sufficient method for managing the sprint but combined with the activities of daily rhythm (see section 4.5), it is a powerful tool for controlling the project.

4.4.3 Sprint Review

Reviewing the sprint takes place in a sprint review session. Participants in the session are the development team, the product owner, the management of the company, sales and project people and possibly selected customers. The scrum master is responsible for coordinating the sprint review session including sending out the invitations with agenda and agreeing with the team who is going to present and what. The sprint review session is an informal meeting where the most important goal is to share information. The main steps of the sprint review meeting are:

Sprint overview describes the highlights of the sprint. The sprint goal and release backlog are compared to the actual results and reasons for differences are discussed. The actual amount of effort done in the sprint considering working hours is compared to the estimated amount of working hours and differences are discussed. The scrum master presents the overview.

Architectural overview is given by a team member to describe the main technical points and their relations to functionality.

Demonstrating the product increments is done feature by feature by team members. Demonstrations are done in a 'real' production environment not on developers' own computers. Participants should understand as many dimensions of the product increment as possible, its strengths and weaknesses.

Feedback questions, observations, discussion and suggestions are allowed and encouraged.

The results from the sprint review session, the product increment and the feedback, are used in the planning session of the next sprint.

4.5 Daily Rhythm

Daily rhythm describes day-to-day activities in product development that drive the team towards the sprint goal. The main goals, activities and participants of the daily rhythm are summarized into table 4.5. The length of activities in the daily rhythm varies from a few hours to about a week.

Table 4.5: Daily rhythm

Goals	• To steer the development towards the sprint goal • To produce quality software • To ease communication inside the team • To synchronize work of team members
Activities	• Designing • Coding • Testing • Synchronizing the work and the project • Demonstrating working software • Adjusting the direction of the project
Participants	• Scrum master • Development team

Daily rhythm is a combination of practices from XP, synch-and-stabilize and Scrum. The engineering practices are mainly adopted from XP while management practices are adopted from Scrum and synch-and-stabilize. Daily rhythm concentrates on fulfiling the requirements *visible development*, *extensive testing*, *learning organization* and *end-user oriented* development.

Daily rhythms can be seen as a circle where the main activities (see table 4.5) are located sequentially. To clarify the real essence of the main activities, more concrete actions need to be related to them. Relations of the activities and the daily rhythm cycle are visualized in figure 4.4.

The main activities are discussed next with their relations to the concrete actions and practices.

4.5.1 Design

Design is done with the two most important things in mind: keep it simple and keep the end-user in mind. The principle of keeping design simple is

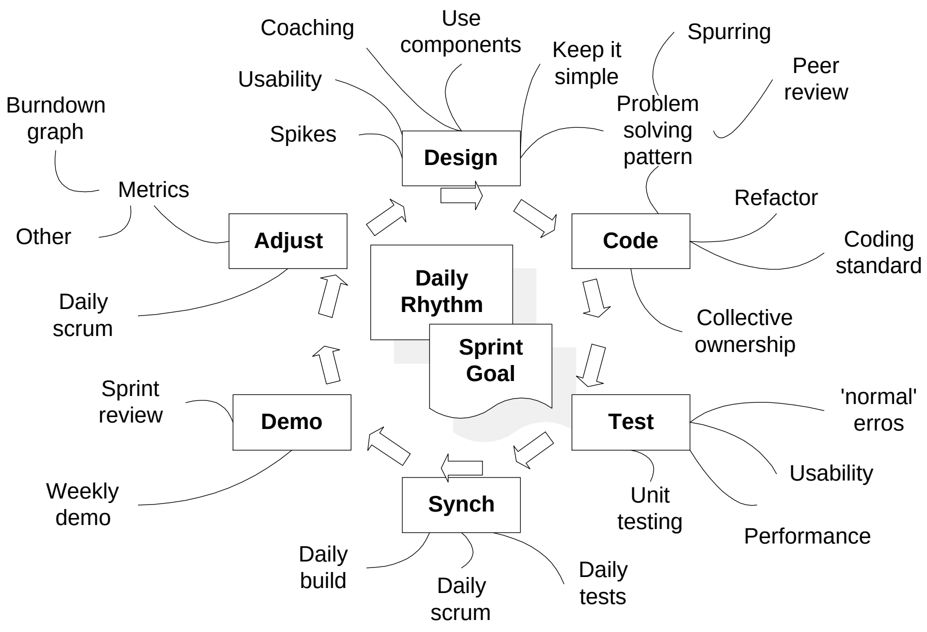


Figure 4.4: Daily rhythm and activities

adapted from XP but other models stress that as well. The end-user viewpoint is adapted from synch-and-stabilize that uses great effort on designing and testing usability.

The new model adapts a role of a coach from XP. Coaching is primarily concerned with technical questions. The coach needs to be fluent in the technology and the product, so for example *chief architect* would be a good choice for the job. His duties are to be available for developers in difficult technical tasks, to encourage and plan refactoring and to help developers with technical tasks like testing and design. To help the coach and to make coaching visible for every developer a problem-solving pattern is introduced. The pattern has five steps (the actor(s) are in parenthesis):

1. Create an initial idea for solving the problem (developer)
2. Spurring and brainstorming the idea (coach and developer)
3. Implementing and testing the idea (developer)
4. Reviewing the implementation (coach and developer)
5. Finalizing the implementation (developer)

This pattern encourages learning by working together, gives the developer confidence about the solution and strives for better quality using peer reviews. Developers may try different approaches to the problem by implementing simple programs that evaluate the approach before presenting the idea to the coach. These small programs are called *spikes* like in XP.

Especially in the sprint that has an architecture theme, development is concentrated on building a stable and well performing architecture. A component based architecture which reflects the product structure and where standards and 3rd party components are used extensively is favored. This approach is adapted from synch-and-stabilize model. A well-thought-out architecture ensures the possibility to form a standard way of integration to existing systems. The coach steers development towards good architecture by spurring and reviewing developers' ideas and implementations.

Design of architecture and components can be done using modeling techniques and workflows presented in RUP. The level of details in the design should reflect the criticality of the component. The new model does not require more than high-level design documents that can be presented in the sprint review session. The development team is free to use more detailed modeling if they see it helpful to reach the sprint goal.

The approach of synch-and-stabilize, where usability is tested throughout the development is adapted to the new model. Usability issues are considered in the design by using heuristic evaluation of the user interface prototype. Heuristic evaluation is a cost efficient way to find usability problems from products. The new model instructs to use approach developed by Jacob Nielsen (Nielsen, 1992), which introduces very concrete principles for heuristic evaluation.

Documentation of the technical solution, design and implementation, is written into source code files. Since the case organization is using Java for all development, it is natural to use Javadoc for generating documentation from the source code. Documentation in the source files needs to be uniform to really be informative. The new model suggests using widely accepted Javadoc writing convention[4] provided by the Sun Microsystems Ltd. complemented with company-specific guidelines in the similar manner as a coding standard.

4.5.2 Code

In the new model coding is really close to design, actually they overlap at some points. This approach is adopted with some changes from XP where the design comes through the code. The problem-solving pattern for design is a link between the design and coding. Besides guiding design in the right direction, it guides the actual implementation using peer reviews.

Refactoring code, a practice adopted from XP, is an essential element of coding in the new model. It helps in keeping the product solid and not

[4]See the convention at http://java.sun.com/j2se/javadoc/writingdoccomments.

to fragment while new increments are added to it. Refactoring guidelines and experience reports have recently been published (for example chapter *Refactoring and Re-Reasoning* in the book *Extreme Programming Examined* (Succi and Marchesi, 2001)) which gives more confidence for using it.

The practice of collective code ownership is adapted from XP as well. Any developer has the right and the responsibility to correct defects in any part of the code. This naturally implies that the one who makes the change is also responsible for testing that the change made works fine and does not break anything else. Using a common coding standard is essential for effective use of collective code ownership. The new model suggests usage of Java coding standard[5] complemented with company specific guidelines that are described in a separate document. Collective code ownership empowers learning from the code and makes the development team more flexible.

One of the main coding practices of XP is dropped from the new model. Pair programming is not used extensively; only when developers see that it is useful for solving a problem or sharing information.

4.5.3 Test

Extensive testing is one of the requirements for the new model. Testing strategy is mainly adopted from XP with some practices from RUP and synch-and-stabilize. The main point about testing is that it continues throughout the whole development process from the beginning to the end. It is not a single activity at the end of the project.

XP argues that every code that can possibly break should be tested. This is taken as a motto for testing in the new model as well. Unit testing is done parallel with coding using independent and automatable test programs. Having testers for each component encourages developers to refactor their code and correct defects made by other developers. If refactoring or fixing results in an error, it is easily found by running the testers. This gives the developer confidence to make changes that make the code and design better. Peter Grassman lists benefits of writing unit tests in the chapter 15 of the book *Extreme Programming Examined* (Succi and Marchesi, 2001):

- Short feedback loop
- Improved changeability of the system
- Fewer errors
- Micro design improvements
- Regression testing
- Refactoring support

[5]See the coding convention at http://java.sun.com/docs/codeconv.

- Communicating design and documenting code
- Teaching junior programmers
- Installing a developer machine
- Boosting morale

and also some pitfalls and problems:

- Short feedback loop
- Testing cache
- Out-of-sync tests
- Accessing database
- Complicated test code
- Meaningless, easy tests
- Problems with people

Comparing these pros and cons, it is reasonable to take advantage of unit testing in the new model.

A framework for making and running unit tests has been created by the XP community. The JUnit framework[6] provides means for fast implementation of the testers and feasible way for running them. Some guidelines and experiences have been published about using the JUnit framework for unit testing (Succi and Marchesi, 2001). Even more advises and experiences are available on the WWW produced by the XP community. The new model suggests using JUnit framework for testing Java software at the unit level. The coach of the development team can help team towards a common way of writing effective unit testers.

The new model requires writing and running performance tests on critical components. These tests can be done using extensions of JUnit, writing special testers or by using ready commercial or open source products.

Checklist for so called 'usual' errors is used for preventing them from appearing time after time in the product. The list of usual errors is selected from a comprehensive list provided in the appendix of the book *"Testing Computer Software"* (Kaner et al., 1999) and it is updated constantly with errors specific to the product and organization. Usual errors are discussed in the review with coach and developer.

Usability testing is done using heuristic evaluation as described in section 4.5.1. If needed, the development team can use usability specialist for spotting the biggest usability problems in the product.

[6]See http://www.junit.org for more information.

4.5.4 Synchronize

Synchronizing is about synchronizing the development work of team members and the understanding of the project status. It builds on daily activities the main purpose of which is to share information and encourage communication inside the development team.

Daily Scrum

The concept of daily scrum is adopted from Scrum. It is a meeting which the development team attends to communicate. It takes place on every working day at the same time and at the same place and lasts a maximum of fifteen minutes. The purpose of this status meeting is to get first-hand information about the project, identifying impediments and making decisions. It is required that every team member participates in the meeting physically or via phone so that no one misses important pieces of information. In the daily scrum every team member is asked three questions:

- What have you done since the last scrum?
- What will you do between now and the next scrum?
- What got in your way of doing work?

With these questions, the development team and the scrum master should have a pretty clear picture about what the status of the project is. The scrum master is responsible of removing the obstacles that come up in the meeting. The whole team is to make decisions based on the information, if needed. If the team does not feel that they are able to make a decision scrum master makes the final call. The team should every once in a while take a look at the project burndown graph to get more information about the progress of the sprint.

The scrum master is responsible for successfully conducting the daily scrum. His job is to keep the meeting short and concentrated on the correct issues. If the meeting is about to turn into specification session, the scrum master is responsible for interrupting it and advising to continue after the daily scrum with relevant team members.

Daily Build

The daily build concept is used in synch-and-stabilize and XP. The new model adapts this concept to help and force the team to synchronize their work. A daily build is done automatically every night. The build makes a 'nightly build' of the product. It takes the latest code versions from the version control system, compiles and links the whole product, including old and new components, and makes an installation package out of it. The result can be used on the next day for testing.

The Scrum master is responsible for checking that build has been successful every night. If the build fails, the one responsible for breaking the build has to fix the defect that caused the problem as his first thing in the morning. In synch-and-stabilize the 'build ruiner' is given a penalty such as task for taking care of the build. The new model also suggests penalty for ruining the build. The penalty is up to the development team to decide. A successful build is important because non-compiling code can prevent the whole team working efficiently next morning. Another point is how the team can say anything about the status of the software if it does not even compile. Other motivations for daily build are described in the book *Microsoft Secrets* (Cusumano and Selby, 1998).

Daily Tests

The concept of daily testing takes one step forward from the daily build. After making the daily build, all unit testers are run against the new version. This practice expects that unit testers have been written as described in the previous section and that the daily build is successful. The new model suggests that the daily test is run nightly just after the nightly build. The result from the test run is available in the morning. If there are failed test cases, the defects should be corrected immediately. If daily build gives a little confidence to the development team about the state of the product, daily tests give much more of it if the tests are written properly.

The daily scrum, build and testing are practices that make development visible as it was required from the new model.

4.5.5 Demonstrate

Demonstrating a working product increment is done in the sprint review session as described in section 4.4.3. The main purpose of that demonstration is to share information about the new product increment in functional and technical sense. Great value of the review comes from seeing the running software and a possibility to comment it. Nothing visualizes the state of the software better than seeing it in action. The new model takes demonstrating the new features and functionality into daily rhythm of the project.

Even if the state of tasks and the project is reviewed daily in the daily scrum meeting, it is not a visual way of seeing the state of the software. A weekly demo is a chance for every developer to show what they have accomplished in terms of running software. Not everyone has to demonstrate something every week but at state when a new component is running. As a principle, the tasks in the sprint backlog that produce software (not for example configuring environment) are demonstrated. If the new component does not have a user interface, it can be demonstrated by the unit or performance testers.

Demonstrating pieces of running software weekly has the same kind of benefits as writing unit tests and holding the sprint review:

Getting a visual image about the state of the software compared to the sprint goal.

Sharing information about the software, how it looks and what kind of solutions have been used.

Confidence of the team in the software rises when they see it running more and more every week.

People that are not part of the project are free to participate in the weekly demonstrations if they are interested. The agenda of the demo is not fixed beforehand but the demo is always at the same time and at the same place.

4.5.6 Adjust

Adjusting the direction of the project is done frequently using first-hand observations and metrics. The scrum master is responsible for initiating adjusting but the whole development team takes part in correcting the direction. If adjusting the project includes changing the sprint goal (which is not advisable if the situation does not really require it), the product manager has to participate in adjusting and setting a new direction to the project.

First-hand observations come mainly from the daily scrum meeting. The status and progress of the individual team members gives required information for evaluating if adjusting of the project is needed.

Metrics are used for adjusting in a limited way. Scrum advises to use the project burndown graph as the only metric. This is adopted by the new model to give quantitative information about the state of the project. Other metrics can vary so that a new metric is introduced when some particular thing needs attention as described in XP. For example if it seems that there are not enough testers written, a metric indicating of the amount of testers could be introduced. XP has an adaptable idea for amount and lifetime of metrics *"Don't have too many metrics, and be prepared to retire metrics that have served their purpose. Three or four measures are typically all a team can stand at one time"*.

Adjusting the project can include minor things like giving a task from one person to an other or changing the implementation order of few things because additional dependencies were found between them. On the other hand, adjusting can include major things like dropping features or even changing the sprint goal.

4.6 Supporting Actions

Supporting actions are tools and methods for helping product development achieve its goals. The main goals, activities and participants of supporting actions are summarized into table 4.5. Supporting actions are not cyclic like the other main concepts of the new model. Supporting actions are continuous things that keep the development rolling from one phase to another. Their importance is not often seen until they fail and stop the wheel of development from rolling. Supporting actions are not a responsibility of one person but responsibility is divided by the activity.

Table 4.6: Supporting Actions

Goals	• To help the development team to concentrate on achieving the sprint goal • To form a common basis for working
Activities	• Configuration management • Requirement management • Defect tracking • Daily builds • Automated testing • Maintaining test and development environments
Participants	• System administrator • Product manager • Scrum master • Development team

Supporting actions are mainly adopted from RUP's core supporting workflows. In addition, some other actions are introduced to support the model. RUP actually combines configuration and requirement management and defect tracking into one supporting workflow called *Configuration and change management workflow* (CCM). It is described to form a CCM cube that covers three interdependent functions: configuration management, change request management and status and measurement. Change management includes defect management as part of it and status and measurement gives time dimension to the other two functions. Here, these actions are described separately to keep the structure clear, even if all the supporting actions are related to each other.

4.6.1 Configuration Management

The main purpose of configuration management is to provide means for storing and relating different versions of different work products such as source code, documents and testers.

Storing and managing different versions of different files is commonly referred to as version controlling. There are different tools for version control that can be used. An easy and light-weight tool that fits well to the approach of the new model is CVS[7]. Naturally, a tool is not adequate for version control but a common way of using it and clearly defined directory structure is also needed. The data repository of the version control tool needs to be backed up regularly so that developers do not have to worry about breaking hard drives if they have put their work into the version control system. The administration of the company is responsible of maintaining the version control tool, server and backups while the development team agrees on the structure and way to use version control system.

Configuration management can be understood as management of different sets of artifacts with different versions. Configurations are a combination of different elements which all have their own version. For example, a software binary is built from combination of source code files that have different versions. In practice, the most important configurations to manage are

- Binaries and related source files.
- Release packages and all binaries and documents that are related to them.
- Customer specific delivery packages and release plus customization packages that are related to it.
- Product fix packages that are for a specific release package.

Most version control systems like CVS support combining various artifacts into one configuration. Combined with a build tool that enables selecting the correct artifacts, configuration can be handled efficiently. A good choice for a build tool for development teams using Java would be Ant[8]. Using a tool here does not guarantee good configuration but a common and semi-automated way of using it makes chances of success a lot better. The product manager is responsible for defining how to use the tools when it comes to product release packages and the development team, when it comes to actual usage of tools.

4.6.2 Requirements Management

Requirements management is done using product and release backlogs. Items should be easy to add to product backlog so that all stakeholders can do it without consultancy from product manager. Also, prioritization of the requirements should be easy. In addition, there should be some basic information about the requirements that is common for all the items:

[7]CVS stands for Concurrent Versions System. See more at http://www.cvshome.org.
[8]See http://jakarta.apache.org/ant for more information.

- Subject of the requirement.
- Priority of the requirement.
- Where the requirement was originated (a person or an organization).
- Name and contact information of who entered the requirement.
- Comprehensive description about the requirement.
- Type of the requirement (functional, usability, performance, etc.).
- The value generated by the feature.
- Decisions made about the requirement (for example: to be implemented in version 1.4).

Based on this information it is easier to prioritize requirements and to make decisions about the content of the release backlog.

The usage of Excel spreadsheets is adequate if they are used in a centralized manner but a simple application using a database with WWW user interface would provide easier access, for example, to add feature ideas to the product backlog.

Change requests are straight forward to handle in similar manner as the requirements. Change requests are added to the product backlog and from there they are taken to release backlog when it is their time.

4.6.3 Defect Tracking

Defect tracking is managing the defects of a product. It needs to provide means for reporting defects, following their status and organizing the fixing. Defect tracking is an activity that starts from the beta-version of the product and goes on until the product is not supported anymore.

For efficient defect tracking, a tool is needed. There are plenty to choose from in the markets but, as an example, an open-source product Bugzilla[9] is a light-weight and widely used tool. The tool itself needs to have some basic features:

- Reporting the defect with proper information (subject, description, severity, product, environment, etc.).
- Grouping defects by products or components.
- Prioritizing the defects.
- Assigning defect to a person.

[9]See http://bugzilla.mozilla.org for more information.

- Changing the state of the defect (for example: open, working, testing, closed).
- Informing concerned people about the changes in a defect report by email.

The defect tracking tool is also to be used in maintenance projects as a task list and organizer of work.

4.6.4 Daily Builds

Daily builds need to be automated in order to be effective and useful in development. Actually a daily build is just a scheduled event that uses a version control system and a building tool. In short, steps of the daily build (adapted from synch-and-stabilize) are:

1. Checkout source code from the version control system.
2. Compile all components.
3. Package the binaries and other files into a build archive.
4. Copy result to web site where the build is available for using the next day .
5. Report the success of the build to the development team.

The daily build needs very little extra to normal product build scripts. The main additional thing is the script that runs the build and analyses the results and reports them to the development team. The scrum master is responsible for the daily build structure.

4.6.5 Automated Testing

Automated testing is needed for daily tests. Automated testing uses the same framework for testing as unit testing in general. It is actually a wrapper for testers that is run automatically. Tests are run against the new binary version of the product after the daily build has been done. The steps of automated testing are:

1. Compile all testers.
2. Run pretest scripts to clean up the environment if needed.
3. Run testers one by one.
4. Report test results to the development team.

If testers are written like they should be, independent of the other tests, running tests centralized is not a big issue. In addition, proper database connections need to be available and mechanism for reporting the test results need to be implemented. The scrum master is responsible for keeping the automated tests running.

4.6.6 Maintaining Environments

Developing software is very dependent on environments: if a developer's computer does not work, no code will be created. Properly working information technology infrastructure is vital for development projects. The environment can be divided to three different areas: common, development and testing environment.

The *common environment* includes a personal computer with proper office applications, email and calendar applications, internet access and backing up of all needed information. Without these basic things, development will freeze. The system administrator is responsible for the basic environment.

The *development environment* includes compilers, IDEs[10], testing frameworks, emulators, database, version control system, defect-tracking system and possibly product-specific software. The responsibility for maintaining these elements is divided between the system administrator and the development team. As a rule of thumb, the system administrator is responsible for applications that have more than one user, such as databases. To ease the maintenance of the environments, developers need to have standard working directory structure and standard tools for development, not including personal IDEs.

The testing environment includes different test environments, different test devices, databases, testing software and product specific applications. The responsibility for the testing environment is divided between the system administrator and the development team on a project-by-project basis.

4.7 Summary of the New Model

In this section, the new software process model is summarized and compared to its requirements. In tables 4.7 and 4.8 the concepts of the new model are mapped against the requirements. For each requirement the main points of the new model are presented.

The new model consists of six separate concepts of which five represent the phases of the model. Presenting the requirements against the phases of the new model gives a clear picture of when each requirement is considered.

[10]IDE stands for Integrated Development Environment.

It is evident from the table that some of the requirements are considered throughout the whole process whereas some are taken care of in just one phase. This is reasonable since some of the requirements deal mainly with long-term issues and are not worth considering on a daily basis.

The new model can be consider to support the requirements well since every requirement is considered at some point of the new process. Naturally, every relevant aspect cannot be included into the summary table but a high-level view on the new model can be achieved.

Table 4.7: Requirements And the New Model

	Periodical approach	Strategic release management	Release project cycle	Sprints	Daiily rhythm	Supporting actions
Architecture centric			Prioritizing architecture		Designing architecture first	
Time oriented	Periodic development cycles	Roadmapping and constant three month release cycle	Fixed deadlines in sprints	Task estimation and active project control	Daily build, tests and scrum meeting	Automated daily build and tests
Fast reaction to change	Continuous feedback loop	Querterly roadmap update	Short release cycle	Features can change for every sprint	Steering project on daily basis	Support for requirements management
Customer oriented		Collecting customer feedback	Feature prioritization with customers			Support for defect tracking
Managed requirements		Product visions in the roadmap	Using release backlog	Using sprint backlog		Support for requirements management

Table 4.8: Requirements And the New Model

	Periodic approach	Strategic release management	Release project cycle	Sprints	Daiily rhythm	Supporting actions
Visible development		Short release cycle	Sprint reviews	Project burndown graph	Weekly demos and daily synchronization	Automated daily builds
End-user oriented		Input for roadmap from users		Usage data for sprint planning	Design with usability in mind	
Focus on core product		Long-term strategic decisions	Prioritizating features			
Extensive testing			Testing throughout the project		Continuous unit testing	Supporting testing and defect tracking
Learning organization	Continuous feedback loop		Feedback from customers	Collective sprint planning and review	Problem solving pattern	
Research for the future		Roadmapping research projects				

Chapter 5

Steps for Improvement

In this chapter the reality of the software development in the case organization is described. Current practices are compared to the new model to find out what needs to be done in order to improve the software process of the case organization. A suggestion for improvement steps is presented based on the comparison.

5.1 Current Situation and the New Model

How the current model is actually working in practice is discussed in this section. Practices are also briefly compared to the new model to form a basis for defining the improvement steps.

The information about the current model in practice is summarized from the author's experiences. The reason for this approach is that the author has been participating in product development of the case organization at many levels: building the roadmap, managing development projects, building and managing testing environments, testing products, coding components for the products and improving the development methods. Information about these different areas could also be gathered from various other people to minimize the subjective point of view. However, it is reasonable to expect that a majority of the information can be derived from the author's experiences (Hafner, 2001). A complete, objective and detailed picture about the current model in practice is not necessarily needed, since the research is about improving the case organization's software process, not to model the current situation in detail.

5.1.1 Current Practices

The structure of the current model is used for describing the practice compared to the current model. Each subsection first compares current practices to current model and secondly current practices to the new model. Problems and deviations from the new model are summarized in table 5.1 after the comparison.

Strategic Release Management

The release schedule quite well follows the planned three-month release cycle in practice. The product roadmap currently extends over a period of six months instead of a year. Features that are not scheduled in the releases for the next six months are left without initial implementation schedules. So, a high-level product vision is defined only for the next two releases at the moment. Requirements are currently collected mainly from prospective customers so that the new version of the product includes features a potential customer requires in order to sign the deal. Feature suggestions are not collected continuously, only before every release project. There is no systematic way of collecting suggestions for features in a continuous manner.

The main difference between current practices and the strategic release management cycle of the new model is that the product roadmap should reflect long-term decisions about the technology and products. This, in practice, means that the roadmap should have research projects scheduled, as well as the product releases. Another considerable difference is including feedback from actual end-users for inputs for the product features. Effective support for collecting feature suggestions is also lacking in the current situation.

Fundamental high-level decisions about resourcing product development projects is not done at the moment. Resourcing is currently done on a day-to-day basis which can make the product development projects unpredictable.

Release Project Management

Feature prioritization is done in practice similarly to how it is described in the current model. Only the feature iteration has been found hard to complete in a week because of the tight schedules of the customers. Scheduling iterations has failed in practice at least to some extent. Especially the release iteration has not produced a release-quality version of the product but more like a beta version. This has been unplanned and it has naturally effected the next release project by consuming resources for finalizing the release. Setting up early access has almost not been functioning at all. Then needed infrastructure is not built for early access and goals for early access have not been defined.

The main difference between the current practices and the release project cycle of the new model in general, is using Scrum terminology and practices and adding the stabilization phase after the third iteration. Also, different themes for iterations are not currently done. Scheduling sprint (or iteration) demonstrations beforehand and informing the larger audience about them is not currently done as suggested in the new model.

Refining the content of a release has few additional points to current practices. Collecting usage data and using it for prioritizing features is not currently done. Architecture and testing issues are not discussed currently as a part of the feature and requirement prioritization.

Early access to new product versions is not currently properly organized and feedback from customers is not collected about the new version, as is required by the new model.

Iteration Management

Current practices differ considerably from the current model in the iteration management phase. Different iterations are fairly similar so the features are developed equally in every iteration and architectural issues are handled only incidentally. Probably the greatest difference between the practice and the current model is in the task effort estimations. Estimations are not currently really done and so the scope of the development project is not bound by the estimates. The project team tries to do everything it can to stay in the schedule but without even trend setting work estimations, it has caused projects to fall behind the schedule or in delivering unfinalized products. Reprioritizing features works in the practice as it is described in the current model.

Current practices in iteration management differ remarkably from the sprints of the new model. First of all, terminology and practices adapted from Scrum are not used. Iteration planning does not address adjusting the scope of the iteration using work estimations for the tasks and the developers available to work for the project. A clear goal for the iteration is not established either. The development team is not currently explicitly involved in defining the scope for the iteration, as required by the new model.

Management of the project during the iteration is not currently supported by any metrics or practices. Management is done using mini-milestones as a tool. In practice, the success of the project has been highly dependent on the leading and management skills of the project manager.

Currently, there is no predefined review session after the iteration. This is a fundamental difference to the new model and its sprint review session, which is one of the main activities in the sprint.

Mini-milestones

Weekly goal setting and status updates work in practice like they are defined by the current model. The deviations from the model are greatest when testing and reviews are considered. Only few components have been unit or load tested simultaneously with implementation. This has caused daily automated testing to be nearly impossible. Code reviews have not taken place regularly even with critical components. Daily builds have worked in practice as defined in the current model.

Current practices lack many of the components included in the daily rhythm of the new model. Since there is currently no sprint goal, work is aligned based on the project manager's understanding of the situation. This is fairly far from the idea of a self-organizing development team introduced by the new model.

Designing does not currently have guidelines concerning the architecture, usability or how new features are designed in general. The current practice depends mainly on every individual developer and his skills. A coding standard and collective code ownership are currently in use, but without proper unit testing and peer reviews their value is not high. Testing practices lack discipline and supporting lists of usual defects.

Development is currently synchronized using daily builds, but daily testing and daily meetings like the daily scrum are missing. Weekly demonstrations of working software are currently given, but at the end of iteration, no special demonstration is held.

Adjusting product development projects is currently done by the project manager without help of any metrics. Success in adjusting the direction of the project depends on the skills of the project manager.

Development Support

Development is currently supported by various systems. Most importantly, configuration management is well supported by using CVS as a version control system and Ant as a build tool for the software. The structure of the file repository is well defined and build scripts for different products exist.

Defect tracking is done using a web-based tool that fairly well fulfils the requirements of the new model. It is also in use after the beta release of the product.

Requirement management is done using various documents and work-group note-boards of the office application. This is quite far from the centralized requirement management suggested by the new model.

Different environment are currently fairly well taken care of, but every once in a while unclear responsibilities result services to be out of use.

5.1.2 Summary of the Comparison

In this section the problems in the current model and the deviations from the new model are summarized into table 5.1. Naturally, only the main points can be fit into the table but they give a clear picture of the improvement areas.

Table 5.1: Comparison of the current practices and the new model.

The phase of the new model	**Problems and deviations**
Strategic release management	• The roadmap does not reflect long-term decisions about technology and products • Research and business intelligence projects are not included into the roadmap • Feature suggestions are not collected systematically.
Release project cycle	• Development projects fall behind the schedule • Release iteration does not produce release-quality software • End-user feedback is not used as an input for product development • There is no stabilization phase after the release • Iterations do not have different themes or stressing • Early access to new product versions is not provided for customers
Sprints	• Iteration planing is done without realistic task estimations • Scrum terminology and practices are not used • Architecture and testing issues are not explicitely discussed • A clear goal for an iteration is not established • Team is not involved in iteration planning • Project management is not supported by any metric or a tool • Iteration review sessions are not hold
Daily rhythm	• No design or problem solving guidelines • Code review are seldomly done • Testing lacks discipling • Daily project meetings are not hold
Supporting actions	• Requirements are not systematically managed • Unclear responsibilities in environment support

5.2 Improvements in the Case Organization

In this section a suggestion for improvement steps for improving the software process in the case organization is presented. Improvement steps describe one possible way to take the new model into use in the case organization. Steps are based on the current situation in the case organization and on the new model.

5.2.1 Improvement Steps

The steps are formed so that the organization can deploy them one by one. The steps have been constructed so that they are as independent of each other as possible to ease the deployment. The steps are also prioritized by the author so that first steps are expected to create the most valuable effect on product development considering the challenges the case organization currently has. The improvement steps are:

1. Promoting the New Model.
2. Tuning the rhythm and alignment.
3. Establishing Scrum management practices.
4. Clarifying feature prioritization and data collection.
5. Tuning engineering practices.
6. Improving roadmapping.
7. Improving daily builds.
8. Enabling early access.

The steps are next described in more detail. They do not quite follow the structure of the main concepts of the new model, instead, they form conceptual sets of actions.

It is important to note that adapting a new process model is not a simple thing to do. Timo Kaltio (Kaltio and Kinnula, 1998) lists three critical elements for successful process deployment:

- Well-organized support infrastructure.
- The right product.
- Effective promotion.

It is evident that in the scope of this research all these elements cannot be discussed in detail. Instead, the basic steps for improvement are described and actual implementation is left for the case organization to do. Deploying the new model, as well as any change activity in an organization, need resources, commitment, time and support. These things are left for the case organization to take care of.

Step 1: Promoting the New Model

Promoting the model includes presenting the model to management and the development team and collecting feedback and improvement ideas. Since the new model has the same structure and principles as the current model, promoting is not probably going to be a huge effort.

First, the new model is presented to the management team of the case organization to get approval and commitment for improving the product development as it is planned here. Comments about the model are thought over and the model is refined if needed.

Secondly, the new model is introduced to the development team. Introduction should take the form of a workshop where current challenges of the development are discussed and the possibilities of the new model are evaluated to solve those problems. If minor changes are needed they are done on the model. The result of the workshop should be an agreement to improve development by adopting the new model step by step. The possibility to affect the content of the model and seeing the problems it is meant solve should give the team confidence towards the new model and commitment to adapt it.

Promoting the new model does not necessarily need process support tools and documentation. A clear summary should, however, be made for both management and development team. A summary for the development team should also act as a reference to the new model.

Naturally, each step described here needs additional promoting. This first step is more promoting the whole new point of view and the improvement plan.

Step 2: Tuning the Rhythm and Alignment

The development rhythm is currently fairly close to the rhythm described in the new model. The main things to work out are adding a stabilization phase, input points for business and technology strategy and clarifying responsibilities.

Responsibilities for different tasks in the new model need to be assined. Especially the responsibility of leading the strategic release management cycle needs to be given to one person. This person and his commitment to improve product development are vital for successful adaptation of the new model.

The sales and customer project organizations need to be convinced of value the stabilization phase after the development sprints. This one-month shift in schedules needs to be sold to the customers and partners as well.

Agreement on the points where the company strategy is refined need to be done with the management and maybe even with the board of directors. This defines the points when the roadmap is updated to reflect changes in the strategy.

The value of tuning the rhythm and alignment comes from minimizing the 'down-time' between the projects and from fast reactions to changes in the strategy.

Step 3: Establishing Scrum Management Practices

The main shift in the development compared to the current practices is adopting leadership and management practices from Scrum. The most visible difference in iteration planning is starting to make estimations for tasks and adjusting the scope of the project by these estimations.

The next release project after promoting the whole new model starts with an extra workshop. There, the development team goes through the main concepts of Scrum and developing software in sprints. The team needs to find its tools and methods for making the task estimations and the project burndown graph. This workshop will most likely result in some development tasks for tools supporting the sprint planning and work estimations. The first sprint review should be held after the first sprint even if it has not gone perfectly. This is done for pointing out the importance of working software and deadlines.

After the development team has agreed on how to carry out Scrum activities in practice, the first sprint can be started. Feedback about Scrum practices is gathered after the sprints to make the practices work smoothly for the development team.

The single most important person in this phase is the scrum master. He needs to be fluent with the Scrum practices. He encourages the team to take initiative on solving its daily problems and steering its work towards the sprint goal. During the first sprints, estimating tasks and taking more responsibility over the success of the project, are really challenging. Schwaber (Schwaber and Beedle, 2002) say that it takes few sprints before a team gets used to the new way of developing software. The scrum master is the one person who is to lead the team through these challenging first sprints.

Management and Sales are given a presentation about the Scrum management practices. Especially points where they are expected to participate, such as sprint reviews, are described. They need to be motivated to put their effort into the development by describing the chances they have to influence the development.

The great value of establishing Scrum management practices comes from empowering the development team and making development more predictable.

Step 4: Clarifying Data Collection and Feature Prioritization

Collecting data for feature prioritization has currently some major flaws compared to the new model: collecting usage data and feedback from the end-users and overall management of the data.

Application logs that include the usage data of the product should be obtained from the customers. Sales and project people need to agree with the customers that relevant log files can be transferred to the case organization. Usage profiles are made by analyzing the log files.

Customers should be provided with a standard end-user satisfaction questionnaire. Sales and project people need to agree with the customers that end-users are asked for feedback and that information is available for the case organization. Usage profiles and user feedback provide valuable information for feature prioritization.

The overall management of the feature ideas, requirements and feedback need to be improved in the case organization. It is evident that a tool is needed for collecting, managing and prioriritizing this information. The Product Management team of the case organization should decide which tool they want to use and provide all stakeholders with information on how to use the tool.

There are two major things that are not currently considered explicitly when prioritizing features for product releases: architectural and testing issues. These issues are best worked out in a workshop in which technology specialists from the development team describe the alternatives and their consequences to the Product Management team. After choosing the proper alternatives for testing and architecture these issues can be included into the feature prioritization.

The value of clarifying data collection and feature prioritization comes from getting the information from all stakeholders for the feature prioritization and hence building the right kind of product.

Step 5: Tuning Engineering Practices

Tuning engineering practices includes taking activities of daily rhythm into use. Tuning especially concentrates on design, coding and testing practices. If establishing Scrum management practices depended on the scrum master, tuning engineering practices highly depends on the coach.

The first step in tuning the engineering practices is to hold a meeting with a few of the best engineers from the development team. The most important engineering practices are selected based on the new model. After that all developers are invited to a workshop the goal of which is to together work out the practices for design, coding and testing. The practices of the new model are used as basis, but improvements are added when needed, so that developers can easily commit to use the resulting practices. Especially the coaching practice, problem solving pattern, peer reviews, design guidelines, unit testers, documentation style and usability testing and design are to be discussed.

If the coaching practice and the problem-solving pattern are successfully adapted, improving of the engineering practices is very likely to succeed.

The value of tuning the engineering practices comes from better quality software and shared knowledge about the technical solutions in the product.

Step 6: Improving Roadmapping

Roadmapping is to be improved in two ways: including research and business intelligence issues into the roadmap and making long-term decisions about technologies and products.

Roadmapping is currently working fairly well. The focus of the improvement should be on including research and intelligence projects into the roadmap and making sure that the roadmap extends over one year. Also the product vision is to be added to the elements in the roadmap so that is easy see what the value added by the new version is.

The roadmap should be visible for the whole organization and it should be presented inside the company always after its revision.

The value of improving roadmapping comes from making research and business intelligence explicitly a part of development.

Step 7: Improving Daily Builds

Improving daily builds is about integrating daily tests into the currently working daily build process and agreeing on the actions to be taken if something does not work in the daily build.

Development team should work out a proper way to run all the tests automatically after the daily build. Running the tests and maintaining the testers should not take much extra time. Unit testers should be made in a way that suits to the test automation from the beginning. The result of the daily build and tests need to be visible for all the developers.

Reaction to defects also need to be agreed on. Everyone in the development team should know what happens if a build fails or testers do not run. Penalty for the 'build ruiner' needs to be agreed on.

The value of improving the daily builds comes from the ability to find defects early and from confidence about knowing the status of the development project.

Step 8: Enabling Early Access

Enabling customers and partners to get early access to new versions and products is not only the job of the development team. Actually, support, sales and project organizations have even more motivation to provide early access.

In short, selected customers should be able get early version for testing and evaluation. As a favor in return customers are to give feedback and report possible defects.

The value of enabling early access comes from customer feedback and new sales potential.

5.2.2 Summary of the Improvement Steps

The improvement steps presented in the previous section describe one possible way to apply the new model. The order of the steps or even the content of the steps is not very important. The single most important thing about improving the software development process is to motivate the employees to seek for better ways to do their work. The new model provides a fairly complete set of practices for easing the improvement efforts.

Since each cycle of the model ends with a feedback session or provides feedback in some other way, also feedback about the process is easy to collect. Just as a development cycle steers the next cycle by providing needed feedback and information, it can steer itself for better performance. The overall responsibility for keeping the continuous improvement cycle running should be given to one person.

Objectives of the product development are fairly stable but means to get there may vary. When the direction is clear the process aligns work towards the vision and motivates people to strive for the goal.

Chapter 6

Conclusions

In this chapter the results of the research are discussed. Their reliability and possibility for generalization are discussed together with suggestions for further research.

6.1 The Research Results

This research aimed to find an answer to the question of how can the software development process be improved in the case organization, a small software product company. This research question was answered by defining the requirements for software development process in the case organization, evaluating four existing software process models, synthesizing these models and experiences from literature and practice into a new process model and finally constructing a suggestion for improvement steps for the case organization based on the new model.

Requirements for the new process model were worked out in a workshop with professionals from the case organization. The result from the workshop was a combination of needs for the process, needs for the product and current challenges in general. Since there were relatively many needs they were grouped by theme to form the actual requirements for the new process model. Most stressed requirements from the workshop were requirements related to time orientation and continuous change in the markets, technology and products. The summary of the requirements was presented in section 2.4.

Existing process models were evaluated against the requirements. First, the models were presented briefly and secondly, all requirements were discussed from the viewpoint of model support. It was found that none of the models is perfect if all the requirements are considered. However, almost every requirement has at least one best support rating by the models. This leads to a conclusion that combining these evaluated models to form a new model could support these requirements fairly well.

The new software development process was introduced. It was based on the evaluated models and experiences from the literature. The new model supports all requirements that were given to the process model by the case organization. The new model is based on interrelated development cycles that follow each other in a continuous feedback loop.

The current situation of the software development process in the case organization was compared to the new model to point out the improvement steps. The improvement steps describe phases of deployment of the new model. It was pointed out that commitment to the new model needs to built from bottom up using workshops where the new model can be communicated to all stakeholders and where the stakeholders have a chance to contribute to the model.

The new software process model and the improvement plan that were developed in this research can be considered to answer to the research question well. The new model provides a framework and a set of practices for improving the software product development in the case organization. The steps introduced for improving the current situation to the direction of the new model form a good basis for improving the software development process in the case organization. The concrete contribution to the case organization is the new model that provides answers to the current challenges and problem areas and is based on the needs of the company. The suggestion for improvement can be used as a basis for project plan for an improvement project. Individual improvement steps can be taken by a smaller group of people, for example to improve engineering practices in product development projects.

Secondary contribution of the research was that people involved in the research gained valuable information about different approaches to product development. Even if the research concentrated on solving the problem in the case organization, some valuable experiences were produced for the SEMS research project as well. Additionally, the research stimulated discussion about software process issues and their importance in the case organization.

6.2 Reliability of the Results

In order for the results of the resarch to constitute a contribution to the research field or the case organization, the study must demonstrate sufficient reliability. Reliability of a measurement is the result staying the same from one sample to the other, assuming that everything else remains the same.

Concerning the reliability of the results, the critical points of the research are selecting the participants for the workshop, constructing the workshop results, selecting the existing process model for evaluation, evaluation of the models creating the new model and conclusions for improvement steps for the case organization.

Defining the requirements and understanding them significantly affect to the rest of the research. This emphasizes the importance of selecting the right persons for the workshop. The chosen persons represented three different views: customer and sales point of view, strategic point of view and technology point of view. These points of view cover well the environment of the case organization. Taking into account the that selected persons have had a very active role in the company from the start of its business, the selection can be considered reasonable. One can ask why the CEO or the CTO were not participating. They were not invited since the roles they would have represented are similar to those already presented in the workshop. The number of the participants was also kept relatively small to ensure effective working in the workshop.

There is no recording from the workshop except the notes of the author and some flip chart drawings. The constructed workshop results were presented in this study using the similar 'mind-map' presentation style as was used in the workshop. This and the fact that the author was able to ask participants to specify unclear points later, makes the results from the workshop relatively reliable. It can be asked if a workshop was a proper way to find out the requirements or would interviews have been a better way for gathering the information. A workshop was selected because of its interactivity and efficiency. It is very likely that if a new workshop was held the resulting requirements would be qualitatively the same.

The process models evaluated were selected subjectively by the author. This introduces a possibility to have lacked some model that would have had an effect on the new model. Currently, there are not many research articles about using different software process models in a small software product company. This is why it can be considered reasonable to make a subjective selection of the models and still keep the result reliable enough for its purpose.

The results of the evaluation of the existing models can be found controversial. The evaluation was made by the author and a subjective opinion cannot be completely avoided. The main reason for this approach was that an objective or an experience-based evaluation was not found from the literature. Because the purpose of the evaltuation was to form a qualitative picture, not a quantitative measurement, of the models, the result of the evaluation can be considered reliable for its purpose. It is likely that if another evaluation were done on the models the result would be qualitatively the same.

The information about the existing process models was from literature that is available publicly and relatively widely spread in the field of software development. This information and the articles referred to can be considered a reliable source of information.

The main result of the research, the new model, is based on the requirements and the information about the existing models. Since they were both found reliable it is reasonable to expect that the resulting model is reliable as well. However, the model was constructed by the author and subjective opinions are evident. This is not necessarily a bad thing for the result since the author has been working in the product development of the case organization. One can of course ask, would the model be similar if someone else would have constructed it from the requirements and the existing models. Probably the main ideas would have been same but details might have been constructed differently.

Conclusion for improvement steps based on the current situation in the case organization and the new model have been made by the author. It can be asked why would this order of the steps be a proper one. The steps are constructed from a consistent set of practices that allow changing the deployment order of the steps.

Possibility to generalize the results without any modifications are limited. The new model is focused on small software product companies and it is based on the requirements defined by the experts from the case organization. If another company would like to adapt the new model it would require that

- the company has its focus on software products
- its business environment is not well established and it is highly competed
- the size of the company is small
- the technology used is new and frequently changing

In short, if the company shares the same requirements for a software process model, the new model with minor changes could be used for improving the software process of that company. Naturally, the improvement steps presented in this research depend remarkably on the current situation of the case organization and therefore cannot be generalized.

On the other hand, the approach to defining a new model can be generalized fairly well. If an organization defines requirements for its software development process, evaluates selected process models against them and constructs a new model based on that information, it is reasonable to expect that the resulting model is suitable for the organization.

6.3 Suggestion for Further Research

Even if defining a new process model is quite a large task for a research like this, many questions related to software product development were left unanswered. A few questions arise immediately after defining the new process model:

- How should the new model be promoted and supported?
- How did deployment of the new model affect the product development in the case organization?
- How should the customer project and delivery process be improved in the case organization to work efficiently with the new model?

The most important of all further research would be studying if the new model really helped the case organization to improve its software development process in practice.

The SEMS research project actually continues studying software processes of small software product companies. Therefore its results are interesting also from the viewpoint of this research.

Appendix A

Workshop Material

A.1 Invitation to the Workshop

This is an invitation email to workshop participant that was sent them before the workshop.

```
Ari, Antti and Lauri,

Would you have time for a two hour workshop? The subject is
improving product development (process) to better fulfill the
needs of the company's strategy, sales and customers. This is
of cource the needs of the whole company.

The goal of the workshop is to dig out those factors in the
product development that are important especially for the
sales and company's strategy. The questions that we will be
seeking answers for are:
- What are the essential things that define Smartner's
  operational environment (strategy, customers, technology)?
- What needs for the product development do these
  factors generate?
- What are the reasons behind the current practices? Are
  there more needs related to those reasons?

For example: Company's size compared to the customers is
small, which might lead to need for proving credibility by
having demo applications for customers.

Agenda for the workshop is shortly:
- Describing the problem and goals
- Brainstorm to work out essential things in environment
  and needs that they introduce for product development
```

```
- Reverse engineering current practices and working out
  needs related to them

Would Friday November 30th at 2PM be fine for the workshop?

--
Lauri Vuornos
```

Appendix B

Citations from the Literature

B.1 Reasons for death march projects

The reasons for the death march projects defined by Ed Yourdon in his book (Yourdon, 1999) are listed fully in the table B.1.

Table B.1: Reasons for death march projects

Politics, politics, politics.
Naive promises made by marketing, senior executives, naive project managers etc.
Naive optimism of youth: 'We can do it over the weekend!'
The 'Marine Corps' mentality: *Real* programmers do not need sleep!
Intense competition caused by globalization of markets.
Intense competition caused by the appearence of new technologies.
Intense pressure caused by unexpected government regulations.
Unexpected and/or unplanned crises - e.g., your hardware/software vendor just went bankrupt, or your best programmers just died of Bubonic Plague.

B.2 Project Burndown Graph in Scrum

The project burndown graph in Scrum visualizes the estimated work left and the estimated sprint completion. Each item in the sprint task list has an estimation for how many hours is still needed to finish the task. The project burn down graph is a sum of all those estimations. The estimate for sprint completion is derived from the current estimate of hours remaining and the amount of the working hours available per day which sets the slope of the curve. Estimation for completion is naturally the point where there is no more hours remaining. Since estimations can sometimes be initially incorrect the project burndown graph may also go up occationally.

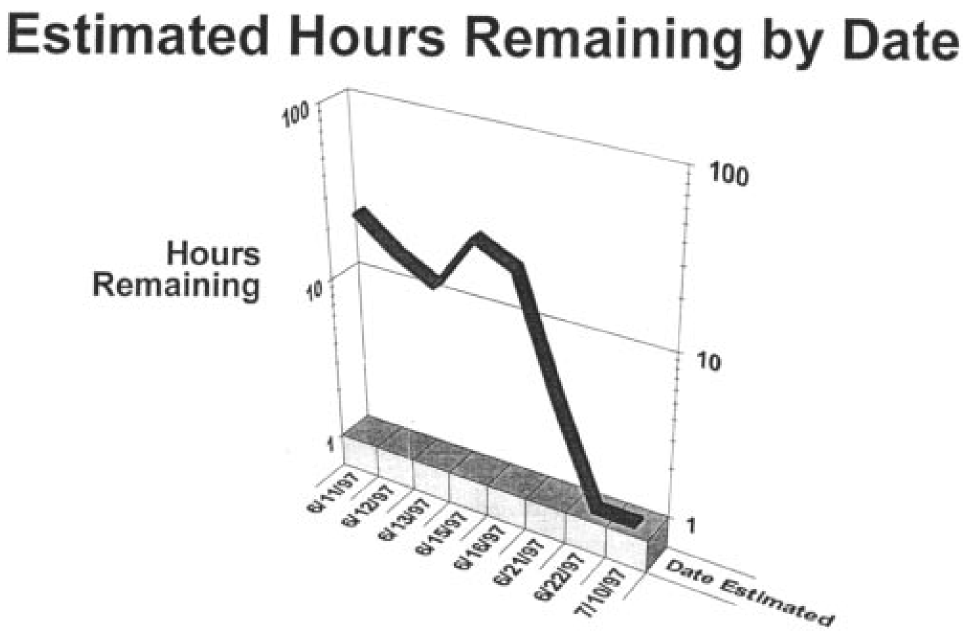


Figure B.1: Project burndown graph

Bibliography

Backholm, A. and Vuornos, L.: 2001, *Smartner Product Development Process in Short*, An unpublished internal document

Beck, K.: 2000, *Extreme Programming Explained: Emrace Change*, Vol. 3, Addison Wesley

Brodman, J. G. and Johnson, D. L.: 1994, in *Proceedings: 16th International Conference on Software Engineering*, pp 331–340, IEEE Computer Society Press / ACM Press

Cusumano, M. A. and Selby, R. W.: 1998, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People*, Touchstone

Demarco, T. and Boehm, B. W.: 1982, *Controlling Software Projects: Management, Measurement, and Estimates*, Prentice Hall PTR

Detlev, H. J., Roeding, C. R., Purkert, G., and Lindner, S. K. L.: 1999, *Secrets of Software Success: Management Insights from 100 Software Firms Around the World*, HBS Press

Dorling, A., Mackie, C., Smith, B., Lazinier, E., Pesant, J., Rand, B., Diaz, A., Winograd, Y., Campbell, M., Buchman, C., Azimi, A., Hodgen, B., and Shintani, K.: 1996, *SPICE, Consolidated product version 1.00*, Stadnard, ISO/IEC

Eisenhardt, K. M. and Brown, S. L.: 1998, *Harvard Business Review* pp 59–69

Elton, J. and Roe, J.: 1998, *Harvard Business Review* pp 3–7

Hafner, A. W.: 2001, *Pareto's Principle: The 80-20 Rule*, An unpublished WWW article

Jacobson, I., Booch, G., and Rumbaugh, J.: 1999, *The Unified Software Development Process*, Addison-Wesley

Järvenpää, E. and Aalto, P.: 2000, *Ohjelmistoyrityskysely 2000*, Technical report, TAI Research Center

Kaltio, T. and Kinnula, A.: 1998, in *Proceedings: SPI'98 at Monte Carlo*

Kaner, C., Falk, J., and Nguyen, H. Q.: 1999, *Testing Computer Software, 2nd Edition*, Vol. 2, John Wiley & Sons

Kotter, J.: 2001, *Harvard Business Review* pp 85–96

Kruchten, P.: 1999, *The Rational Unified Process: An Introduction*, Vol. 2, Addison Wesley

Kuvaja, P., Simila, J., Krzanik, L., Bicego, A., Koch, G., and Saukonen, S.: 1994, *Software Process Assessment and Improvement: the BOOTSTRAP approach*, Blackwell Publishers

Laitinen, M., Fayad, M., and Ward, R.: 2000, *IEEE Software* **17(5)**, 75

MacCormack, A.: 2001, *IEEE Engineering Management Review* **29(2)**, 90

McCormick, M.: 2001, *Communications of the ACM* **44(6)**, 109

Nielsen, J.: 1992, in *Proceedings: Human Factors in Computing Systems. (Monterey, Calif., May 3-7)). ACM/SIGCHI, New York*

Paulk, M. C., Curtis, B., Averill, E., Bamberger, J., Kasse, T., Konrad, M., Perdue, J., Weber, C., and Withey, J.: 1991, *Capability Maturity Model for Software*, Technical Report CMU/SEI-93-TR-024 ADA240603, Software Engineering Institute (Carnegie Mellon University)

Pollice, G.: 2001, *Using Rational Unified Process for Small Projects: Expanding Upon eXtreme Programming*, An unpublished Rational Software whitepaper

Probasco, L.: 2000, *The Ten Essentials of RUP. The Essence of an Effective Development Process*, Technical report, Rational Software Corporation

Rational: 1998, *Rational Unified Process Whitepaper: Best Practices for Software Development Teams*, Technical report, Rational Software Corporation

Rautiainen, K., Lassenius, C., Vähäniitty, J., Pyhäjärvi, M., and Vanhanen, J.: 2002, in *Proceedings: Hawai'i International Conference on System Science, January 1-10, 2002*

Sachs, G.: 2001, *The Impact of 3G Mobile Phone Services: Telecom Services*, Report, Global Equity Research

Schwaber, K. and Beedle, M.: 2002, *Agile Software Development with Scrum*, Prentice Hall

Smith, J.: 2001, *A comparison of RUP and XP*, a Rational Software whitepaper

Succi, G. and Marchesi, M.: 2001, *Extreme Programming Examined*, Vol. 1, Addison Wesley

Sun-Tzu: 1983, *The Art of War*, Delacorte Press

Takeuchi, H. and Nonaka, I.: 1986, *Harvard Business Review* pp 137–146

Varkoi, T. and Jaakkola, T.: 1999, in *Proceedings: PICMET´99, Portland, Oregon*

Yourdon, E.: 1999, *Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects*, Prentice Hall