UNIVERSITY OF OULU
DEPARTMENT OF INDUSTRIAL ENGINEERING AND MANAGEMENT
MASTER'S THESIS

Enhanced tool support for daily work management in agile software development

Author: Antti Haapala
Supervisor: Pekka Kess
Instructor: Ville Heikkilä
Date: July 2010

OULUN YLIOPISTO **Abstract of thesis**
Faculty of technology

Department	Laboratory
Department of Industrial Engineering and Management	
Author	**Supervisor**
Haapala, Antti Kalevi	Kess, P., Professor

Name of the thesis

Enhanced tool support for daily work management in agile software development

Subject	Level of studies	Time	Number of pages
Industrial engineering and management	Master's thesis	July 2010	75+19

Abstract

This thesis examines the features of a software tool that is needed for managing efficiently the daily work in agile software development methods, using the constructive research approach. Although agile software development is becoming more and more popular, agile software development management tools are lacking necessary capabilities for managing daily work in these methods. A part of current agile software development guidebooks are critical of existing software tools, and view them as inappropriate and inflexible for daily work management, compared to index cards or spreadsheet programs. Other sources emphasize that information systems are essential in some working environments.

The most popular agile software development methods, Scrum and Extreme Programming are examined. In the theoretical part the daily work practices of these methods and recommended tools are identified from the practitioner guidebooks. Based on the results of literature review, the support for identified daily work management practices in the open source Agilefant tool is reviewed prior to enhancing the support.

The requirements of case companies for daily work management were elicited in the user story format using the requirements workshop method. Acquired user stories were grouped into consistent features; these features were prioritized by the representatives in case companies using modified 100-dollar method. The results of prioritization were used to select a top priority features for deeper analysis and possible implementation.

10 practices and 5 different tools for daily work management were identified in the guidebooks. According to analysis, the support for most practices was insufficient in Agilefant. The workshop resulted in 26 different features that were not present in Agilefant; the ”team view” was considered the most important in the prioritization. 14 most important features, representing 80 % of votes cast are analyzed further: the analysis evaluates the candidate features based on the identified practices; a design for their implementation in the tool is proposed. Of these features, ”consolidated story and task list”, ”work queue”, ”task splitting” and ”strategy-to-action (bottom-up)” were implemented in Agilefant.

The results of this research can be utilized for example as a roadmap when creating a new software tool for agile software development management, but also to review the support for essential daily work management practices in existing tools.

Library location

Science and Technology Library Tellus, University of Oulu

Additional information

Keywords: Agile software development, product development, daily work management, project management, management information systems

OULUN YLIOPISTO **Tiivistelmä opinnäytetyöstä**
Teknillinen tiedekunta

Osasto		Laboratorio	
Tuotantotalouden osasto			
Tekijä		Valvoja	
Haapala, Antti Kalevi		Kess P., Professori	
Työn nimi			
Parannettu päivittäisen työn hallinnan työkalutuki ketterässä ohjelmistokehityksessä			
Oppiaine	Työn laji	Aika	Sivumäärä
Tuotantotalous	Diplomityö	Heinäkuu 2010	75+19

Tiivistelmä

Diplomityössä selvitetään tehokkaaseen ketterien ohjelmistokehitysmenetelmien päivittäisen työn hallintaan tarvittavia ohjelmistotyökalun ominaisuuksia konstruktiivisen tutkimuksen tutkimusongelmana. Vaikka ketterän ohjelmistokehityksen suosio kasvaa alati, on nykyisten ketterän sovelluskehityksen työkalujen tuki päivittäisen työn hallinnalle puutteellinen. Osa menetelmäkirjallisuudesta suhtautuu niihin ristiriitaisesti ja ne nähdään epätarkoituksenmukaisina ja joustamattomina päivittäisen työn hallinnassa suhteessa arkistokortteihin tai taulukkolaskentaohjelmiin. Kuitenkin toiset lähteet korostavat ohjelmistotyökalujen olevan välttämättömiä useissa toimintaympäristöissä.

Ketteristä ohjelmistokehitysmenetelmistä käsitellään suosituimpia, Extreme Programmingia ja Scrumia. Teoriaosuudessa näiden menetelmien päivittäisen työn hallinnan käytännöt ja suositellut työkalut selvitetään menetelmäkirjallisuuteen pohjautuen. Tuloksien pohjalta evaluoidaan avoimen lähdekoodin Agilefant-työkalun tuki päivittäisen työn hallinnan käytännöille ennen tuen parannusta.

Tapausyrityksien vaatimukset päivittäisen työn hallintaan selvitettiin käyttäjätarinamuodossa vaatimustyöpajamenettelyllä; saadut käyttäjätarinat ryhmiteltiin yhtenäisiksi ominaisuuksiksi, jonka jälkeen tapausyrityksien edustajat priorisoivat ne muunnetulla 100:n dollarin menetelmällä. Priorisoinnin perusteella valittiin osajoukko tarkempaa analyysia varten.

Menetelmäkirjallisuudesta tunnistettiin 10 päivittäisen työn hallinnan käytäntöä ja 5 eri työkalua. Analyysin perusteella Agilefant tuki useimpia käytäntöjä puutteellisesti. Työpajan pohjalta saatiin 26 eri ominaisuutta, joita ei ollut Agilefantissa. Näistä ”tiiminäkymä” arvostettiin priorisoinnissa tärkeimmäksi. 14 tärkeintä ominaisuutta saivat 80 % äänistä, ja näitä analysoitiin tarkemmin. Analyysissa arvioidaan ehdotettuja ominaisuuksia kirjallisuuden pohjalta sekä esitetään niille mahdollinen toteutus työkaluun. Näistä Agilefantiin toteutettiin uusina ominaisuuksina ”yhdistelty tarina- ja tehtävälista”, ”työjono”, ”tehtävän pilkkominen” ja ”strategiasta toimintaan (alhaalta ylös)”.

Tutkimuksen tuloksia voidaan käyttää toisaalta tiekarttana ketterän ohjelmistokehityksen työkaluja laadittaessa, toisaalta myös arvioitaessa olemassa olevien työkalujen tukea päivittäisen työn hallinnan tarpeellisille käytännöille.

Säilytyspaikka

Oulun yliopisto, Tiedekirjasto Tellus

Muita tietoja

Avainsanat: Ketterä ohjelmistokehitys, tuotekehitys, päivittäisen työn hallinta, projektinhallinta, johtamisen tietojärjestelmät

Preface

This thesis has been written for and funded by the ATMAN project (Approach and Tool support for development portfolio MANagement) at the Software Business Research Group of Software Business and Engineering Institute at the Aalto University School of Science and Technology. The three-year ATMAN project is jointly funded by the Finnish National Technology Agency TEKES and participating software companies (F-Secure, PAF, eCraft, IPSS, Napa, Mipro and Tekla). The goal of ATMAN project is "to help the Finnish Software Industry better link business strategies and long-term product development planning with daily work through managing the developers' efforts as an explicit portfolio and providing an understanding of the needed tool support".

I want to express my gratitude to all those people who guided me in the creation of this thesis, especially Lic. Sc. Jarno Vähäniitty for getting me started on this field, M. Sc. Ville Heikkilä for valuable insights and helpful instruction, and all other personnel at Software Process Research Group and SoberIT. I would also like to thank Reko Jokelainen and Pasi Pekkanen for a crash course into J2EE world and the technology of Agilefant, and the workshop participants and those interviewees who participated into feature prioritization; without them the empirical part of this thesis would not have been as fruitful.

Special thanks go to Donald Knuth for the TEX typesetting program, to the creators of LATEX2e for the excellent macro package, to the LYX team for their wonderful WYSIWYG TEX editor, and to the Inkscape project for the free vector drawing program. I cannot imagine how I would have managed without these valuable tools.

In Oulu, Finland, on July 1, 2010

Antti Haapala

Abbreviations

4CC	*The Four Cycles of Control model*
AHP	*Analytical Hierarchy Process*
BL	*Backlog*
BLI	*Backlog Item*
DW	*Daily Work*
EL	*Effort Left*
ES	*Effort Spent*
IID	*Iterative and Incremental Development*
J2EE	*Java 2 Platform, Enterprise Edition*
OE	*Original Estimate*
NPD	*New Product Development*
PG	*Planning Game*
PWC	*Pairwise Comparisons*
SQL	*Structured Query Language*
SP	*Story Point*
XP	*Extreme Programming*

List of Tables

List of Figures

Contents

1 Introduction

This thesis concerns the management of every-day efforts of software developers in agile software development teams. Currently, the management tools that are considered most *usable* by a majority of agile software development practitioner literature are pencils, index cards, whiteboards and spreadsheet files. The aim of this thesis is, if not remove, then at least to narrow the gap between these low-technology tools and management information systems. This chapter starts with an introduction to agile methods and daily work management (1.1), which is followed by the motivation of this thesis (1.2). The specific case tool under study, Agilefant, is presented in Section 1.3. Finally, the research problem, approach and research questions are introduced in Section 1.4.

1.1 Agile methods and agile software development

The history of software development has seen two competing project management paradigms — the single-pass sequential development paradigm, which is also known as the waterfall model, and the iterative and incremental development (Larman & Basili 2003). Even though Royce (1970) had criticized the straightforward sequential development in his famous article, the single-pass sequential model was still for the following two decades considered the ideal model for software development. Only in the 1990s the IID paradigm started gaining wider support. (Larman & Basili 2003.) In February 2001, seventeen experts on light-weight IID processes convened in Utah and produced a declaration, the Agile Manifesto (Fowler & Highsmith 2001), which defines the four common values of the methods they represented. Thereafter those light-weight IID methods which also support these values have been called agile methods (Larman & Basili 2003). Of these agile methods, Scrum and Extreme Programming are currently most widely adopted (VersionOne, Inc 2010).

1.1.1 Daily work management in agile methods

Rautiainen *et al.* (2002) have presented the Four Cycles of Control model for software development governance. In 4CC, the topmost control cycle is the strategic release management, which divided into multiple release cycles. Each release cycle consists of several iteration cycles, and each iteration cycle is divided into mini-milestones. Each outer cycle includes the planning of its immediate inner cycles, and monitoring their execution. The inner cycles in turn provide feedback to the immediate outer cycles. The length of a mini-milestone is usually one day or one week at maximum. (Rautiainen *et al.*

2002.)

In this thesis the concept *daily work* is defined as all the work that the agile development teams and their individual team members shall do within these mini-milestone cycles to produce a software increment that fulfills iteration goals. The mini-milestones are planned in advance and represented in an iteration plan. The concept *daily work management* shall be defined as all management and coordination practices that development teams and individual developers employ during these mini-milestones, including feedback to the iteration cycle. The *goal of daily work managemen*t is to maximize the *sustainable* velocity at which the team and individual developers can produce value in the software increment during the iteration. This definition includes the refining of iteration plan, as it is considered to be feedback to the iteration cycle; however, it specifically excludes management decisions that change the scope of iteration or release; as these are considered belonging to the higher cycles instead.

1.1.2 Scrum

Takeuchi & Nonaka (1986) had studied the new product development in non-software technology projects, and based on their empirical findings proposed that the old model of new product development, where the "product development process moved like a relay race, with one group of functional specialists passing the baton to the next group", should be replaced by a *new* new product development game, wherein a single multidisciplinary team would be responsible for the product development through all these phases; the phases would also be overlapped, or the entire process integrated so that one cannot distinguish the phases from one another. Takeuchi and Nonaka compared this product development game to rugby union.

Schwaber (1995) and Beedle *et al.* (1999) transferred the ideas of Takeuchi & Nonaka into the field of software engineering, combining it with the ideas of IID in software development into a new model called Scrum, (a metaphor from the rugby union terminology). According to Beedle *et al.* (1999) several rigorous process models such as the Capability Maturity Model are incorrect, because in promoting repeatable and defined processes these models tend to assume that factors inside and outside factors are repeatable or defined. In their view, one should not try to remove uncertainties but use a process model that can adapt to outside chaos. The Scrum process described by Schwaber (1995) is not only iterative and incremental process, but also empirically managed. During the development phase the self-organizing development teams work in fixed-length iterations called sprints. When sprinting, a team has no defined process. Furthermore sprints are not linear like waterfall, but they consist of numerous non-predetermined interactions between team members. Also, because explicit process knowledge is not available, the team will gradually refine its own development process through trial and error and by employing tacit knowledge. The process is also inherently risk-driven, and open, allowing the project scope and direction to be changed after every sprint. (Schwaber 1995.)

1.1.3 Extreme Programming

While Scrum is more a general project management framework which can be used for other kinds of project management (Schwaber 2004), Extreme Programming is strictly a discipline for software development. Extreme Programming prescribes twelve software engineering practices; some of them are more management oriented, such as *onsite*

customer, *planning game*, *40-hour week* and *short releases*, whereas others are more specifically software development oriented, such as *refactoring*, *pair-programming*, or *continuous integration*. Extreme Programming is too an empirically managed model, with releases taking place in short iterations; the Extreme Programming team does not either have a defined workflow during the iteration; the only constraint is that all programming work should be done in pairs. (Beck 1999.)

Though in principle close to Scrum, it deviates by promoting the software engineering practices extensively, whereas in Scrum it is the duty of the self-organizing team to choose their preferred engineering practices. Scrum also emphasizes the need of continuously widening the constraints of team work, or removing impediments in Scrum terminology; this is not emphasized at all in Extreme Programming. (Schwaber & Beedle 2001, Beck 1999.) Due to the complementary nature of these two methods regarding management and engineering practices, they are often combined in software development organizations (VersionOne, Inc 2010).

1.2 Motivation

Originally the coordination of daily work in agile software development methods was accomplished through using simple management tools. Post-it notes on the wall depicting various engineering tasks and customer requirements, hand-drawn measurement charts on whiteboard, or spread-sheet files stored on a network drive were the primary management tools and information systems, along with flexible management practices such as daily stand-up meetings, follow-up meetings, impediment removal, task estimation and subtle status tracking. (Beck 1999, Schwaber & Beedle 2001).

The existing agile software development literature on Scrum and Extreme Programming is still divided on the relative advantages and disadvantages of specialized management information systems. A portion of professional agile software development guidebooks advocate the use of specialized software for managing the teamwork within an iteration (Auer & Miller 2001, Leffingwell 2007), some books ignore the issue altogether (Schwaber 2004), while some take a negative stance against replacing the simple management tools with any kind of technical solution (Beck & Andres 2004). Larman & Vodde (2008) denounce many of the existing agile project management software tools, because in their opinion they are biased towards easy reporting while being cumbersome to use by the actual team members. Still these simple management tools are not sufficient in larger-scale agile organizations, as additional work is needed to reflect back the progress of the team in product backlog (Lehto & Rautiainen 2009). Reports also show that some kind of software tool is necessary when the team and its members are working on more than one project at a time (Scotland 2003), or if the development team is distributed (Sutherland *et al.* 2007).

Thus to be a replacement for existing methods, an agile software development management tool has to be designed to be more efficient for managing the daily work and mini-milestones than the pencil and paper, or spreadsheets. To accomplish this, the details of daily work management in agile software development methods have to be thoroughly studied. Furthermore, to ensure that the design is relevant in practical setting, it has to be ensured that the solutions satisfy the needs of actual users of the management system. This sets the premise for the research methodology and the research problem of this thesis.

1.3 Agilefant, a proof-of-concept tool

The ATMAN project has produced a proof-of-concept software tool for agile software development called Agilefant. It is mainly used for validating the research results of the project (Vähäniitty & Rautiainen 2008). The foundation of Agilefant is backlog management; backlogs are containers of future work to be done on the software product. In Agilefant, the product development is managed in a hierarchy of backlogs consisting of product, project and iteration backlogs. These backlogs correspond to 4CC (Rautiainen *et al.* 2002) strategic release management, release and iteration cycles respectively. Several views are provided to inspect different aspects of this data and to modify it; these include a specialized backlog view for each kind of backlog, the timesheets view that is used to log work time and generating reports, and the project portfolio view that can be used to prioritize simultaneous release projects in the project portfolio.

Agilefant is realized as a J2EE web application that can be deployed on any compatible servlet container. The data is stored into an SQL database; Agilefant is built to primarily support the MySQL database. The users can interface with Agilefant using an up-to-date version of any of the popular web browsers. Agilefant employs JavaScript client-side programming, which makes the user interface responsive. Most views are updated automatically to reflect the concurrent edits made by other users. This effectively provides the user with a real-time picture of the current state of development efforts on each view without manual refreshes.

As a readily available, open-source and feature-complete research tool, Agilefant shall be used to validate the results of this thesis. Agilefant has already been adopted by several companies, thus providing an existing user base for testing the new approaches to daily work management.

1.4 Research problem, approach and research questions

The goal of this thesis is to construct a set of new features for a software tool to better support the team-level efforts at agile software development organizations and those organizations that are transitioning to agile. The research problem of this thesis is *"How to support effective daily work management in an agile software development management tool?"* The research problem is further refined with following research questions:

Q1 What are the specific practices of managing the daily work of a self-organizing team and of its individual members in the selected agile software development methods?

Q2 What tools does the literature suggest for supporting the implementation of aforementioned practices?

Q3 What kind of support does unmodified Agilefant provide for these practices?

Q4 What features are requested by the key stakeholders of software development organizations to be implemented in an agile software development management tool for managing daily work?

Q5 What is the priority order of these features according to the key stakeholders at these software development organizations?

Q6 How a top-priority subset of these features could be supported in an agile software development management tool while ensuring compatibility with existing agile software development methods?

The problem is approached by the means of constructive research; the constructive research approach aims at building new innovative constructions to solve existing problems (Kasanen *et al.* 1993). In this thesis, a set of requirements for an agile software development management tool is elicited and analyzed based on existing theory, and used then to create new innovative constructions: proposed designs for features to be implemented in management tools. The validation of these constructions is based on linking them with existing theory, and by providing a tested and thoroughly validated implementation of a subset of them. The nature of constructive research (Lukka 2001) demands that the studied problems are present in real world and solving them has practical relevance (Q4, Q5); that the construction is connected to the existing theory (Q1, Q2, Q3) and that the construction contributes to the current scientific knowledge and it can be used in practice (Q6). The details of research process are discussed in Chapter 2.

2 Research process and methodology

In the beginning the scope of the research was determined — Scrum and Extreme Programming as the currently most popular (VersionOne, Inc 2010) agile software development methods were chosen to be studied. Accordingly, the literature review of this thesis concentrates on the management practices in these two specific methods. Independent from the literature review, a requirements workshop (Leffingwell & Widrig 2003) on daily work management was held for agile software development organizations. The goal of this workshop was to elicit a set of candidate features for daily work management in an agile software development management tool. After the workshop, the participating companies were then independently asked to prioritize the candidate features from their point of view. The results of this prioritization were used to choose a top-priority subset of candidate features for further study The results of literature review were used to analyze this subset of features to produce proposed designs for these features. Finally a portion of these designed features were implemented in Agilefant. A figure of the research process is provided in Fig. 1.

2.1 Overview of the research process

The research questions Q1 and Q2 are answered based on the literature review that was conducted on practitioner guidebooks on the selected agile software development methods. The selection of book sources is detailed in Section 2.2. The book review process is described in Section 2.3. Based on the results of the literature review, prior support for daily work management practices is analyzed in Agilefant; this answers to the research question Q3. The process is detailed in 2.4.

The research question Q4 is answered by the results of a daily work management workshop that was held for representatives from agile software development organizations. The workshop is detailed in Section 2.5. Answer to Q5 is provided by a multi-stakeholder prioritization process on the features that were proposed in the workshop, which is discussed in detail in Section 2.6.

The research question Q6 is answered by synthesizing the answers to research questions Q1–Q5. Based on the prioritization results (Q5), a top-priority set of the proposed features (Q4) was selected for further analysis. This set was analyzed based on the practices and tools identified in the agile software development practitioner guidebooks (Q1 and Q2). Due to the independent nature of these features, a subset of them was implemented in the existing proof-of-concept agile software development tool; another subset was left for future implementation. Workarounds for most of these unimplemented features are

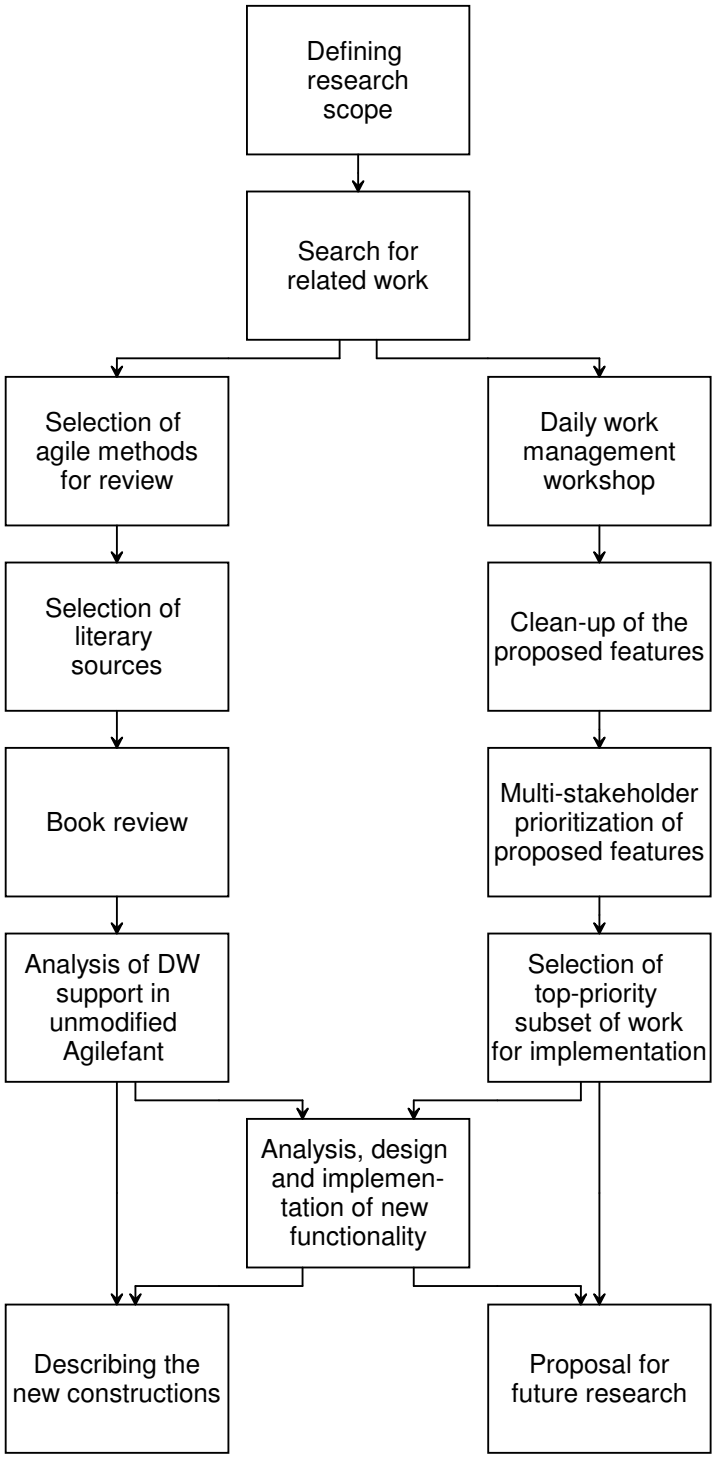

Fig. 1. The research process

proposed based on the present functionality in the proof-of-concept (Q3). This analysis, design and implementation process is discussed in more detail in Section 2.7.

2.2 Selection of the book sources

Due to limited timeframe of the thesis, the primary factor for the selection of the books was their availability. Among the available books, those books that had sections concentrating

on management of work within iterations in Extreme Programming, Scrum, or both, were chosen for further review. Table 1 lists the 12 books that were considered for book review; the third column tells whether or not the book was actually included in the review. After a quick survey, 9 of these 12 were selected for the literature review.

Extreme Programming Explained: Embrace Change (Beck 1999) was the first book to describe the Extreme Programming, and is thus a necessary choice for the book review. Auer & Miller (2001) call it as "the manifesto for Extreme Programming". Similarly, *Agile Software Development with Scrum* (Schwaber & Beedle 2001) was the first guidebook on Scrum methodology. Both books are based on actual experiences on these methods. The *Extreme Programming Explained: Embrace Change: 2nd edition* (Beck & Andres 2004) was published 5 years after the first one; it actually is somewhat more than the title would suggest in that it is a complete revision of the book, and significant update to the previous; it was deservedly included in the review. *Agile Project Management with Scrum* (Schwaber 2004) is an update on the first Scrum book; it highlights the nature of Scrum, and its application to real world organization, through case studies. As these case studies also highlight the management practices in their proper context, this book was also included in the review.

As the Extreme Programming Explained books concentrate more on explaining the philosophy of Extreme Programming, and the details on actual management practices are discussed only vaguely, other books from Extreme Programming series were also considered. *Planning Extreme Programming* (Beck & Fowler 2000) was included, because it focuses only on planning and tracking software development in Extreme Programming projects, down to day level. *Extreme Programming Applied: Playing to Win* (Auer & Miller 2001) focuses on describing how Extreme Programming can be implemented in real organizations. It was also included in the review.

The third Scrum book by Schwaber (2007), *The Enterprise and Scrum*, was considered, but it was not included in the review because it did not seem to provide any new information on daily work management beyond the two original Scrum books. However, Schwaber (2007) recommended Cohn's (2005) *Agile Estimating and Planning*, which was included in the review; like Beck & Fowler (2000), the book not only focuses on planning but goes down to day level and also provides insight on management of work within these plans.

Questioning Extreme Programming (McBreen 2002) was suggested by one senior researcher as a "thought-provoking" book on Extreme Programming. However, according to the foreword, the author of the book has never worked on an Extreme Programming project. As the book does not describe the development process of Extreme Programming, but remains a high-level discussion of the supposed advantages and disadvantages of Extreme Programming in various situations. Thus it was left out of the book review. *Manage it! Your guide to modern, pragmatic project management* (Rothman 2007) was another recommended book. Even though it addresses daily work management, and Extreme Programming and Scrum, it does not discuss only them, or even only agile methods. Thus, it does not answer the first research question — "What are the specific practices of managing the daily work of a self-organizing team and of its individual members in selected agile methods?" This led to its exclusion from the review.

Scaling Software Agility: Best Practices for Large Enterprises (Leffingwell 2007) and *Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum* (Larman & Vodde 2008) represent the newest literature on agile methods. They both contain very detailed discussion of daily work management, and they both highlight issues that arise especially in large-scale agile organizations, that is, organizations that have many self-organizing teams working on the same product simultaneously. Both

Table 1. Books considered for the book review

Title	Method	Included
Extreme Programming Explained: Embrace Change (Beck 1999)	XP	Yes
Planning Extreme Programming (Beck & Fowler 2000)	XP	Yes
Agile Software Development with Scrum (Schwaber & Beedle 2001)	Scrum	Yes
Extreme Programming Applied: Playing to Win (Auer & Miller 2001)	XP	Yes
Questioning Extreme Programming (McBreen 2002)	XP	No
Agile Project Management with Scrum (Schwaber 2004)	Scrum	Yes
Extreme Programming Explained: Embrace Change (2nd Edition) (Beck & Andres 2004)	XP	Yes
Agile Estimating and Planning (Cohn 2005)	General	Yes
Manage it! Your guide to modern, pragmatic project management (Rothman 2007)	General	No
The Enterprise and Scrum (Schwaber 2007)	Scrum	No
Scaling Software Agility: Best Practices for Large Enterprises (Leffingwell 2007)	General	Yes
Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum (Larman & Vodde 2008)	Scrum	Yes

books were included in the review.

2.3 The book review process

In the book review process, each reviewed book was read in entirety. At first, the key terms and concepts related to daily work management were recorded and their usage explained. Based on these findings, the relations of basic iteration level concepts in both of the studied methods were also represented in an UML 2.0 class diagram. Then 10 general practices of daily work management were identified in the sources (see Table 2 on page 22). Different approaches to these 10 practices were searched from the sources. During the review process all references to suggested tangible or software tools for these for these practices were also recorded. The results of book review are represented in Chapter 3.

2.4 Analysis of daily work support in the prior version of Agilefant

The conceptual model of the most central concepts in Agilefant prior to implementation of any new features was drawn as UML class diagram. Then, based on the results of the book review, the support for identified daily work management practices was evaluated in

Agilefant. For each identified practice, Agilefant was evaluated for features that would support the practices as written in the books. The results of this analysis are provided in Chapter 4.

2.5 Daily work management workshop

After the research scope was defined, a requirements workshop (Leffingwell & Widrig 2003) was held for case companies. The companies sent 1–3 representatives to participate in a 2.5 hour long workshop. The workshop produced a set of candidate features for the next revision of Agilefant. Each participant had a laptop and thus could work independently from others.

A total of 12 people participated in the daily work management workshop. As a warm-up exercise, all participants were asked to write down independently of each other for five minutes what kind of practices they would consider to belong to good daily work management as a team member or management role in an agile team. The results were then shared among all participants and discussed so that a shared mindset was achieved. Then the participants were divided into three teams. One of the teams had its participants from companies whose business is subcontracting work, another had its members drawn from companies who had their own software products. The third team consisted of agile software development researchers, who all had first-hand experience from agile software development.

The teams were tasked to write candidate user stories of things they as team members, business owners, product owners or Scrum masters would consider having value for daily work management in an agile software development management information system. The user stories were to be written using the detailed format "As a ≪*role*≫, I want ≪*function*≫, so that ≪*business value*≫" (Cohn 2004). Each team worked on a shared spreadsheet, which allowed them to view in near real time what the other team members were writing. However the three teams were working independently from each other and thus possibly produced a partially overlapping set of stories.

2.5.1 Cleaning up the stories

The workshop was followed by a clean-up by the author. If the story had two pieces of functionality separated by the "and" conjunction, they were split into two. Of the resulting stories, stories that detailed duplicated functionality were dropped; also were dropped stories that were clearly out of scope of even the iteration cycle (Rautiainen *et al.* 2002). Finally, those stories that would be assumed to be fulfilled by every viable backlog management system, and thus not novel, were also discarded; that is, if the user story was satisfied with unmodified Agilefant, it was discarded. These stories were refined and grouped under features by the author. The results of daily work management workshop and clean-up are provided in Chapter 5.

2.6 Multi-stakeholder backlog prioritization

The outcome of the daily work workshop was a set of independent candidate features which constituted a Scrum product backlog (Schwaber & Beedle 2001) for a software tool. As a product backlog in Scrum is a prioritized list of work proposed to be done on the

product, the candidate features had to be prioritized. To ensure that the prioritization of features reflected the business value of these features, the case companies that participated in the daily work workshop would be asked to prioritize the features independently of each other, and the individual prioritization results to be combined to produce the final prioritization.

2.6.1 Prioritization methods

Most prioritization methods provide results on either ordinal or ratio scales (Karlsson *et al.* 2006). On ordinal scale, each item is assigned to a class, and the relative order of the class are known; however an ordinal scale does not provide information on the value of each class. Furthermore, as all items within a class are considered equal to each other, nothing can be said of their relative priorities, unless they are ranked again using a more fine-grained ranking. Examples of requirement prioritization methods that use ordinal scale include the numeral assignment technique (Karlsson 1996) and the Planning Game of Extreme Programming (Beck 1999).

On ratio scale not only is the order of items known, but also the ratio of values of any two items, which allows detailed value analysis. According to Karlsson *et al.* (2006), examples of prioritization methods that use a ratio scale include the cumulative voting scheme, also known as the Hundred Dollar Test (Leffingwell & Widrig 2003), Pair-wise comparison (Saaty 1990) and Wiegers' method (Wiegers 1999). The disadvantage of ratio scale methods is the perceived difficulty compared to ordinal scale methods, and greater time consumption (Karlsson *et al.* 2006).

The pair-wise comparison method was originally a part of the Analytical Hierarchy Process (Saaty 1990). The advantage of pair-wise comparisons over the other ratio scale methods is the redundancy for an AHP PWC matrix a metric, consistency ratio can be computed, which measures the internal consistency of the comparisons in the matrix. The disadvantage of pair-wise comparison is that it is time-consuming, because to prioritize a set of features one has to do a comparison between each feature-pair, and estimate the relative values of the pairs on a ratio scale. To prioritize n features, AHP requires $\frac{n(n-1)}{2}$ comparisons. (Karlsson *et al.* 1998, 2007.) Software tools for incomplete pairwise comparison have been created; these significantly decrease the amount of needed comparisons (Karlsson *et al.* 1997).

In the hundred dollar test each stakeholder is given a fixed amount of votes — usually 100, hence the name — which they can distribute according to their consideration among the features (Leffingwell & Widrig 2000). The hundred dollar test does not contain the consistency checks that are inherent in redundant pair-wise comparisons like those in AHP, but it is much simpler to implement. A study by Hatton (2007) showed that the hundred dollar test to be on average 4 times less time consuming than the pair-wise comparison utilized in AHP, although it provided comparable results. Furthermore, the decision makers were more confident that the results reflected their intentions accurately.

2.6.2 Selection of the prioritization method

A proven method had to be chosen for prioritization. A ratio-scale method would be needed to make the analysis of the business value of the prioritized features (n = 26) possible. The main criterion was that the liaisons of stakeholders could use it without any outside support, and with ease. As no suitable tool was found that could facilitate

prioritization by incomplete pair-wise comparison, and using an ordinary PWC matrix would have required 325 pair-wise comparisons to be made manually, it was decided that modified hundred dollar method would used.

2.6.3 Gathering prioritization data

After the user stories were processed and the feature list constructed, a prioritization questionnaire was created. The questionnaire was realized as a spreadsheet, which automatically calculated the amount of votes casts and remaining, and displayed these numbers prominently. As there were 26 features, the amount of credits was increased to 1000 to allow more fine-granular prioritization. The interviewees were asked to divide their votes among the features based on the needs of their organization. In the questionnaire each feature was briefly described, and followed by its constituent user stories. The interviewees received and answered the questionnaire via e-mail. The instructions of the questionnaire are in Appendix II, and the questionnaire form is in Appendix III.

The spreadsheet was sent by e-mail to the representatives in the companies. They then returned the spreadsheet files filled with their preferred distribution of the votes. The final prioritization was done by calculating the number of votes cast in favor of each independent feature. The results of the prioritization are presented in Chapter 5.

2.7 Analysis, design and implementation of selected features

Each feature among the top-priority subset was analyzed based on practices identified in the book. A design was proposed for each of these features; wherever a context for design was needed, the unmodified version of Agilefant (Chapter 4) was used as a base. A subset of the designed features was implemented in Agilefant for further validation. For those features that were not yet implemented in Agilefant, a workaround was proposed if possible. The workarounds were based on the feature set present in unmodified version of Agilefant and the newly implemented features. The results of analysis, design and implementation phase are presented in Chapter 6.

3 Management of work within iterations according to book and article review

This chapter describes the results of the book review on the suggested management practices of daily work within a Scrum sprint or an Extreme Programming iteration. The Scrum and Extreme Programming practices presented in this chapter are as described in the books selected in the review; they should not be considered to be complete examples of these methodologies. Any such practice that is not ordinarily done within development iterations are not considered here. Furthermore, only such practices that concern the developer team itself (including the Scrum master role but excluding the product owner) are considered.

The following concepts repeatedly occur in this text; the exact meanings of the terms in the context of this thesis are defined in Section 3.1.

- team and roles (3.1.1)
- iteration or sprint (3.1.2)
- sprint goal (3.1.3)
- story (3.1.4)
- task (3.1.5)
- backlog (3.1.6)
- impediment (3.1.7)

Based on the book review the following 10 practices were found to be related to management of daily work in Scrum and/or XP. The practices and the sources where they were identified are presented in Table 2. A detailed review of these practices based on these sources is presented in Section 3.2 on page 29.

3.1 Definition of terms and high-level concepts

The Scrum and Extreme Programming high-level concepts and terminology related to daily work management is described under this Section.

3.1.1 Team and roles

Both Scrum and Extreme programming specify that the development is handled by a team, which shares the collective responsibility for its goals. The team in both methodologies is cross-functional and self-organizing; it contains all necessary skills required to implement

Table 2. The management practices discussed in the sources.

Management practice	Source								
	Beck (1999)	Beck & Fowler (2000)	Auer & Miller (2001)	Schwaber & Beedle (2001)	Schwaber (2004)	(Beck & Andres 2004)	Cohn (2005)	Leffingwell (2007)	Larman & Vodde (2008)
Visual management	x	x	x	x	x	x	x	x	x
Assigning responsibles	1	1	1,2	1	1	1, 2	2	1	2
Selection of next task	x	x	x	x	x	x	x	x	x
Status tracking	x	x	x	x	x	x	x	x	x
Lightweight measurements	x	x	x	x	x	x	x	x	x
Status-update meetings	x	x	x	x	x	-	-	x	x
Measuring and balancing load	x	x	x	x	x	x	x	-	x
Impediment handling	-	-	x	x	x	-	-	x	x
Dedicating team members	-	-	-	x	x	-	-	-	x
Maintaining focus and establishing cadence	-	-	-	-	-	-	x	x	x

Legend: *x = the book discusses the practice.. - = the book does not discuss the practice. 1 = the book promotes the "one at a time" task allocation strategy. 2 = the book promotes the "fill your bag" task allocation strategy*

a selection of customer requirements and transform them into a complete executable product increment within a fixed time box. In software development, such cross-functional team should contain analysts, designers, quality control and coding engineers. (Schwaber & Beedle 2001, Beck 1999.)

According to Schwaber & Beedle (2001) there are three defined roles in a Scrum team: Scrum master, product owner and team member. The developers — team members — together form the Scrum development team, a part of the Scrum team, which is responsible for engineering work. The *Scrum master* serves as a coach and facilitator for the Scrum development team; the *product owner* works as a proxy between the external stakeholders and the development team, and is thus a kind of project manager. A Scrum team is self-organizing and self-managing; the recommended number of members in such self-managing team should be no more than 9, because the communication between the team members is impeded as new members are added. Schwaber and Beedle also provide anecdotal evidence that larger teams, if allowed to organize themselves freely, would subdivide into smaller subteams. The responsibilities of a Scrum team include, but are not limited to analysis, interaction design, internal design, programming and testing. However only the product owner and the Scrum master have their special roles; the other team members do not have titles. (Schwaber & Beedle 2001.)

In Scrum all work that is to be done is first defined coarsely by the product owner, which must be a designated single person, not a committee or team. The product owner negotiates with all stakeholders, including the team, the priority of requirements. A top-priority subset of these requirements is then implemented in time-boxed iterations, or *sprints* in

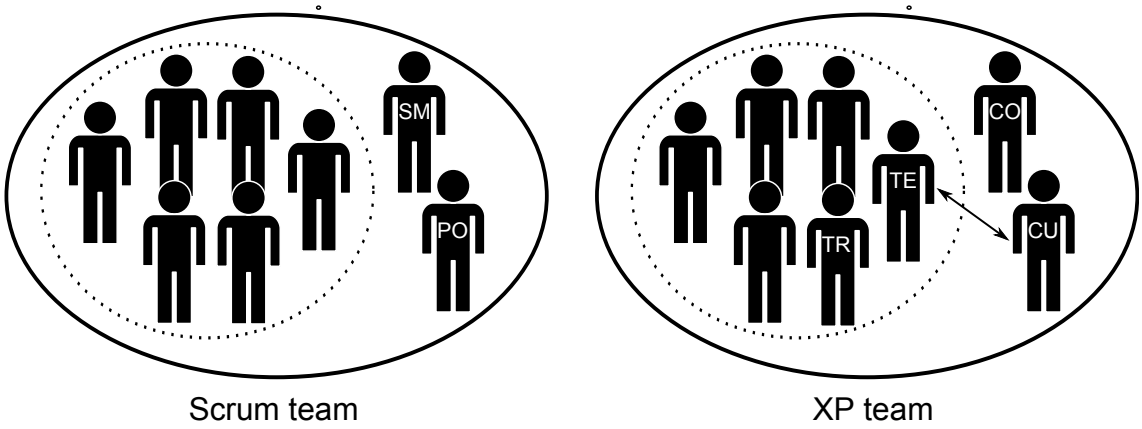

Fig. 2. The structure of the Scrum team (Schwaber 2004) and the Extreme Programming team (Beck 1999). SM = Scrum master, PO = Product owner, CU = Customer or customer representative within the team, CO = Coach, TE = Tester, TR = Tracker. The dotted ellipse surrounds the developer roles. The tester in Extreme Programming is the dedicated customer liaison among the developers.

Scrum terminology; the product owner has to negotiate the subset of requirements that shall be committed to be completed within a sprint with the Scrum development team. The Scrum development team as a whole is responsible for trying to accomplish the goals it has committed to. (Schwaber & Beedle 2001.)

In the Extreme Programming the team size can be more varying: Beck (1999) refers to projects with only two people - the minimal requirement for pair programming which is a central technique in XP, the upper limit being approximately 10 persons. Beck (1999) further identifies the roles of *customer*, *coach*, *tracker* and *tester*. Unlike in Scrum, where the product owner role is internal to the organization, the customer in XP is an outsider, and can be also considered consisting of many stakeholders, who have to work collectively. As in Scrum, only the customer role can prioritize the requirements or features. Beck (1999) proposes that the team could ask the customer to help in writing the tests. The coach and tracker roles together loosely correspond to the role Scrum master; the Tester role helps the customer implementing functional testing on the product, and could be someone of the programmers. Beck (1999) also recommends that the customer should be someone who will actually be using the end product.

In the context of this thesis the term team refers in Scrum to the Scrum team, including the Scrum master and the product owner roles. Likewise in XP, the team consists of the programmers, the Coach, the Tracker and the Tester roles; a customer, if he is involved in the development, can also be considered a team member.

3.1.2 Iteration

Both Scrum and XP suggest doing the development during relatively short iterations, usually only 1 to 4 weeks. (Schwaber & Beedle 2001, Beck 1999, Larman & Vodde 2008, Leffingwell 2007). In an iteration the team is supposed to build new functionality having value to the stakeholders and ensure that at the end of the iteration the software product is potentially shippable, without causing regression on those features that have been built in previous iterations (Schwaber & Beedle 2001, Beck 1999). The length of an iteration is always fixed beforehand (Schwaber & Beedle 2001, Beck 1999), and usually

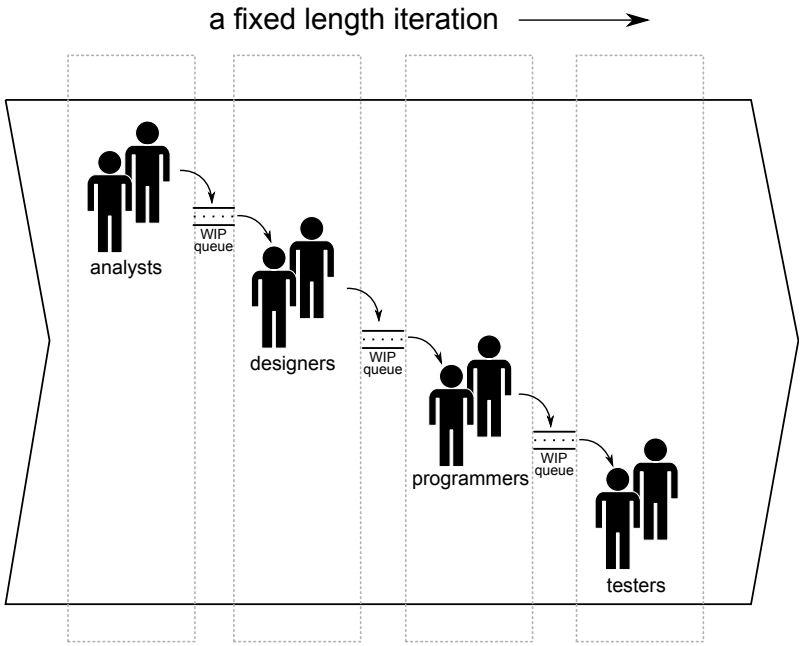

Fig. 3. The mini-waterfall anti-pattern (Larman & Vodde 2008), which is a remnant of an matrix organization, is a common organizational anti-pattern in immature agile organizations. In Extreme Programming and Scrum there are no specialist roles, and optimally no artifact handovers within an iteration (Schwaber & Beedle 2001, Beck 1999).

also throughout the project lifespan (Schwaber & Beedle 2001). However, Beck (1999) proposes that early into product life-cycle the length could be longer, three weeks, whereas in the "productionizing" stage the iteration length should be considerably shorter than in the beginning, to allow rapid feedback from the customer.

As the team is cross-functional without any specific roles beyond those described in Subsection 3.1.1, it also has no definite workflow during an iteration. Instead, all team members work on all possible aspects of the development. There is also no defined organizational structure within the team, or the iteration, the team is fully self-organizing (Fig. 4). (Schwaber & Beedle 2001, Beck 1999, Larman & Vodde 2008). Optimally there is also no handover of work in progress from one member to another (Larman & Vodde 2008). This can be contrasted with an organizational antipattern called mini-waterfall (Fig. 3), which is an iterative pattern as all value is not delivered at once, but has separate functional groups or teams doing analysis, design, programming and testing. The mini-waterfall will increase the lead-time of new features and thus decrease the adaptability of the organization as well as the value-ratio. The Work in Progress queues and buffers aggravate the delay further due to natural variability in story or backlog item sizes. Furthermore, the functional groups require varying amount of time to process each story or requirement which will further constrain the flow of work. (Larman & Vodde 2008.)

In Scrum, iterations are called sprints, and their length was specified to 30 consecutive calendar days (Schwaber & Beedle 2001). Schwaber (2004) further defines the rationale of selecting the length of the sprint as the balance between sales department desiring instant response to changes, and the need of the developers to be able to concentrate on their work without interference. Furthermore Schwaber considers 30 days it as the maximum time that the team can work without resorting to writing extra artifacts and documentation to support its processes. Larman & Vodde (2008) suggest the length of 2 to 4 weeks for

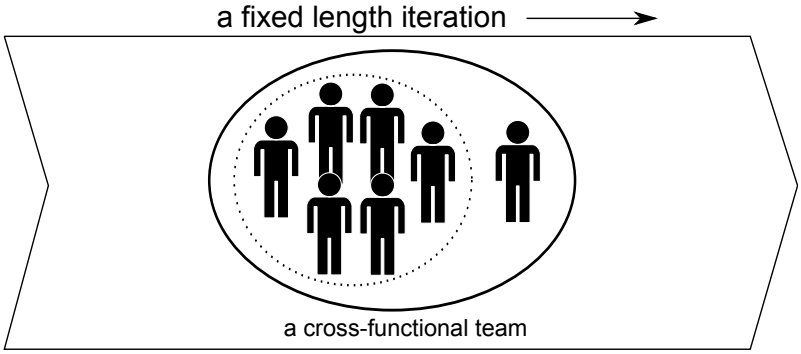

Fig. 4. The iteration with cross-functional self-organizing team: in both Extreme Programming and Scrum during the iteration the cross-functional team behaves as an inseparable organizational unit with shared responsibility on the iteration goals. All members have end-to-end responsibility on the teamwork, each participating in all aspects of value creation. (Larman & Vodde 2008.)

Scrum. The sprint process is depicted in Fig. 5.

During a Scrum sprint, the Scrum team can seek outside advice. However, as the Scrum team is completely self-managing, outsiders must not interfere with the Scrum development team during a sprint unless first solicited by the Scrum development team itself. If the Scrum development team is not able to complete all the items it has committed to, it can negotiate with the product owner what items it should drop. Similarly, if the team feels it has enough slack to complete more backlog items, it can solicit the product owner to assign additional items to the sprint. (Schwaber & Beedle 2001.) Extreme Programming (Beck 1999) is more allowing towards interference; the customer can ask for new stories to be added to the ongoing iteration as he pleases, but in that case, equivalent amount of lower priority work is removed from the iteration plan.

In large-scale software development, there often are multiple development teams working on the same product concurrently. According to Larman & Vodde (2008), in a large scale enterprise setting a Scrum eam would be working on the same product along with other teams, would have work that is interdependent on the work of other teams and would have a set of working agreements. Yet such team would still be responsible for managing relations to the relevant stakeholders by itself and has distributed leadership among team members.

3.1.3 Sprint goal

In Scrum, a *sprint goal* is a written objective on what should be accomplished within a sprint. In the beginning of Scrum sprint, after the team the team has chosen a set of product backlog items to work on, the team will also formulate the sprint goal statement based on the set of selected product backlog items. The team is allowed to make adjustments to the sprint plan, as long as it fulfills the sprint goal statement. (Schwaber & Beedle 2001). In Extreme Programming iteration, the team is tasked to implement a set of high priority *stories.* The equivalent of sprint goal statement is not used in Extreme Programming; the team should consult the onsite customer whenever they want to change the iteration plan. (Beck 1999.)

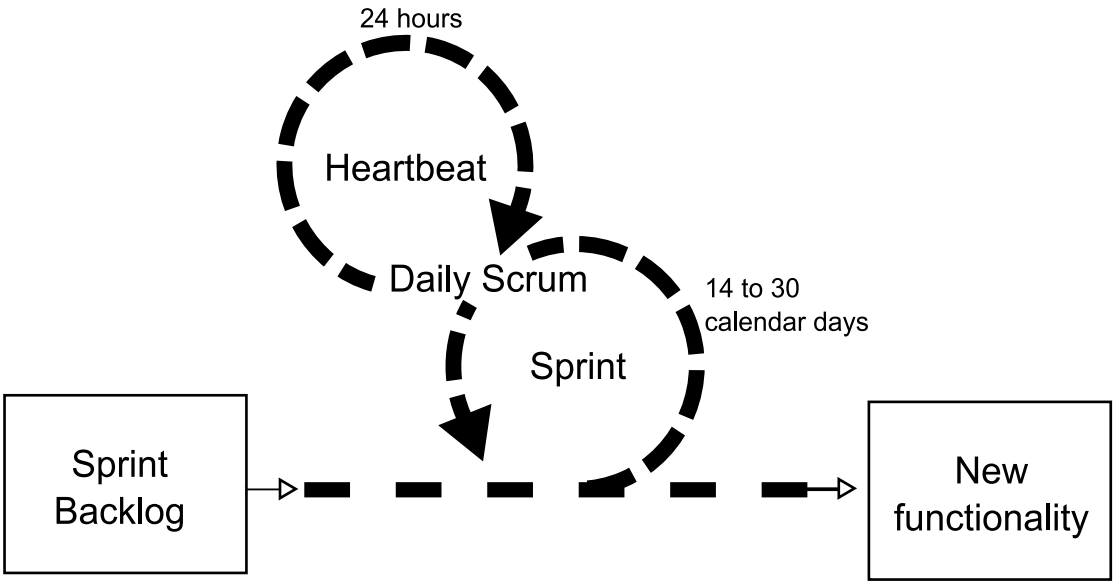

Fig. 5. Scrum sprint process overview (Schwaber & Beedle 2001, Leffingwell 2007)

3.1.4 Story

The essence of software development is the implementation of functionality that satisfies the needs of the customers. According to Beck (1999), in Extreme Programming the customer needs are described as customer-centered user stories. These user stories originate from the customer role in XP. These stories may be described either informally in the format "*The (role) can do (something)*", or more formally in the format proposed by Cohn (2004): "*As a (role), I want (something), so that (benefit)*". Cohn argues that using user stories helps focusing on delivering value to the customer, and the formal format makes the business value of a requirement even more explicit. In Extreme Programming the stories are often written only on index cards or post-it notes. (Beck 1999.)

In original Scrum the customer needs were called *product backlog items.* They are not customer-centric user stories per se, but more broadly proposed changes — features, functions, technologies, enhancements and bug fixes — that are considered by any stakeholder to be a necessary or nice addition to the product(Schwaber & Beedle 2001, Schwaber 2004). However, the user story concept is also applicable to product backlog items (Larman & Vodde 2008).

The stories (product backlog items) might be split and combined as new knowledge emerges, and their effort estimates be updated. Before they are implemented, they are usually split so that several of them can be implemented by the team within a single iteration (Schwaber & Beedle 2001, Beck 1999). Larman & Vodde (2008) propose that they be split so that the resulting stories take no more than a quarter of an iteration to implement from a team. In the context of this thesis, the term story is used exchangeably with the term product backlog item.

3.1.5 Task

The stories are not used as such for managing and tracking the progress daily work in either Scrum or XP. Instead a set of concrete engineering tasks is derived from the set of

selected stories, product backlog items or the sprint goal. Each engineering task has one or several assigned people, who are working on the task and responsible for completing it. The dependencies of tasks are not managed explicitly; instead it is assumed that the self-managing team can resolve these issues themselves. The effort required complete each task is estimated, and these effort estimations are the only means of tracking the iteration status. (Schwaber & Beedle 2001, Beck 1999.) Beck & Fowler (2000) identify three types of tasks in Extreme Programming: those that belong to a single story, those that contribute to many stories at once, and technical tasks that do not belong to any story. In Scrum the tasks are derived from the sprint goal, using the product backlog as a roadmap (Schwaber & Beedle 2001).

In Scrum each task should be sized so that it takes approximately 4–16 person-hours to complete, and larger tasks can be used as placeholders for tasks that have not yet been refined to the appropriate size. Also, as unanticipated work related to backlog items is discovered, a new task is created for it and appropriate estimations done. (Schwaber & Beedle 2001.) Beck (1999) proposes that the size of tasks in Extreme Programming should be 1–3 ideal developer days. According to Jeffries *et al.* (2000) the lower limit could be only 2 to 4 hours in Extreme Programming.

The strategies used to assign responsibility of tasks are reviewed in Section 3.2.2, the practices of selecting the next task to work on are reviewed in Section 3.2.3, the task estimates and measurements are reviewed in Section 3.2.5.

The unit for task effort is usually hours, but might be other time related units (Schwaber & Beedle 2001, Beck 1999). As the task is being worked on by a developer in Scrum, he is responsible for re-estimating the effort needed to complete the task. If the required effort was originally underestimated, the estimate might increase. (Schwaber & Beedle 2001.)

3.1.6 Backlogs

Backlog is a Scrum-specific term. Backlogs consist of future work to be done on the product. The product and release level backlogs consist of stories, whereas the iteration level plan consists of engineering tasks. (Schwaber & Beedle 2001.)

Product backlog

The Scrum product backlog is a list of requirements that are being considered to be implemented in a product. The product backlog is always changing as new requirements are discovered and business priorities change. The product owner is the sole person responsible for ensuring that the product backlog is always up-to-date and prioritized. The Scrum model as defined by Schwaber & Beedle (2001) has a single product backlog that sets the priorities for development teams, and the development teams are not allowed to work on any different set of priorities. The business decisions made by the product owner are visible in the product backlog as the prioritization of backlog items. (Schwaber & Beedle 2001.) In Extreme Programming the equivalent of product backlog would be the set of all story cards that have not yet been assigned to any iteration (Beck 1999).

Task description	Originator	Responsible	Status (Not started/ Started/ Completed)	Hours of work remaining until completion 1	2	3	4	5	6	7	8	9
Write release notes		Pete	Completed	8	8	8	6	0	0	0	0	
Package alpha 3	Pete	Susan	Not started	16	16	16	16	16	16	16	16	
Renew row layout		Jim	Not started	4	4	4	4	4	4	4	4	
Add project backlog tab		Jim	Completed	16	16	10	4	0	0	0	0	
Handle release stories without children		Tom	Not started	8	8	8	8	8	8	8	8	
Implement showing stories on the list		Kate	In progress	16	16	16	16	16	16	16	8	
Optimize the rendering in reloads	Pete	Richard/Pete	Not started	32	32	32	32	32	23	32	32	
Remove "reload notifications"		Kate	Completed					4	4	4	0	

Fig. 6. An example of displaying and managing the iteration plan: an example of sprint backlog from Schwaber (2004).

Release backlog

In Scrum, the release backlog is a subset of product backlog that is selected by the product owner to be completed before the next release to the customer. A release usually consists of several sprints. As sprints are completed the product owner may make adjustments to the remaining release backlog. These adjustments are reprioritizing backlog items, adding new backlog items targeted to be completed in a release (upscoping), or removing backlog items from a release (downscoping). (Schwaber & Beedle 2001.) Beck (1999) specifies that the contents of a release are the stories the customer and the team have chosen together to be included in the next release; this set is thus the equivalent of release backlog in Scrum.

Iteration plan, or sprint backlog

Iteration plan (Beck 1999), or sprint backlog in Scrum (Schwaber & Beedle 2001), consists of all tasks the team has figured it has to do within the iteration to meet the iteration goal. The relations between iteration level concepts in Scrum are presented in Fig. 8, and in Extreme Programming in Fig. 9. The difference is that in Extreme Programming most tasks contribute explicitly towards individual stories, but some — for example maintenance tasks — do not (Beck & Fowler 2000). In Scrum sprint backlog, all tasks are directly derived from the sprint goal statement (Schwaber & Beedle 2001).

3.1.7 Impediment

A concept present in Scrum, Schwaber & Beedle (2001) define the term impediment as anything that prohibits the team or a member from working with maximum efficiency on the sprint goals. Impediments can be technical, as in network server being down, or slow; managerial, as in being asked by the management to work on tasks that the team member or the team did not commit to work on in the sprint; or engineering related, as in uncertainty about how to continue working in the sprint. Schwaber & Beedle (2001) also specify that Scrum master is responsible for tracking and removing the impediments.

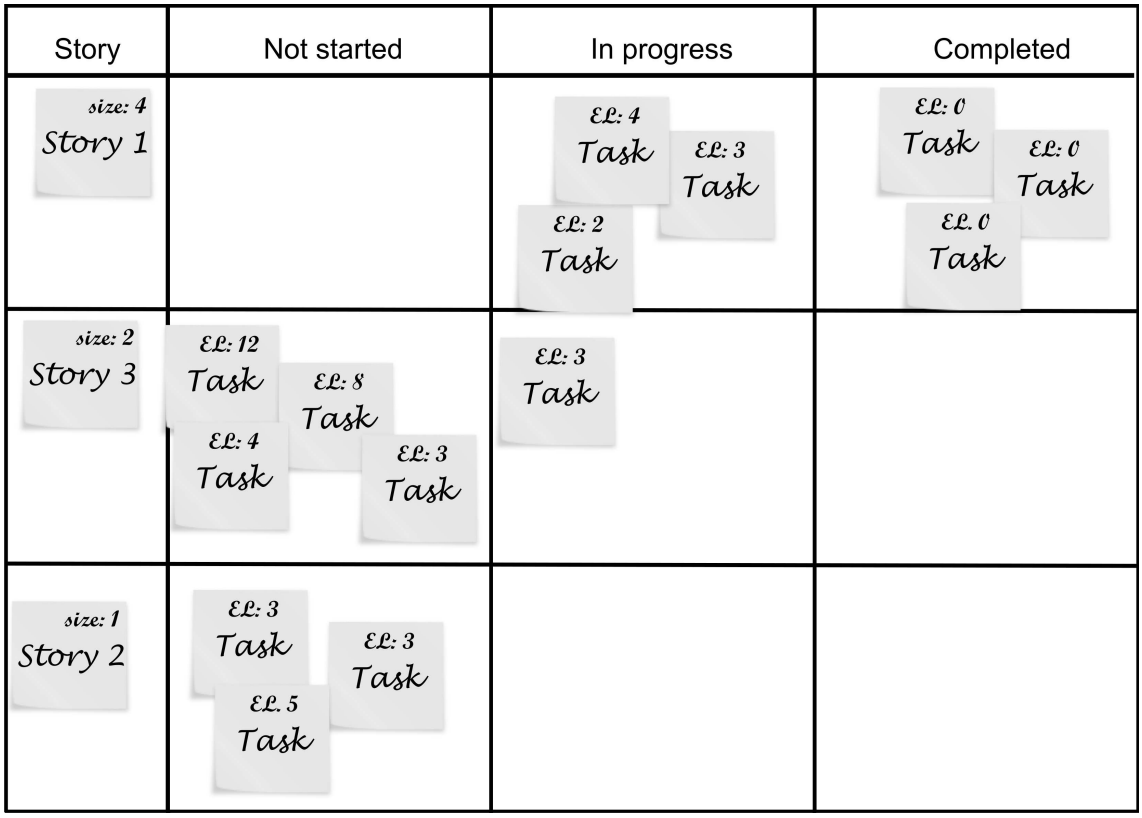

Fig. 7. An iteration plan represented on a task board (Larman & Vodde 2008). The task board shows that the team is approximately 50 % done.

3.2 Practices of daily work management in practitioner guidebooks

This section represents the 10 daily work management practices identified in the books. The Extreme Programming software engineering practices are excluded by definition, even though they would contribute to the sustainable velocity of the team.

3.2.1 Visual management of work

In Scrum, the product backlog is visible to everyone in the development organization; however only product owner has authority to decide on the prioritization of product backlog. Likewise, the sprint backlog is highly visible artifact that only the development team itself can modify. The backlog item effort estimates in product backlog are used to draw release burn-up graphs; likewise the task effort estimates in the sprint backlog are used to draw the sprint burn-down graph (see Figure 10 on page 35), which is the central tracking and analysis tool of the team performance during a sprint. (Schwaber & Beedle 2001, Schwaber 2004.)

In Scrum the iteration plan, or sprint backlog, is stored as a spreadsheet file on a networked server that is accessible to every team member. The impediment list is proposed to be tracked on a whiteboard in the team room. (Schwaber & Beedle 2001, Schwaber 2004.) According to Beck (1999) in Extreme Programming the stories and tasks are written on special purpose note cards. As the developers sign up for a set of tasks, they personally hold the stack of tasks to themselves, or they can be kept on a wall.

Other sources suggest various tools for differing reasons. Larman & Vodde (2008)

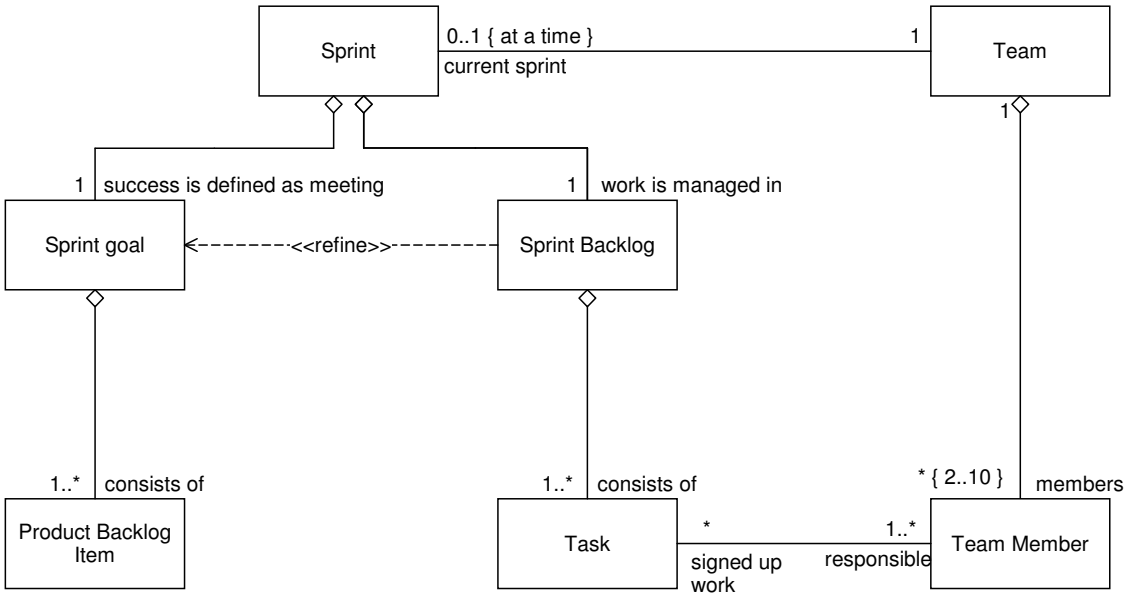

Fig. 8. Conceptual model of Scrum sprint level concepts. A UML class diagram.

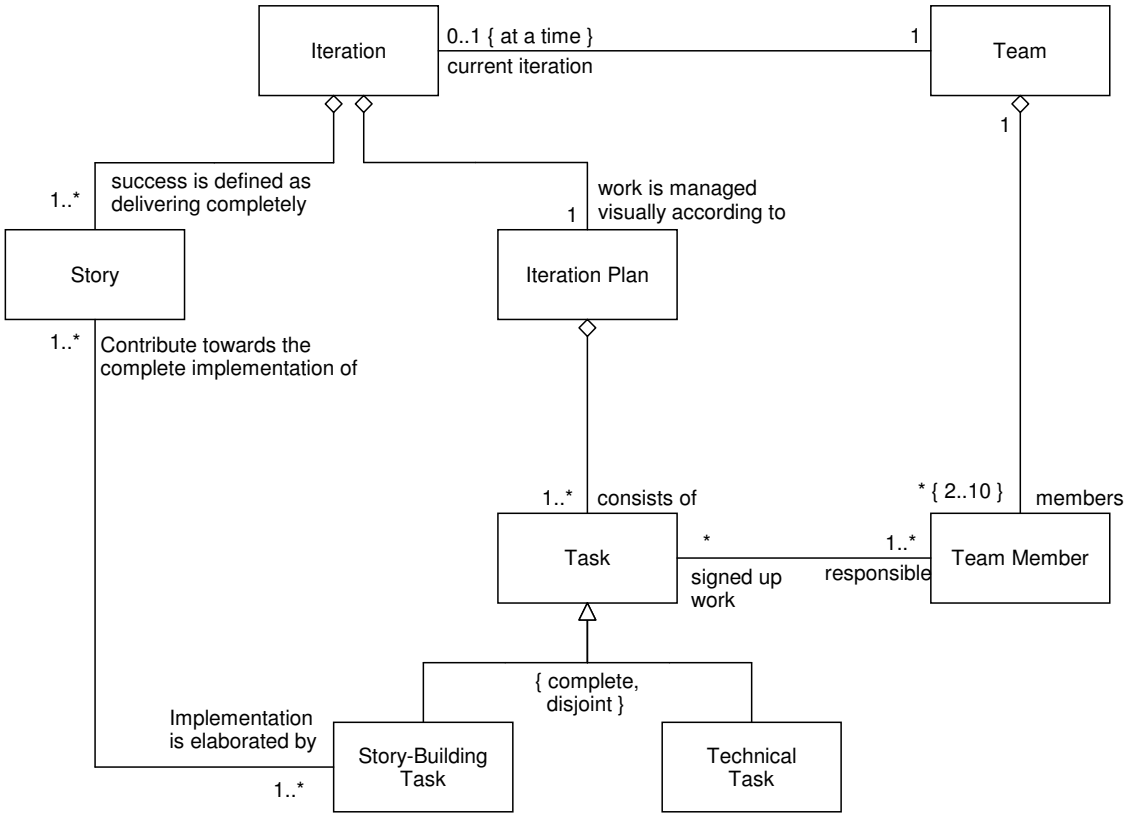

Fig. 9. Conceptual model of between Programming iteration level concepts. A UML class diagram.

deviate by preferring visual management with physical tokens such as task board with story and task cards over computer system. Their stance is that storing the data in a computer system defeats the purpose of visual management, which is the handling of work items as tangible objects — index cards and post-it notes. According to Larman & Vodde (2008) the iteration plan should then be tracked on a visible taskboard with the note cards moving from one column to another as they are started and completed. Larman & Vodde (2008) argue that such practice will make the existence of queues and spurious multitasking visible to the team.

Cohn (2005) argues that note cards are especially better than spreadsheets for collocated teams, because they allow the entire team to participate in the iteration planning process, whereas with a spreadsheet, only the person with keyboard may enter tasks. His opinion is that allowing everyone to participate in the process simultaneously leads to more democratic process and likely better results.

According to Leffingwell (2007) a spreadsheet for backlog status tracking together with a wiki, index cards or whiteboard could be the ideal management to teams of less than 10 collocated persons. However, as soon as the organization consists of multiple teams — especially if the teams are geographically distributed — or teams of teams, the intra-team and cross-team coordination poses challenges. Leffingwell further states that to support such distributed scenarios the management environment should be not only viewable by everyone but also updateable by everyone to keep track of the status of features, log impediments and refresh effort estimates.

3.2.2 Assigning responsible members

All book sources suggest that the tasks should be preferably allocated to team members based on their own willingness and initiative, and not by a manager. Auer & Miller (2001) identify two basic strategies of assigning tasks to team members:

- One at a time: a team member picks the one task that he wants to work on, and only after completing it chooses another one.
- Fill your bag: a team member volunteers for an iteration's worth of tasks during the iteration planning.

Both strategies have advantages and disadvantages. In the one at a time strategy a single developer does not become the bottleneck, as dependent tasks are not yet allocated, the disadvantage being that the effort estimates might not be as accurate, and thus it might be more difficult to ensure that all committed-to tasks fit in the iteration. The advantage of the fill your bag strategy is that every task now has ownership and thus possibly more accurate estimates. Early allocation of tasks also facilitates communication, because it will be known early to all relevant stakeholders who to contact if some details regarding the implementation of a task has changed. (Auer & Miller 2001.)

Beck (1999) discusses both strategies and proposes that the fill your bag strategy be used in XP, because in his opinion it is important that even the early estimate is done by the original assignee to be able to balance the load. Later, Beck & Andres (2004) suggests that the one at a time strategy could be also useful: the developer would select the task card on top of the shared stack of task cards would be advantageous by promoting transfer of learning; if the developer is not an expert in the work required for the task, he can pair with a specialist. Larman & Vodde (2008) are also proponents of "one at a time".

Cohn (2005) argues that one or two tasks at a time is the preferred method of signing up for tasks in agile software development, as early assignment of responsibility "may

work against the unified commitment of the team". Cohn (2005) also claims individual estimation of tasks would do little to improve the estimates, arguing that collective estimation of tasks produces good enough estimates. Cohn (2005) further adds that if the tasks are allocated to individuals, they should nevertheless be estimated as a team, and still considered shared.

Neither Schwaber & Beedle (2001), Schwaber (2004) nor Leffingwell (2007) discuss task allocation strategies. Instead the Fill your bag strategy is used throughout these sources without any explicit justification. In their views each task in the original Sprint backlog should be assigned to one or several responsible team members in the latter half of sprint planning meeting. However, the development team and individual team members themselves are to decide who should be assigned which individual task.

Larman & Vodde (2008) claim that the selection task allocation might have a profound impact on organizational transfer of learning. They report of a case, wherein the team achieved permanent 20 % increase in feature velocity by utilizing a task selection method called "least qualified implementer", combined with "promiscuous pairing". The least qualified implementer method means that a developer would pick the task that he or she would consider to be least qualified to implement of all team members. In promiscuous pairing, the programming pairs are not fixed for the entire duration of the task, but the expert members change pairs often, in this case every 90 minutes.

The choice of task allocation strategy additionally affects the practices selection of the next task to work on (3.2.3), estimation of the task effort (3.2.5) and load balancing (3.2.7).

3.2.3 Selection of next task

When the iteration starts, whenever a previous task is finished or blocked, the team member needs to choose the next task he works on. The books suggest various approaches of choosing the next task. According to Auer & Miller (2001), if the "fill your bag" strategy is used, the team member would prefer completing first those tasks he has signed up for. If the "one at a time" strategy is used, or the team member has completed all tasks he has signed up for, all tasks in the iteration backlog have not yet been started can be considered. If the team practices pair-programming (Beck 1999), the team member obviously also needs to find a pair for the task. Jeffries *et al.* (2000) note that the main goal of each Extreme Programming iteration is to complete stories. Thus, if only a fraction of previously committed to work can be successfully implemented in an iteration, it would be better to fully implement some stories and not even start the lowest priority ones.

Beck (1999) suggests that the tasks should go through Extreme Programming planning game, and thus the task cards have some kind of priority, however the completion order of tasks is not discussed further. Also, some tasks are dependent on others, but the exact order of execution is not to be planned beforehand and instead left to the team to decide during an iteration. Beck & Andres (2004) propose that when "one at a time" strategy is employed, the task cards could be stored in a stack, which is sorted by priority; the team member needing more work would choose the topmost card and implement it with a pair.

Larman & Vodde (2008) advocate the previously mentioned least-qualified implementer method of task selection, that is, of all possible tasks that the team member could work on next, he should choose the one on which he has least expertise among the team members. Schwaber & Beedle (2001), Schwaber (2004) and Leffingwell (2007) do not discuss specifically discuss the execution order of tasks, except that it is left for the team and the team members to decide. However, the product backlog is frozen during an iteration

and the backlog items are prioritized, thus the corresponding sprint backlog items can be considered prioritized too.

3.2.4 Lightweight measurements

According to Schwaber & Beedle (2001), in Scrum the only general metric gathered during the sprint are the effort left estimates of sprint backlog tasks. Particularly, the time spent on tasks is not to be recorded at all, as it is seen as counterproductive (Schwaber & Beedle 2001). Schwaber (2004) also argues that explicitly comparing original effort estimates to actualized number of hours spent on tasks in order to make the estimates better might lead to lower quality as the developers sense the pressure to meet the estimates they have done.

Beck (1999) proposes that in Extreme Programming a special team role, Tracker, gathers the metrics on team performance. The gathered data includes the actual effort spent on each task being worked on and ideal effort left on each task. The Tracker gathers this data by explicitly discussing with each developer, without disturbing the work of developers unnecessarily. The Tracker also provides feedback to the developers by comparing the actual effort spent to tasks with the original estimate of tasks.

However Auer & Miller (2001) suggest that the effort estimates should be done in Ideal Time: "Ideal Time is time without interruption where you can concentrate on your work and you feel fully productive". Ideal time may be measured not only in ideal working hours or days, but also in "task points" or "Gummi Bears". Auer & Miller (2001) also refer to Francesco Cirillo using a 30-minute tomato-shaped kitchen timer to track the passing of ideal working time, and measuring the task efforts in "tomatoes" ("pomodori" in Italian); Cirillo's Pomodoro Technique is discussed in Section 3.2.10.

Beck (1999) proposes using *big visible charts* to aid the team work in Extreme Programming. The charts display the changes of selected performance metrics over time; the tracker role is concerned with gathering the required data and responsible for update these charts. Unlike burn-down chart in Scrum, these charts are temporary; only those metrics that are currently of interest should be recorded. Larman & Vodde (2008) warn against using management measurements on lower process cycles, because they might lead to the developers working towards the metric, not overall performance. Larman & Vodde (2008) thus argue that any such measurements must be managed by the team itself, not come from an outside source.

3.2.5 Status tracking

The realized status of an iteration is constantly or periodically compared to the iteration plan (Schwaber & Beedle 2001, Beck 1999). In Scrum the status tracking of a sprint is achieved by constantly keeping the effort estimates in the sprint backlog up-to-date. A developer working on a task is responsible for estimating the remaining effort needed to complete it at least every day. The unit of effort left is person working hour. The effort estimates in the sprint backlog are used to create the sprint burn-down graph (See example in Fig. 10). In the burn-down, time advances along the x-axis; the y-axis represents the sum of remaining estimated effort to complete all tasks of the team at the given time. It is the responsibility of the Scrum master to ensure that sprint burn-down graph is updated daily. (Schwaber & Beedle 2001.)

The Scrum burn-down can be used to assess the viability of the sprint — if an extrapolation of effort left does not seem to intersect the x-axis before the end of sprint, a

reduction in the sprint scope might be warranted (see Load balancing, 3.2.7). Changes in sprint burn-down graph patterns compared to previous sprints may also signify problems that might need to be addressed. The amount of work remaining can increase whenever new tasks are discovered and added to sprint backlog, or previously underestimated tasks have their efforts re-estimated; likewise the remaining effort estimate decreases as progress is made, but also if the scope of an iteration is changed or overestimated tasks are re-estimated. (Schwaber & Beedle 2001.)

Originally the Tracker role of Extreme Programming Tracker was responsible for periodically asking the developers how much ideal development time they will require to complete their set of tasks (Beck 1999). Auer & Miller (2001) suggest that the tracking of the project status in Extreme Programming should be automated as much as possible. Most of the measurements used in tracking the status of an iteration or project, such as number of acceptance tests passed, could be gathered automatically. Then only certain data would need to be input manually, most notably the person who took the responsibility of each task, his original effort estimate and the actual effort spent on the task. The monitoring responsibilities of Tracker role can then be distributed among the developers.

3.2.6 Status update meetings

Both Scrum and Extreme Programming prescribe daily status update meetings that are a tool for the development team to synchronize on the status of other people. The sources are unanimous on the structure of the daily status update meeting: each team member in succession, addressing the team, answers to three questions:

- what the member has done during the previous heartbeat
- what the member is planning to work on during the next heartbeat
- what problems he is experiencing or foreseeing that impede optimal performance

Anything else does not belong to the status update meeting, and should be handled in separate follow-up meetings.

Schwaber & Beedle (2001) call the status update meeting *the Daily Scrum*. The Daily Scrum should be time-boxed to 15 minutes and every team member should preferably participate personally, either by being present, or if telecommuting, via call-in. The Daily Scrum should convene at the exactly same time on every working day in the same place. The Scrum master chairs the meeting and enforces the Scrum rules. In the meeting only the committed-in team members are allowed to talk, but in accordance with openness the meeting can be attended by anyone in the organization. Schwaber & Beedle (2001) suggest that those team members assigned to multiple teams should report and talk only of that work that belongs to this Sprint and this team.

Schwaber & Beedle (2001) propose that senior management should especially attend the Daily Scrum to monitor the progress of the team and to review the chances of success within the sprint. Larman & Vodde (2008) compare this with the Toyota Production System principle *Genchi Genbutsu — go and see for yourself (Liker 2004).*

Beck & Fowler (2000) note that many programmers view meetings as waste of time; against this they state that a status update meetings have proven invaluable in their experiences of Extreme Programming in that the meetings also facilitate team communication, provided that the length of meetings stays within the time box.

The daily stand-up meetings are only supposed to facilitate in communicating problems, not solving them. Any discussion that does not belong to the status update meeting should be handled in follow-up meetings that are scheduled after the Daily Scrum has

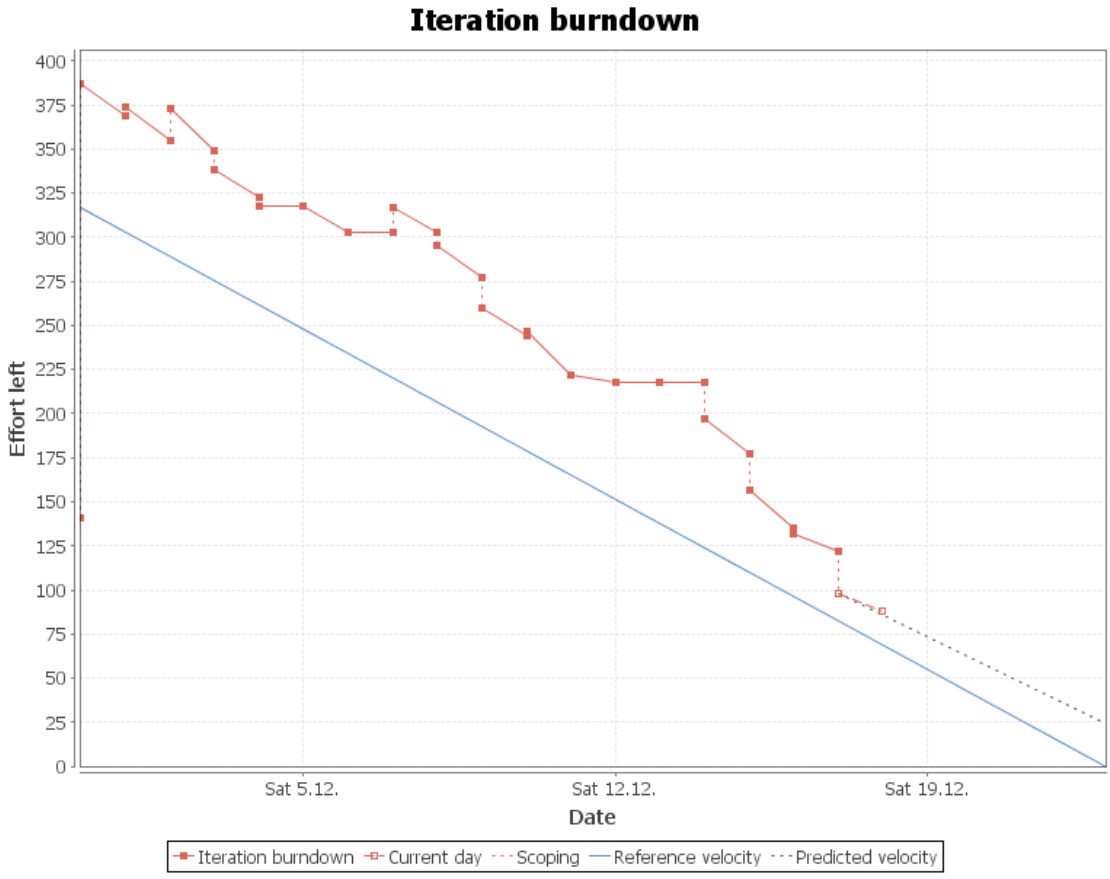

Fig. 10. An example of iteration burn-down graph from Agilefant. The time unit used is working hours.

adjourned; any team member may request a follow-up meeting to be held after the status update meeting. In accordance with openness, anyone may attend the meeting. Unlike in status update meetings, the participation in follow-up meetings is not mandatory for team members. (Schwaber & Beedle 2001, Beck & Fowler 2000.)

3.2.7 Measuring and balancing workload

Load balancing is the activity of changing the scope of commitment in the middle of an iteration. As iterations are always fixed in length, and the difficulty of work can never be accurately estimated beforehand, a team member or the entire team may have overcommitted to amount of work that cannot be successfully completed within the boundaries of the iteration time box, and the load must be rebalanced. (Beck & Fowler 2000, Schwaber & Beedle 2001.) There are two kinds of overload requiring rebalancing intervention: internal and external overload. In internal overload the remaining work that the team members have been assigned responsible for, is unevenly distributed (see Fig. 12). In external overload the team as a whole has underestimated the effort required to implement the entire sprint backlog (Fig. 11). (Beck & Fowler 2000, Schwaber & Beedle 2001.)

The remaining work of the entire team is always visible from the burn-down diagram (Schwaber & Beedle 2001), or is assessed by the Extreme Programming tracker role, as discussed in Section 3.2.5 (Beck 1999). There are two possible ways the team can balance

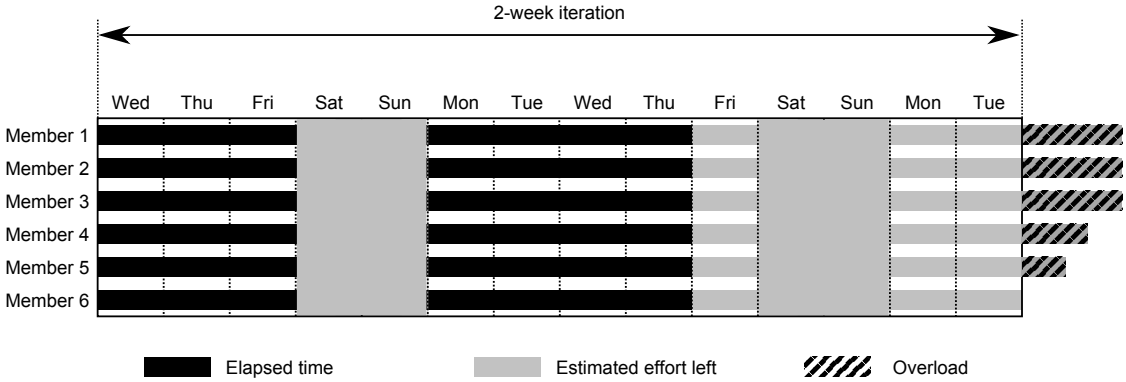

Fig. 11. External overload. Three working days and a weekend is remaining, and the effort estimates indicate the team would require two additional working days to complete the iteration plan in the present scope.

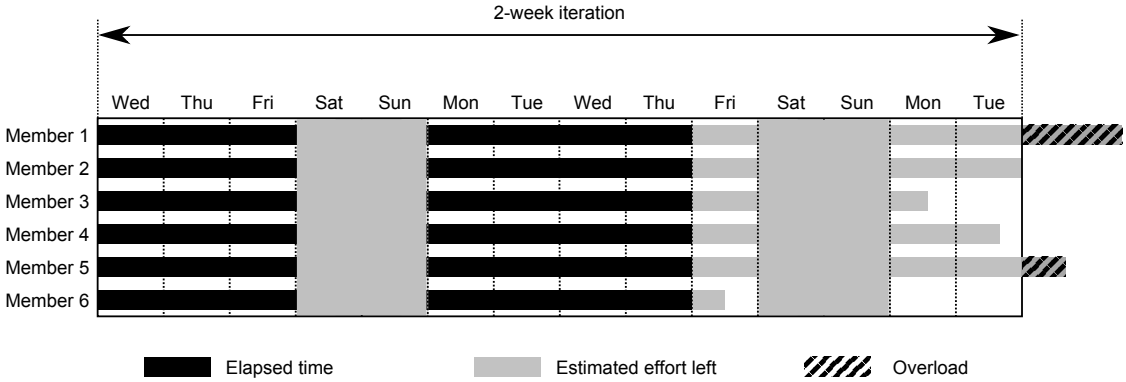

Fig. 12. Internal overload without external overload. Three working days and a weekend are remaining. Team members 1 and 5 have committed to too much work, whereas team members 2, 3, 4 and 6 have slack. The team as a whole needs less working hours to complete the iteration than there are available. Thus the team can deliver the increment as planned, if others take tasks from the overloaded members.

its external load. Firstly, it may achieve it by reducing the scope of implementation of one or several product backlog items or stories (Schwaber & Beedle 2001). Secondly, by negotiating with the customer or product owner if one or several stories or product backlog items could be postponed until a future iteration. (Beck & Fowler 2000, Schwaber & Beedle 2001.)

The workload of an individual team member consists of the tasks he or she has signed up to be responsible for. Auer & Miller (2001) suggest that one advantage of the one at a time task allocation is that it balances the internal workload implicitly. Auer & Miller (2001) also note that if Fill your bag strategy is employed, the unevenness in workloads may be non-negligible. In such case during the last days of an iteration those who finished their tasks early will then aid those who have still work remaining (Beck & Fowler 2000). If the team consists of specialists with highly varied skills and knowledge, it might not be possible for the other members to aid the overloaded member. Such specialists can more easily become bottlenecks within the team and thus decrease the productivity of the team, unless they share their knowledge to the other members of the team. (Larman & Vodde 2008.)

Neither Extreme Programming nor Scrum includes the notion of tracking individual workload automatically. However, the data of remaining effort for each developer is available in the Scrum sprint backlog (Schwaber & Beedle 2001), or possibly in an automatic tracking system (Auer & Miller 2001). If the automatic tracker system is not used, it is the tracker role that periodically asks from every team member how much time they will need to complete their task set (Beck 1999). Finally, the team member can actively request help from other team members, or report the infeasibility of completing his tasks as an impediment in a stand-up meeting (Auer & Miller 2001, Schwaber & Beedle 2001).

3.2.8 Impediment tracking and handling

Schwaber & Beedle (2001) suggest that if during a stand-up meeting team member reports an obstacle impeding optimal work performance, the Scrum master is responsible for recording and removing that obstacle. The impediments are to be recorded on the team white board. Removing the impediments is the top-priority duty of the Scrum master. Larman & Vodde (2008) recommend that deeper causes of impediments should be thoroughly studied in retrospective meetings after the sprint has ended.

Leffingwell (2007) propose that a multi-team organization should have an executive manager role called Scrum sponsor, who would be responsible for managing the organizational impediment backlog, in essence working as its product owner. The organizational impediments that are beyond the control of individual teams and their Scrum masters are added to this impediment backlog. The Scrum masters in teams would consider this prioritized list of impediments as their own product backlog, working on the backlog items in priority order, removing these impediments one by one.

The Extreme Programming sources do not propose any particular practices for recording and removing those impediments that are reported in daily stand-up meetings.

3.2.9 Dedicating team members

Team members are dedicated to a team during an iteration if they have committed to contribute to only to the assignments of that team during the iteration. The contrary

practice is called timesharing; a timeshared person is a committed to contribute productive work in multiple teams during a single iteration. (Larman & Vodde 2008)

Schwaber & Beedle (2001) suggest that timesharing is a standard practice in Scrum, noting that not all expertise is available in the organization that every Scrum team can get a specialist in certain fields. In such case the person can be allocated partially to several projects. The person needs then personally track his commitments so as not to exceed the partial allocation. However in a later Scrum book, Schwaber (2004) discourages the use of timesharing and recommends that each team should have among dedicated team members all necessary skills to complete the sprint goal; Schwaber (2004) also recommends that if this proved to be impossible, the shared team member should commit only to his or her primary team, and serve only in an advisory role on the other. Larman & Vodde (2008) agree, citing Jensen (1996): "Part-time people equate to part-time commitment. Part-time commitment leads to team failure" and thus recommend maximizing the dedication of team members to a single team.

None of the studied Extreme Programming sources discuss the possibility of sharing team members with other teams. However, it is the understanding of the author that these sources implicitly assume that all team members are dedicated to their teams.

3.2.10 Maintaining focus and establishing cadence

The two complementary self-management practices for achieving sustainable pace are preserving the focus and establishing overall cadence (Cohn 2005). According to Cohn (2005), maintaining focus at the task at hand is essential, because task-switching always incurs a penalty. Cohn (2005) also argues that rhythm is essential to achieving sustainable pace.

Larman & Vodde (2008) and Leffingwell (2007) both note that the iteration time box provides a natural rhythm that paces the teamwork; they both argue that the iteration length should be fixed throughout the organization. Larman & Vodde (2008) and Leffingwell (2007) both also state that if any non-continuous events, such as design meetings, are to be held in every iteration, they should be always held at the exact intervals to promote cadence. Leffingwell (2007) discusses an artifact called iteration cadence calendar, that can be used to schedule periodic events in iterations.

To retain focus on the current task, Cohn (2005) recommends the one-at-a-time task allocation for the team, as then the developer does not have a large set of tasks to choose from. In addition, Cohn (2005) suggests the use of the Pomodoro technique (Cirillo 2006). In Pomodoro technique, the team works in 30-minute increments, focusing on work for 25 minutes, and having a 5-minute forced pause before next increment. This effectively establishes a mini-cadence for the team.(Cohn 2005.) Cirillo (2006) claims that using the Pomodoro technique, one could become better at estimating tasks, as the sense of passing time is made explicit. Furthermore, the technique would aid in measuring the external interruptions and interference, and mitigating their negative effect. (Cirillo 2006.)

3.3 Summary of team tools according to the book review

The tools from book sources are listed in Table 3. There are three different tools used to manage the iteration plan. The plan might be managed as tasks written on cards or post-it notes that are represented visually on a task board (for example Cohn (2005)). The plan might also be stored in a spreadsheet file, as was done originally in Scrum (Schwaber &

Table 3. The tools proposed by the reviewed books.

Tool	Source								
	Beck (1999)	Beck & Fowler (2000)	Auer & Miller (2001)	Schwaber & Beedle (2001)	Schwaber (2004)	(Beck & Andres 2004)	Cohn (2005)	Leffingwell (2007)	Larman & Vodde (2008)
Cards, post-it notes	x	x	x	-	-	x	x	x	x
Taskboard	x	x	x	-	-	x	x	x	x
Spreadsheet files	-	-	-	x	x	-	x	x	x
Burndown graphs	-	-	-	x	x	-	x	x	x
Temporary measurement graphs	x	x	x	-	-	x	x	-	-
Specialized management software	-	x	-	-	-	-	-	x	-

Legend: *x = the source recommends the tool, - = the source does not recommend the tool or does not discuss it*

Beedle 2001); the rows in the spreadsheet are the engineering tasks. The third way would be to store them in a specialized management information system that would provide the advantages of tangible cards and easily updateable spreadsheet file, while minimizing their disadvantages (Leffingwell 2007).

While a taskboard or a spreadsheet file is useful to demonstrate the current status of the iteration, the overall trend of progress over the iteration is best represented on the burndown graph (Schwaber & Beedle 2001). If spreadsheet is used to store the iteration plan, the burndown can be embedded on it and updated automatically; with taskboard, someone must calculate the effort remaining to complete all the tasks and update the burndown graph with a pen (Cohn 2005). A specialized tool would be able to generate these graphs automatically (Leffingwell 2007). The Extreme Programming sources prescribe similar big visual charts for visualizing quality measurements. Unlike burn-downs, these charts are considerd temporary and should be removed from sight when they have served their purpose. (Beck 1999.)

3.4 Conclusions on the literature requirements

While some sources advocated specialized backlog management tools for supporting team work (Leffingwell 2007, Auer & Miller 2001), others explicitly recommend low-tech tools over high-tech. All sources agree, however, that the team should use the tools that it is comfortable with for managing the iteration. Thus to be a viable alternative to the low-tech tools, a management information system has to be at least as usable. Heikkilä (2008) has studied the requirements of a backlog management tool on the level of conceptual model, but this approach is not sufficient to maximize usability. An information system must also support all necessary interactions with the model.

Also, as Scrum and XP are empirically controlled, there is no standard process for managing daily work. Thus, a daily work management tool should either actively support,

or at least not make more difficult than with the standard tools — be it taskboard or spreadsheet — all management practices found in this literature review.

4 Analysis of the software tool before enhanced support for daily work

The conceptual model of Agilefant is represented in figure 13 as a UML class diagram. Agilefant corresponds more closely to the Extreme Programming conceptual model (figure 9) than the original Scrum model (figure 8). There is one notable addition compared to the conceptual models of these methods: hierarchical stories and their traceability. A story can have any number of children; these form a generalized unlimited hierarchy, of which the Leffingwell's (2008) Epic–Feature–Story hierarchy is a specialized subset. Only leaf stories — stories that do not have any children — can be assigned into iteration backlogs; both branching stories and leaf stories can be present in either product or project backlogs.

There are also several deviations from both Extreme Programming and Scrum. The most obvious difference is the lack of proper teams in Agilefant. In Agilefant the "team" is simply a list of developers that can be used as a shortcut when assigning people to iterations or projects. A developer may be a member to unlimited number of "teams" and there can be an unlimited number of members in a "team". However, only users can be assigned to iterations, projects, stories and tasks, not teams. This also means that a newly added team member does not inherit any of the responsibilities or assignments; each of these must be applied individually. Also, the iteration backlog is semantically part of the project backlog, and thus can only contain artifacts from one project or product backlog.

In Agilefant, the iteration backlog also directly contains the stories, or the product backlog items; as was noted in Section 3.1, the contents of an iteration plan are the tasks that the team has devised to build the shippable increment. Also, leaf stories must be conceptually moved from product or project backlog to an iteration backlog, before they can be split into tasks and implemented. However this is a subtle difference, as an iteration backlog is semantically a part of project backlog also, and project backlog is part of a product backlog.

The entities within every container — that is, stories in backlogs and tasks in stories or iteration backlogs — have only a total priority order among their own. This deviates from Scrum or Extreme Programming, where all iteration tasks can be prioritized in relation to each other, regardless of with which stories or product backlog items they are related. In Agilefant it is impossible to tell if a task not related to any story is more important than some task that belongs to a story; also, lower priority tasks in a high-priority stories might be less important than highest-priority tasks in a low-priority story — this kind of information cannot be made visible by the ordering of tasks in iteration backlog.

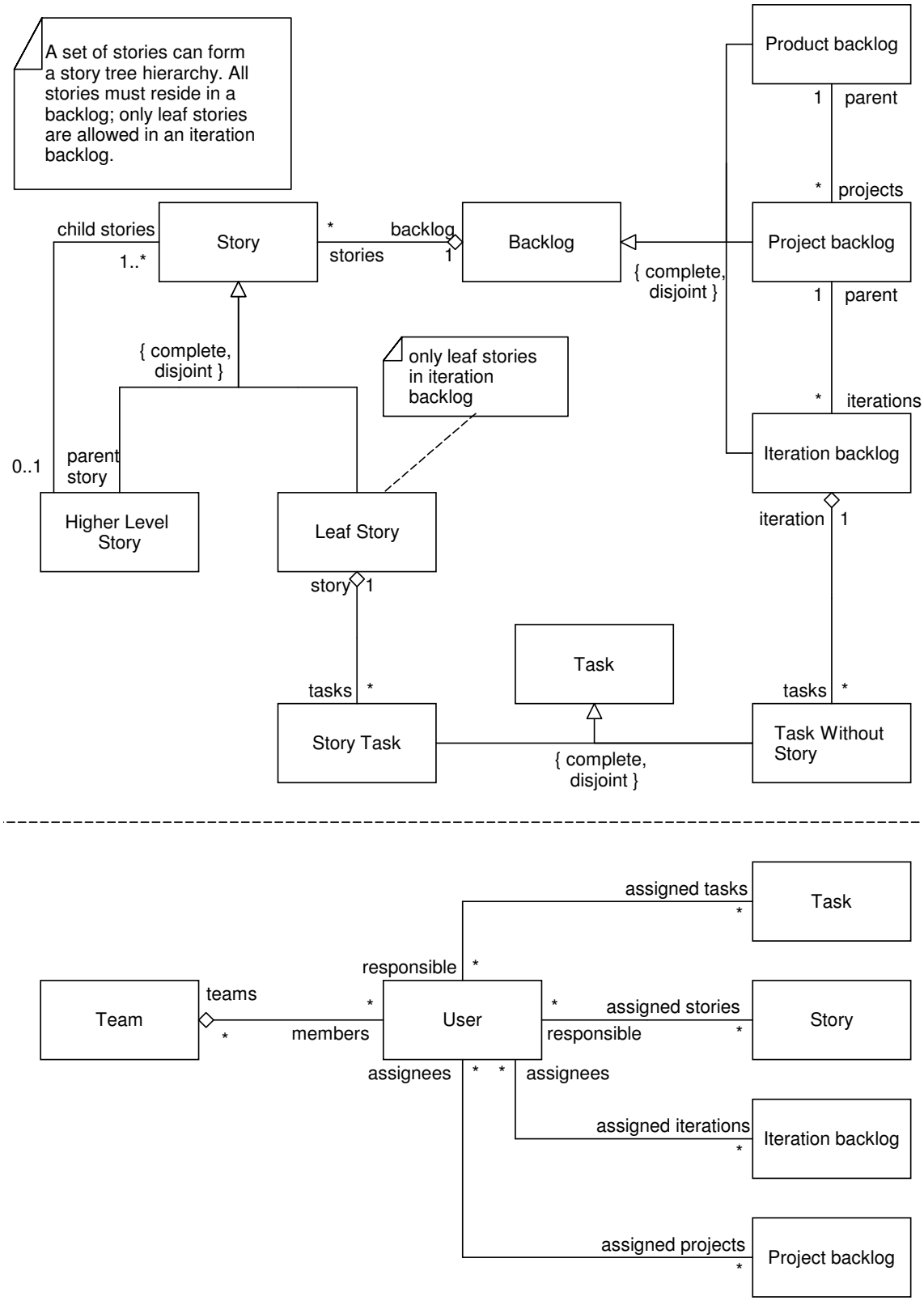

Fig. 13. High-level conceptual model of entities and their relations in Agilefant, a UML 2.0 class diagram. For clarity, the diagram is divided into two parts; the upper part depicts the backlogs, stories and tasks, and the lower the user entity and its relations. Some minor concepts, such as timesheet entries, that are of no interest to this thesis have been omitted.

4.1 Support for visual management of work

Agilefant represents the iteration backlog stories as rows in a table in the iteration backlog view; likewise, the tasks without story are represented in their own table. This approach resembles original Scrum, wherein the various backlogs were stored in spreadsheets (Schwaber & Beedle 2001, Schwaber 2004) (see also figure 6). Tasks within stories are not visible by default; the story contents can be expanded by clicking on a small arrow figure. Agilefant does not have the taskboard view (7) at all, even though the task board representation is preferred over the spreadsheet representation by the recent literature (Larman & Vodde 2008, Leffingwell 2007, Cohn 2005).

Each task and each story has a status field; there are 6 fixed, color-coded states: *not started*, *started*, *pending*, *blocked*, *ready* and *done*. Agilefant does not dictate any workflow or order for these states, except that when created, the tasks are not started - and that a task that is done, is marked as *done*. The remaining four can be considered different conditions in between these extremes: *started* can be used as in Scrum or Extreme Programming to signify that the task is being processed by responsible members. The *blocked* state can be used to signify that work on the task is blocked due to some internal or external condition, and that no progress will be made before the obstacle is removed. *Pending* tasks are being processed by external parties, such as orders for new tools, and require currently no attention from the team. Finally, a *ready* task is one that is completed but is not yet reported in a daily stand-up meeting, or has not yet been verified.

Agilefant automatically displays an iteration burndown view on the iteration backlog page; it is based on the history and the current sum of effort estimates among the tasks in iteration backlog. No other measurements are visualized, even though the data model could support them. Also, as a deviation from Scrum, there is no separate, visible impediment list.

4.2 Support for assigning responsible members

Agilefant supports both fill-your-bag and one-at-a-time task allocation strategies. Volunteering is supported, but not explicitly, as anyone can assign any task to others with the exactly same cumbersome procedure as they can assume the responsibility for themselves. Each task can have multiple responsible users. Furthermore, each story can have any number of responsible users. If a task within a story does not have any responsible developers, the task semantically inherits the assignees from its parent story.

4.3 Support for selection of next task

A new task to work on can be selected from the iteration backlog view. This is easy for the one-at-a-time task allocation; the developer can choose the first not started task from the highest-priority not-completed story and start working on it. If the fill-your-bag approach is used, the developer must specifically look for tasks he has signed up for; this can be tedious especially if the backlog is very large (10 team members and 30-day iterations would mean a backlog of hundreds of tasks if each task is sized between 4 and 16 hours). If the developer is not dedicated to a single project, or single iteration/sprint, he must go through all current iteration backlogs looking for the items he should be working on.

4.4 Support for measurements

Along with the necessary effort estimates, Agilefant also allows tracking spent effort, using the optional timesheets functionality. The timesheet entries must be inputted manually. Recording of any other iteration level measurements is not supported.

4.5 Support for status tracking

The status of an iteration can be tracked by various means in Agilefant, though only one is automated: on iteration plan view, an iteration burn-down graph is displayed, using the sum of effort estimates from tasks. A team member can update effort estimates on tasks from the iteration plan view. Again he must specifically locate the task among the various iteration backlogs to adjust the estimate. No other possible status indicators, such as the number of remaining stories or remaining story points, are calculated automatically.

4.6 Support for status update meetings

A mere report in an information system should never replace the status update meeting (Larman & Vodde 2008). Nevertheless, an information system should reflect the status of the team and the team members as accurately as possible, especially in situations where the team is not collocated (Leffingwell 2007). Thus the support for status update meetings means that the answers to the three questions are reflected on the information system.

The answer to the first question of status update meetings can be reflected in two ways in Agilefant. A team member can refer to data shown in timesheet functionality to answer what he has been doing during the previous heartbeat. Optionally, the team may decide to use the *ready* status on tasks to signify tasks that have not yet been reported in status update meetings; after the meeting they could be marked as *done*.

Answering the question on what the team member is planning to work on during the next heartbeat is supported if one-at-a-time task allocation scheme is chosen; then the team member can mark himself responsible for those tasks before the status update meeting convenes; he can show the started tasks in iteration backlog view. This is not applicable for the fill-your-bag strategy, as the team member has volunteered responsible for the task during the iteration planning. He might, however, signal his intentions to work on a task by marking the state of the next tasks as *started*, or *pending*.

The reflection on third question, or what is impeding the progress of a member or the entire team, is not explicitly supported. If the impediment is related to a single task, a developer might put a task to *blocked* state and provide the description of the impediment in the description field. However, as not all impediments are task-related, and as the Scrum master in Scrum might have to be able to prioritize impediments, this approach is insufficient. Two possible workarounds exist: the Scrum master might create another story in the iteration backlog containing the descriptions of all current impediments as tasks, or have a separate backlog for impediments. The first one implicitly assumes that the Scrum master is able to remove the impediments within the boundaries of the iteration, while the second one has poorer usability, as the impediment backlog is not visible alongside the ordinary iteration backlog, and thus not ordinarily visible to the team.

4.7 Support for load measuring and balancing

Agilefant lacks the support for calculating the external workload of the team, thus noticing external overcommitment is more difficult. However, if the team is assigned to only one iteration, external workload is visible in the iteration backlog view and in iteration burndown graph. External workload per team member is not calculated automatically.

Internal workload is displayed graphically on daily work view; it is based on the remaining effort estimates of committed to tasks. However, Agilefant places restrictions on the time units of the tasks; they must be in ideal man-hours and minutes. If the task has shared responsibility, the effort is divided equally among the responsible members. When using fill-your-bag task allocation, an undercommitted team member who is willing to help overcommitted team members must go through all team members by name in daily work to see who is the most overloaded one.

4.8 Support for impediment tracking and handling

As noted in Subsection 4.6, Agilefant does not have a specialized impediment backlog, and tracking of impediments can be achieved through a special “Impediments” story, or through creating a separate iteration backlog for impediments.

4.9 Support for stable teams and dedicated team members

Agilefant does not consider teams as first class entities. On the contrary, assigning a backlog, task, or story to a team is just a functional equivalent and shortcut for assigning that entity to each and every current team member. This has some consequences such as making the team feature velocity difficult, if not impossible, to calculate. Also, as noted in Subsection 4.7, the lack of real team concept leads to difficulties in measuring and balancing the external overload, especially if the team is working on several projects simultaneously.

4.10 Support for maintaining focus and establishing cadence.

Agilefant does not have any features that would support maintaining focus or establishing cadence.

4.11 Conclusion

Agilefant as a research-originated tool is a significant improvement over the prior research tools, such as Bryce or XPSWiki in terms of usability, due to client-side programming that allows concurrent edits and real-time view to the data. Despite this, as such it does not sufficiently support the daily work management practices in the book; thus, mature Scrum and Extreme Programming teams following the methods by the books would consider spreadsheets files and physical tools superior in daily work management due to their flexibility.

5 The results of daily work management workshop and prioritization

5.1 Results of the experience exchange workshop

The teams wrote a total of 69 user stories; 26 of these were written by the participants from product companies, 21 by the participants from project companies. The remaining 22 stories were written by the researchers. Some of the proposed stories had two pieces of functionality separated by the conjunction ”and”; these were split into two separate stories, resulting in a total of 83 stories. 13 of the resulting stories were duplicates of others, or out of the scope of daily work; 14 described features that were already implemented in the tool were also discarded (See Appendix IV for the list of discarded stories).

Not all remaining user stories were independent of each other — most of them described functionality that was partially shared with a set of stories. These sets of codependent stories were considered to form a set of completely independent features. In a thorough analysis, 26 features were identified. The names of these features and names of their constituent stories are listed in Appendix I. The full texts of created user stories and features are listed in Appendix III.

5.2 Results of the prioritization

11 representatives answered the questionnaire, representing 9 organizations. 7 of these organizations were users of Agilefant. The positions held by the interviewees and the company profiles are listed in Appendix V; individual prioritization results are available in Appendix VI on page 93. The combined results are in Table 4. The table is ordered by value, highest-valued feature on top. The value of a feature is shown as a fraction of the total value of all features; the cumulated value for a feature is the sum of the value of this story and all higher-ranked stories. Each interviewee had equal, $\frac{1}{11}$ weight in the final prioritization. Table 4 shows that the highest-valued feature is the team view, with 19.8 % of the value assigned by the stakeholders. The prioritization results also show that 14 highest-priority features provide over 80 % of estimated value to the stakeholders (Fig. 14). However, Fig. 15 shows that the number of votes cast to individual features by interviewees vary significantly. In addition, 25 of the 26 features were considered worthless by at least one interviewee. One interviewee had cast only 1 % of his available votes to the 26th feature, Team view (1), whereas another had cast 80 %. This shows that preference for this set of features is not uniform.

Table 4. Final results of the prioritization. The features are sorted by their share of votes. The cumulated share is the sum of shares for this and all higher-ranked features. The features above the double line represent more than 80 % of votes cast.

Rank	Feature name	Share of votes	Cumulated share
1	Team view	0.198	0.198
2	Consolidated task and story list	0.082	0.280
3	Individual load balancing	0.075	0.355
4	Task deadlines	0.075	0.430
5	Work queue	0.066	0.496
6	Task splitting	0.051	0.547
7	Strategy-to-action (top-down)	0.045	0.592
8	Visibility into calendar	0.040	0.632
9	Strategy-to-action (bottom-up)	0.038	0.670
10	Notifications	0.037	0.707
11	Stand-up-support	0.031	0.738
12	Impediment handling	0.028	0.766
13	Support for WiP reduction	0.025	0.791
14	Monitoring spent effort limits	0.024	0.815
15	Working on now-task	0.024	0.839
16	Filtering tasks	0.023	0.862
17	Task quick-add	0.023	0.885
18	Periodic tasks	0.022	0.907
19	Exporting tasks as appointments	0.018	0.925
20	Do not disturb -sign	0.017	0.942
21	Making resource stealing visible	0.012	0.954
22	Ticket/mail notification integration	0.012	0.966
23	Newsfeed	0.010	0.976
24	Reminders	0.009	0.985
25	Pomodoro support	0.009	0.994
26	Over-/undertime slider	0.006	1.000

5.3 Selection of features for detailed analysis

As many of the features received only nominal support, it was decided that it was not necessary to study them at this stage. Thus an arbitrary limit was chosen: the minimal set of features that would provide 80 % of value would be selected for detailed analysis and for possible implementation. According to the prioritization results, the 14 topmost features would cover more than 80 % of all cast votes (Table 4, Fig. 14 on the next page). Detailed analysis on these features is in Chapter 6.

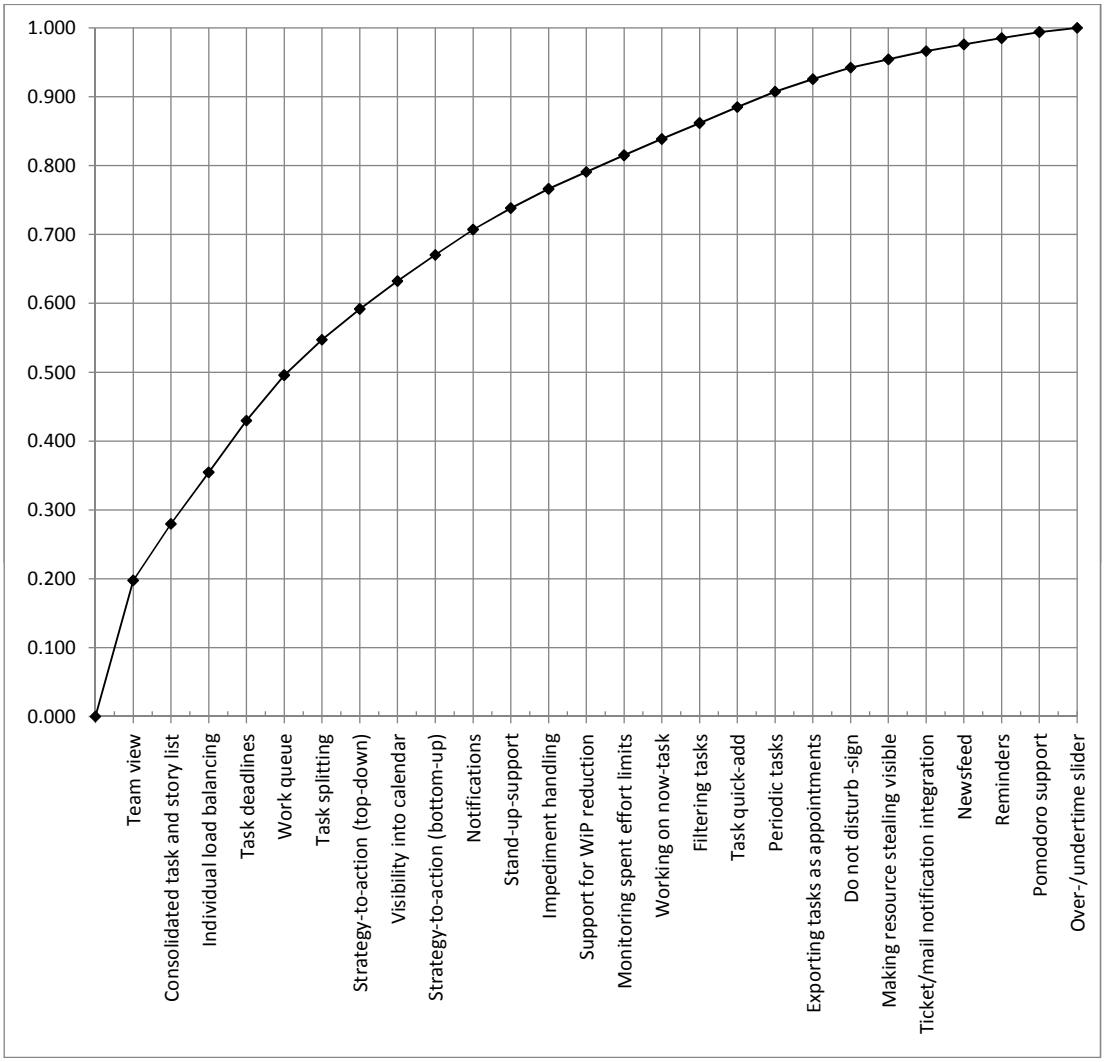

Fig. 14. The cumulative value of features in priority order as a fraction of the value of all feature.

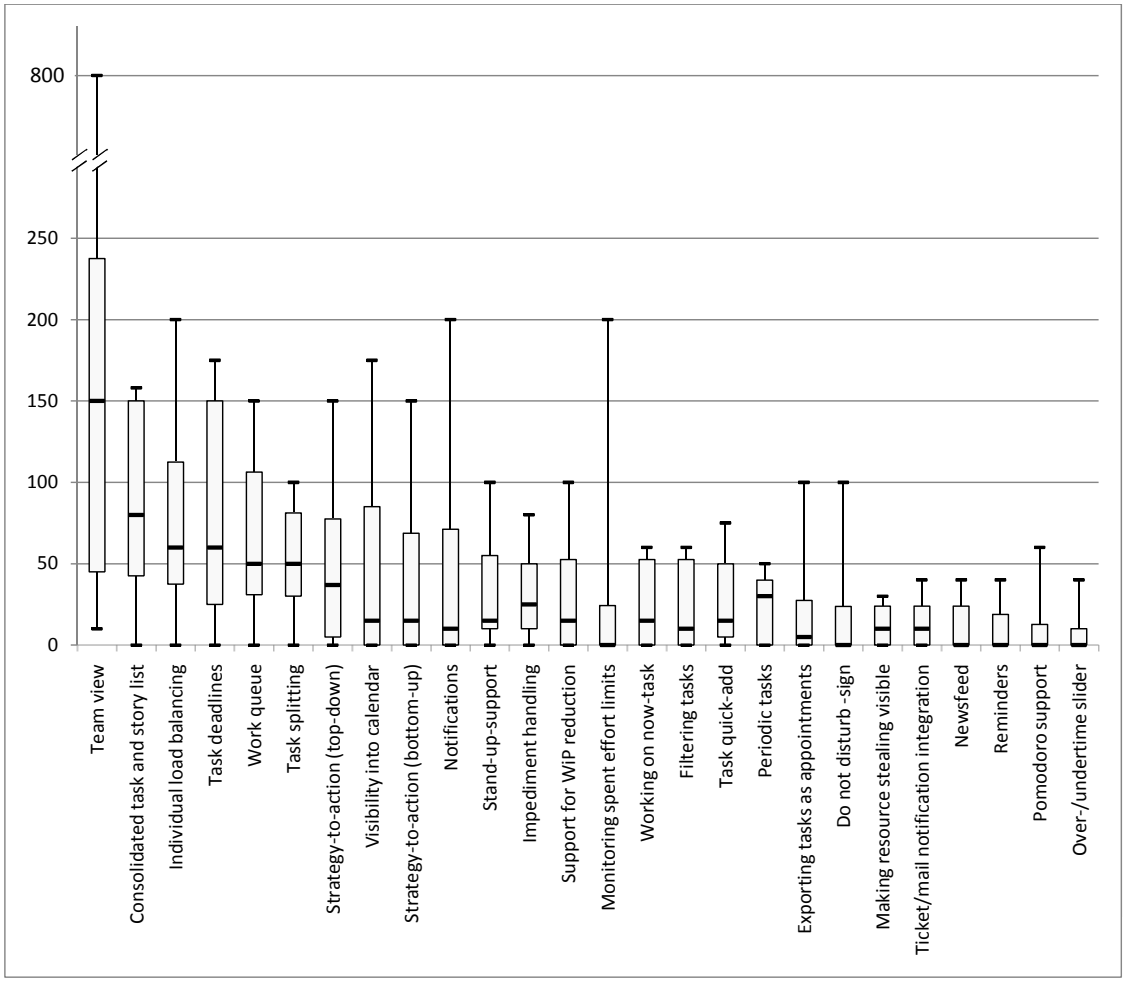

Fig. 15. Distribution of the number of votes given by each interviewee to features, represented as a box plot. The values on the box plot are from bottom to top: the minimum, the first quartile, the median, the third quartile and the maximum of the number of votes given by an individual to the feature.

6 Analysis, design and proof-of-concept implementation of selected features

6.1 Implemented new features

This section discusses the 4 new features that were implemented into Agilefant. The implemented features were "consolidated task and story list", "work queue", "task splitting", "strategy to action (bottom-up)". These features represent 23.7 % of all votes given in the prioritization phase.

6.1.1 Consolidated task and story list

The highest valued feature that was implemented was "consolidated task and story list". The feature was described in three user stories. According to the first, a person with many simultaneous assignments in several iteration backlogs simultaneously needs to be able to see the collection of work he is supposed to work on during a given time period so that he does not have to visit several backlogs to find the items he should be working on. Second suggests that on that view, the person should be able to see the context where his assigned tasks belong to — including the story, iteration, project and product — to make prioritization of work easier. The last one proposes that the tasks should be in a sensible priority order. This proposed view essentially cuts across several of the identified daily work management practices.

The task and story list view is an information radiator, and thus covers the visual management paradigm (3.2.1); it also can be seen as the virtual counterpart of the original visual management practice of Extreme Programming, where the developer would literally have a set of cards to work on his desk. However, the downside of paper and pencil practice would be that no other could see the status of these tasks easily.

Usefulness of the view depends on the used task allocation mechanism (3.2.2). If one-at-a-time is used, the person would most often have only one task assigned to him at any given moment, and the ordering in this view would not be significant. However, if fill-your-bag is in use, the view would contain tens of tasks, and then the ordering and the displayed information becomes significant.

When fill-your-bag task allocation is used, the user would use this view to select the next task to work on (3.2.3), as it lists only those tasks that have been assigned to him. To aid in the selection of the next task, the view should list the tasks of the user in sensible priority order. Furthermore, as a story that is not done is worthless (Jeffries *et al.* 2000), it is better to select a task from a story that is almost complete instead of starting a task in a

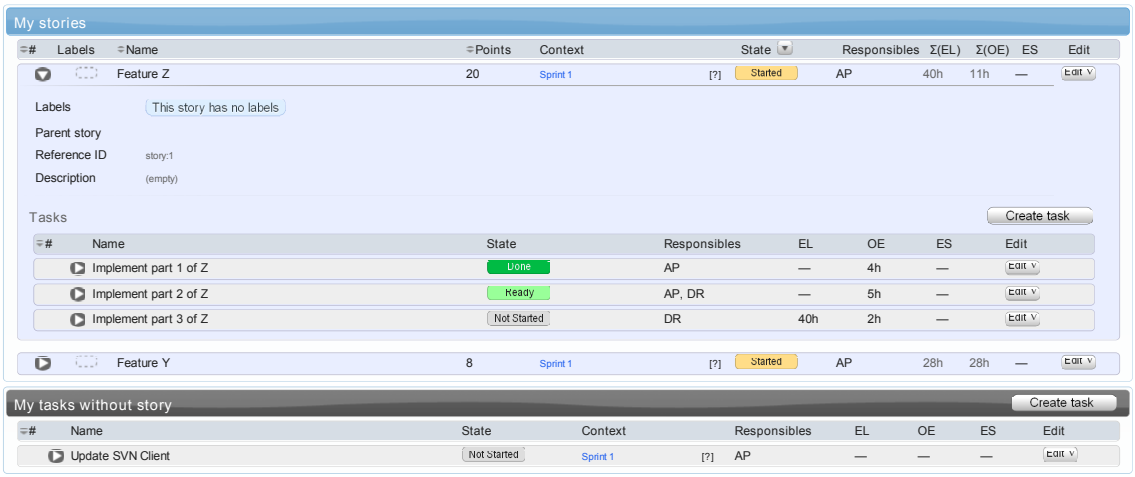

Fig. 16. The consolidated task and story list. The view for a fictional user AP are shown. Above are the currently active stories ("Feature X" and "Feature Y"); below, the tasks without stories ("Update SVN Client").

not started story.

As this view consolidates assignments from various sources, it could naturally be used by the users to enter measurements on current tasks (3.2.4), such as to log the effort spent; to provide the estimates on effort remaining for status tracking (3.2.5); and for the user to observe his current workload (3.2.7). And because they are his highest priority assignments, the Scrum master of a team should have the team impediments displayed prominently on this list (4.8).

The consolidated story and task list view feature was designed based on these criteria (Fig. 16). The view shows all assigned and currently active tasks for a user, consolidated from various sources. The logic for gathering the work for a user is:

1. Find all iterations which overlap with this day, e.g. their beginning is before 23:59 hours *and* their end after 00:00 hours today.
2. In the backlogs of iterations from the step (1), find all tasks that are not yet marked "done" and for which the user is responsible.
3. For each task found in the step (2), if the task is connected to a story, then add the story to the set of current stories. If not, then add the story to the set of current tasks without story.
4. In the backlogs of the iterations from the step (1), find all stories that are not yet marked "done" and for which the user is responsible.
5. Add the stories from the step (4) to the current stories.
6. For each story from step (5), couple them with their constituent tasks.

The consolidated task list is rendered based on these two sets; Fig. 16 shows the consolidated task list for a user named AP; in the figure are two currently active stories, "Feature X" and "Feature Y", and one task without story, "Update SVN Client". The "My stories" and "My task without story" lists have both a total order, based on the comparison criteria represented in Fig. 17.

Finally, the constituent tasks of each story are ordered by their relative rank. Thus, the relative priority of entities in the view is in general such that the one that is above is at a given time more important than the one that is below. These priorities can be changed on

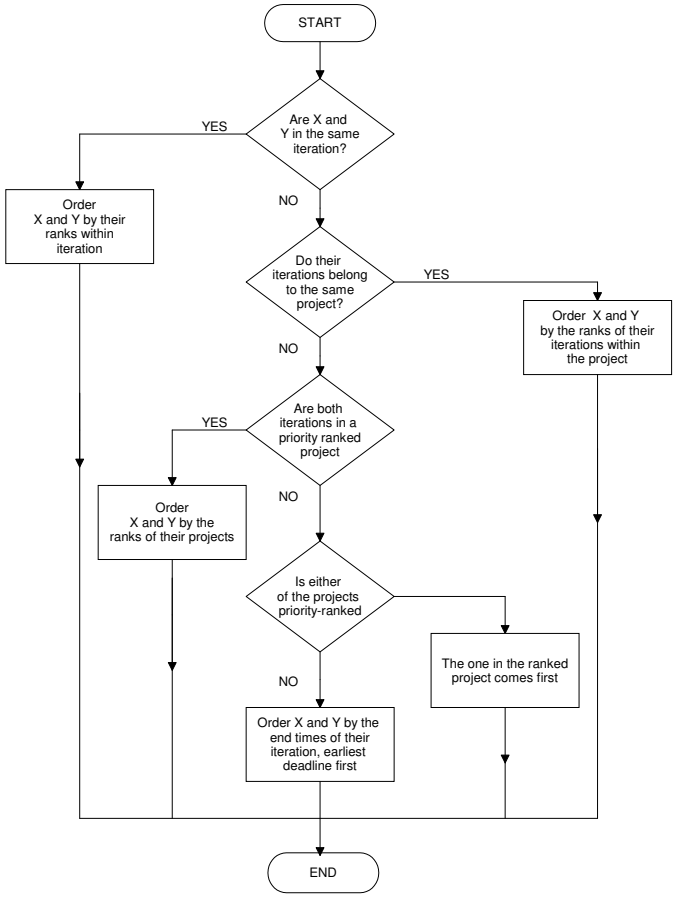

Fig. 17. The ordering comparisons for stories and tasks without stories in the consolidated list view.

various views: for example, if the senior management makes changes in the portfolio view, changing the relative ranks of projects in the portfolio, this will be reflected within seconds on the users' consolidated task lists, corresponding to this portfolio decision.

The selection of next task or tasks to work on is then straightforward — the user should choose the highest-ranked, not completed tasks assigned to him. (In Fig. 16, the "Implement part 3 of Z" in "Feature Y" is the highest-ranked not completed task, as the other tasks in the story "Feature Y" have already been implemented). However, if a lower-ranking story is almost completed, the user might choose to not start a higher-ranking story and instead help complete the lower-ranking one.

The consolidated task and story list functionality also helps one to log timesheet entries to items, by double-clicking any cell on the ES (Effort Spent) field and entering number of hours to add to the task. Likewise, the user can enter the new estimated effort left on any task by double-clicking the EL (Effort Left) field on a task and entering the estimated amount of remaining working hours. The consolidated task and story list can also be used to decide for which tasks he should get help. This implementation does not address the impediment tracking and handling, because the feature impediment handling was not yet implemented; a design proposal for impediment handling is discussed in Section 6.2.9.

6.1.2 Work queue

Another new feature is the work queue, which can be also described as a backlog of an user for a mini-milestone. It is a per-user construct that contains an ordered list of tasks which the user plans to work on until the next stand-up meeting. It is yet another form of

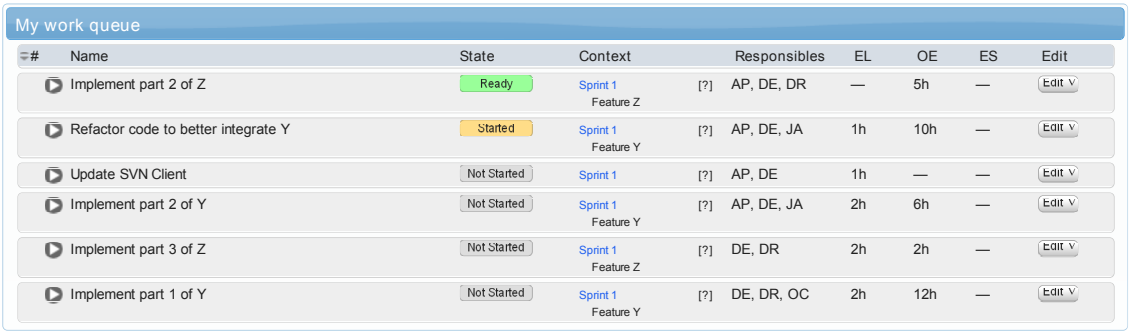

Fig. 18. The work queue. An example work queue of the user DE.

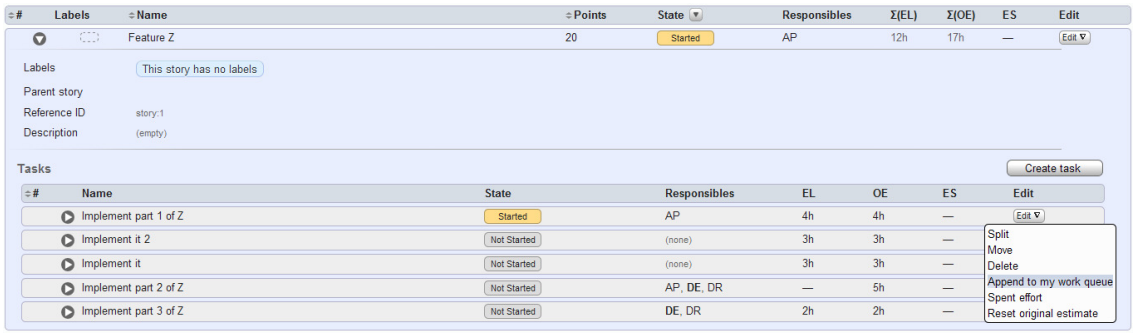

Fig. 19. The Edit menu for a task. Appending a task to the work queue is achieved by clicking on the "Append to my work queue" on the Edit menu on any task. A task can be split by choosing the "Split" menu item.

information radiation that tells the other team members the near-term plans of a team member, thus a form of visual management (3.2.1). It can aid with the task allocation methods (3.2.2); in the one-at-a-time strategy, a team member can append the task to his work queue instead of just adding him responsible for the task. In fill-your-bag, a team member can signal in advance on which tasks out of many tasks he will work on next without having to change the status of the task to "started" in advance. Furthermore the team member can plan his day in advance: gather enough tasks for the entire day (3.2.3) to the queue and work on them without interruptions. The work queue supports the daily stand-up meeting practice; the answer to "what you are planning to do until the next meeting" is reflected on the work queue. Finally, it supports the maintaining focus practice (3.2.10), because forces the team member to rank his selected tasks.

The implemented work queue feature is shown on Fig. 18. The tasks in the work queue can be ordered by drag-and-drop technique; every user has their own work queue that they can prioritize independent of others. Tasks in the queue can be edited in the same way as in the iteration backlog and the consolidated task and story list. To add the task to his own work queue, the user can click on the Edit menu on a task anywhere and select "Append to my work queue" (Fig. 19). This menu item not only adds the task to the bottom of the queue, but also automatically assigns the user responsibility for the task. The work queue status is reflected on the tasks everywhere by displaying the initials of those developers who have a task in work queue in bold face. When the task is marked "done" it is automatically removed from every work queue.

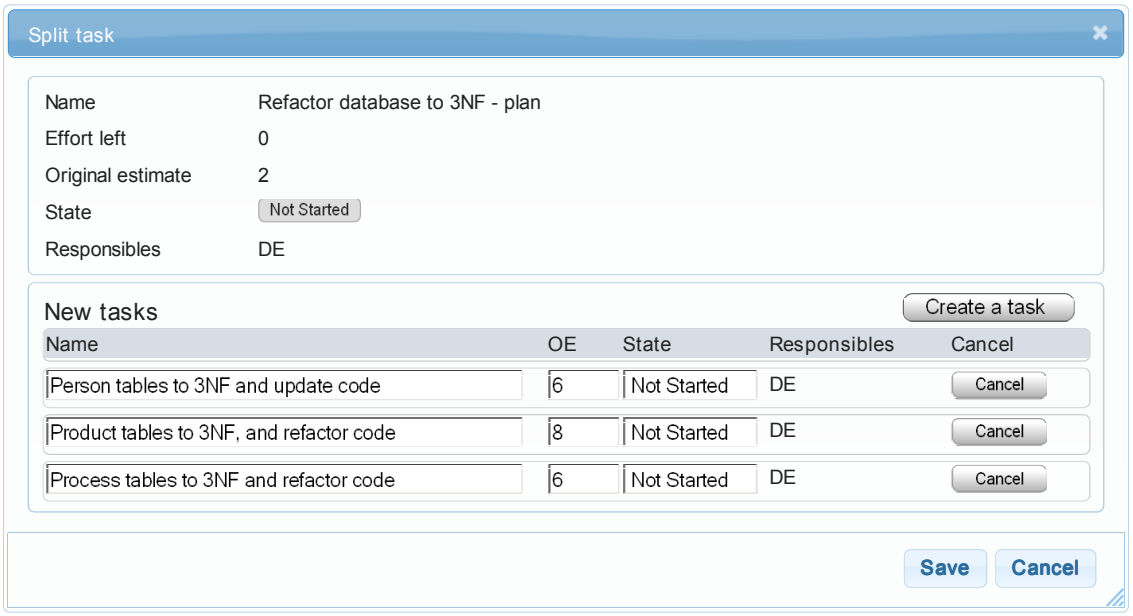

Fig. 20. The task splitting dialog, showing one task being split into four. The original task is above, and three new tasks below.

6.1.3 Task splitting

Third feature that was implemented was "Task splitting". In 3.1.5 it was noted that the iteration plan does not have to be complete; tasks larger than 1-2 days can be used as placeholders for more fine-grained work items. As such, splitting a task is not part of managing mini-milestones, but deferred iteration planning. However, because it is done within iterations, it needs special considerations to make it not interfere with the ten practices and the features to support them. Even though it is possible to create a new task under the container where the original task resides, such approach has poor usability, as other data from the original task needs to be copied to the newly split tasks, such as the responsible team members and the positions in developers' work queues.

Thus a task splitting feature was implemented. A task can be split by opening the Edit menu on a task and choosing the "Split" menu item (Fig. 19). The task splitting dialog is opened (Fig. 20). In the task splitting dialog the original task is shown; the user can add as many new tasks as he wants. Each task inherits the responsibles, rank, container and work queue positions of the original task. The new tasks shall be ranked below the original task in every container, so that the topmost new task in the dialog shall be the one ranked immediately after the original.

6.1.4 Strategy-to-action (bottom-up)

The last feature that was fully implemented was named "Strategy-to-action (bottom-up)". The feature concerns following the story hierarchy from a task up to the upper level of strategic investment themes and epics (Leffingwell 2008). It is meant for developers to understand the origins of the work item they are contributing to, and to assure them that they are doing the right thing. This feature was represented by two user stories. The first user story describes the need of a user to be able to navigate the story hierarchy from a

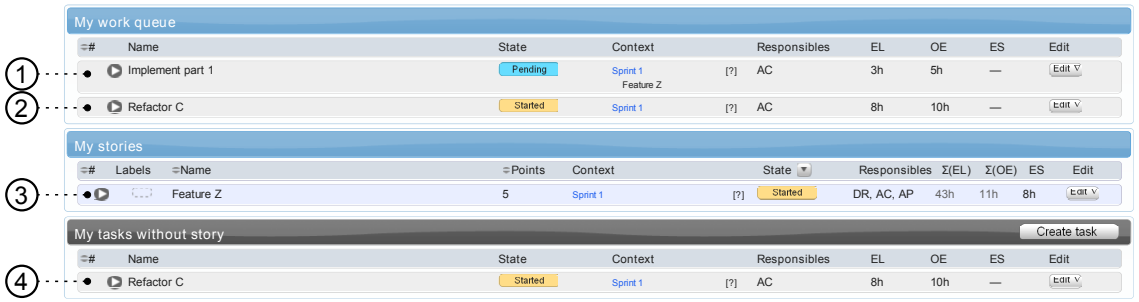

Fig. 21. Contexts of work items on the daily work view. The numbered items are (1) A story task in the work queue. Belongs to a story named "Feature Z" in iteration backlog named "Sprint 1". (2) A task without a story, in the work queue. An independent task from iteration backlog "Sprint 1". (3) The story "Feature Z" in the consolidated task and story list, from iteration backlog "Sprint 1". (4) The same task as in (2), but in the task and story list.

The Context field displays the name of containing iteration for all items; for a story task in the work queue (1), the name of the containing story is shown below the iteration name.

task to an upper level story. According to the second, a user must be able to do that from both the consolidated story and task list (6.1.1) and the work queue (6.1.2).

Thus to fulfill this story, if the tool supports hierarchy of stories, with links from leaf stories to higher-level branch stories, the story tree should be easily navigable from tasks attached to leaf nodes. Agilefant introduced hierarchical stories in version 2.0. The tracing of story hierarchy from tasks to epics had already been planned on Agilefant prior this thesis was commenced; the data model was organized in such way that every leaf story is linked to its parent stories, and the immediate children of a branching story are also priority ordered (see Fig. 13).

As the navigation from a story in iteration backlog to higher-level context was being implemented by other developers, only the second story had to be implemented to support this feature: easy navigation from daily work. To accomplish this, an additional field was added to the representation of a task, or a story in the work queue. This field contains the *context* of the item. For tasks without story, and stories, the context is the iteration backlog where they reside. Additionally, for tasks that belong to stories, that includes the name of the story. (Fig. 21). Clicking on the name of the iteration in the Context field, the user can navigate to the respective iteration backlog.

By clicking on the small "[?]" symbol the user can open a context popup that can be used to navigate the story hierarchy of the attached story, if any, and the hierarchy of containing backlogs (Fig. 22). The item names in the context popup are also links to respective backlog views.

6.2 Proposed designs for remaining features

The remaining 10 of the 14 top-value features are analyzed in this section. Each feature is reflected on the identified daily work management practices and other information available in the book sources. A proposal of possible implementation is provided for features, along with possible workarounds that can be used to accomplish the same value

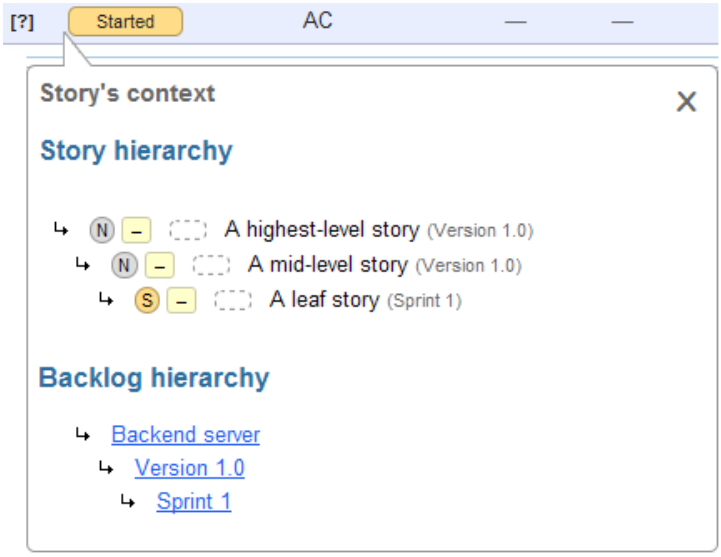

Fig. 22. The context popup for a story in consolidated story list view, showing the story and the backlog hierarchy. On the story hierarchy, this story (named "A leaf story") and all its ancestors ("A mid-level story" and "A highest-level story" are shown, with their statuses, size estimates and the containing backlogs of each story in parenthesis. In the backlog hierarchy, the iteration backlog that contains this story ("Sprint 1"), its project backlog ("Version 1.0") and product backlog ("Backend server"). This figure is of the final version of the feature, for which the author provided a preliminary implementation.

with the existing system.

6.2.1 Team view

The feature that received most votes was the Team view that was described by 6 workshop stories. According to the stories, the team view would display the combined task and story list of all team members, along with the current work queues of all members, and their load indicators. This would also make easier to observe the differences in work load between different team members.

This feature is troublesome to a tool that does not support teams as first-class entities, as is the case with Agilefant. Even though partial support for team view could be implemented on top of "teams as lists of users", the current many-to-many mapping approach in Agilefant does not provide answer to the simple question "which team do I belong to". Also, because iteration backlogs are not owned by teams per se, it is difficult to automatically gather historical data on team performance, as would be required for calculating the team velocity in Extreme Programming (Beck 1999). The team velocity is measured in story points per iteration; how should the completion of stories contribute to the team velocity, when the tool does not know which team should have its velocity increased. Thus a proper implementation of team view would require that it should be possible to assign teams to projects and iterations instead of users.

The implementation of this feature is further complicated by the design of Agilefant, which mandates that the iteration backlogs are aggregated into release projects, and inside products, instead of being the property of the team. If the team that is working on multiple projects simultaneously is forced to work on multiple iteration backlogs at a time, without

a possibility of mixing the tasks and stories belonging to different products in the iteration plan, there will always be ambiguity on the mutual priorities of stories and tasks. In that case the combined story and task list for a team would essentially be a non-prioritizable backlog. However, as the iteration represents the first level on which the work is assigned to developers, the iteration plan could be owned by the teams, too.

Ultimately, to fully support stable feature teams with dedicated team members (3.2.9), the teams would have to be the owners of iteration backlogs. Each team would have at most one active iteration backlog, to which stories can be selected from multiple projects; the team, or its product owner could provide relative priorities for stories from different products. This would simplify the selection of next task as there would be consensus on the ordering of pieces of work among the team; outsider stakeholders could visit the team backlog to ensure that the stories from different products are in correct priority order. The allocations of these stories to teams would still be visible in the project backlog. This would not invalidate the need for consolidated task and story list for an individual developer, as it would still be useful for specialists that are shared between teams.

6.2.2 Individual load balancing

This proposed feature would introduce new features for visualizing and balancing the workload. One user story, "balancing work queue load based on calendar appointments & holidays" requests that possible vacations, holidays and appointments in the calendar should be taken into account when the work organizing in the work queue, while the other, "Load shifting according to work queue" states that the contents of work queue should be added to the load on this day, and be subtracted from other days.

To be implemented as requested, the former story would need the "Visibility into calendar" feature, to import appointments and vacation days from an external calendar. Alternatively, a tool should contain a true calendar functionality or at least "daily availability" calendar, where each team member could enter the hours they can work on each day of the iteration. Nevertheless, there is also an easy workaround: for every appointment or vacation day that overlaps with the iteration, a placeholder task can be created, with the effort estimate of total working hours that will not be available due to this activity. Then, as the activity is progressing, or completed, or the vacation is taking place, the effort estimates on these placeholder tasks can be decreased. If the placeholder task would represent vacation, or sick leave for an absent team member, others can take care of it

The latter story is not as useful as it might at first appear: if the largest tasks are 12 to 16 hours in size, the straightforward implementation of this user story would require that this amount of effort be added to the load of current day, and subtracted from elsewhere; however, such task naturally could not be completed within a single day by a single person, so it would have to be continued on the following day. This adjustment would not make the visualization any more correct. Thus in this case the average load over the iteration is the only meaningful metric. The second story will not be likely implemented.

To meet the objectives of the user stories of this feature, the visualization and calculation of the workload of an individual team member should be enhanced using actual working times for availability calculations. It would require either integration with existing calendaring systems, or a simple iteration calendaring functionality within the tool.

6.2.3 Task deadlines

The fourth-most valued feature was the addition of optional deadlines for tasks, and sorting tasks by deadline in views. This deviates from both Scrum and Extreme Programming as the increment produced by the team is guaranteed to be complete only at the end of the iteration, and the implementation order of tasks is left for the team to decide; the only deadline for all tasks and stories is the end of the iteration timebox. The ordering of the task is signaled by their prioritization in the iteration plan.

The task deadlines feature would require an optional deadline field to all tasks; the views that contain tasks should be sortable by deadline also. A simpler solution exist: if the deadline is essential, one can prefix the *name* of the task with the deadline, such as "[DL:06-24] Fix the bug #NNN for customer X". Then one would rank those tasks everywhere by deadline in the iteration plan, earliest deadline first; and only work on tasks in this priority order. Analysis by Liu & Layland (1973) proves that this scheduling algorithm is optimal for one processor (or developer), if the set of tasks as a whole is schedulable, that is, all of them can be implemented within the timeframe. However, due to the inherently imprecise nature of task estimates, a software tool might not be able to analyze if the tasks can be executed in time; thus it is also doubtful that the deadlines would produce valuable information to the workload calculations.

Even if the deadline field would not be added to tasks, the sorting by deadline feature can be useful. Sorting by deadline should especially be considered for the consolidated task and story list; because it is specifically used to consolidate assignments from several overlapping iterations, the iterations might also have differing deadlines. The approach taken by Agilefant specifically already allows a developer to be assigned to multiple simultaneous iterations; each iteration has its own timebox that does not be in sync with other simultaneous iterations.

6.2.4 Strategy-to-action (top-down)

Three of the user stories concerned functionality that would allow users to monitor the state of leaf stories and their tasks on daily levels from the top-level stories in product and release backlogs. One of them was team-member-oriented; with this functionality a developer could see the progress of the parent epic (Leffingwell 2008) of the task he is working on. The two other detail use cases for managers who want to monitor higher-level stories in terms of tasks can see how the tasks the developers and teams are working shall contribute to the business objectives. As these all concern tracing the story hierarchy from strategic level down to engineering tasks — though by different actors and motives — they were combined into one proposed feature called "Strategy-to-action (top-down)".

Tracing of story hierarchy from highest-level stories in the product backlog down to leaf stories in iteration backlogs had already been implemented in Agilefant, but the tasks are not visible in this hierarchy. The primary reason is that unlike the story hierarchy in the product backlog and the release backlogs that are long-living artifacts, the tasks are only an aid for the team to manage its work within the relatively short iterations. Also the meaning of progress is vague as task effort estimates do not tell anything about the progress of the stories in hierarchy; except the estimated number of hours that will be spent to implement a subset of the hierarchy during ongoing iterations.

6.2.5 Notifications

The proposed "Notifications" feature would add functionality for notifying an user of the tool of some asynchronous event. As such, it does not directly concern any of the identified daily work practices; it is merely a communication aid between parties who do not work face-to-face.

The "Notifications on task completed" story details functionality, where an interested party can request the tool to inform whenever a task has been completed. On tasks, this could be realized by adding an item in the Edit menu for "Notify me of changes", which would open a dialog, where the user could choose what kinds of events he is interested in. When the task under interest is marked done, a notification request could be stored in the database. At any time, as the interested party uses the tool in a web browser, all outstanding notification requests could be pushed to the browser and displayed prominently on the top of the screen, until they are marked acknowledged by the interested party.

The second user story, "Assignment notifications" would be a straightforward extension to the notifications system. Only this time it would work without the targeted user having to mark the task as interesting. Instead, whenever the user is being assigned responsibility by another user, a notification request for the assignee on assignment by another user would be stored in the database.

It should be however noted that in both Scrum and Extreme Programming assignments without consent would be considered breach of rules, and thus the usefulness of this functionality questionable in those methods. Also, in neither Scrum nor Extreme Programming would a product owner or a project manager be monitoring a single task — opposed to a story — as was written on the user story, it could be used as an aid for self-organization.

6.2.6 Visibility into calendar

"Visibility into calendar" feature concerns integration of the tool with an external calendar application. This item can be considered a true epic (Leffingwell 2008) in nature, due to variety of existing calendaring systems and interfacing standards. Nevertheless, it could be very useful more involved team members, such as the product owner, in that it would allow him to have all stakeholder meetings in the calendar application, but he could nevertheless observe its contents through the tool. In addition, the possibility of importing appointments as tasks would make the calendar entries automatically contribute to the load calculations. The external calendar could also be to calculate the daily work time for the team member (see Section 6.2.2). This feature will very likely never be implemented in Agilefant due to its complexity.

6.2.7 Support for WiP reduction

This feature contains support for reducing the work in process. The four user stories that constitute for this feature, "Minimizing features in progress for a team", "Story horizon", "Blindfolds" and "Minimizing tasks in progress". They all provide different approaches to the same problem: avoiding increased multitasking and utilization rates that would only lead to increasing the lead times for stories and decreased efficiency of engineering (Larman & Vodde 2008). The three first user stories discuss different aspects of the same thing: the team should concentrate on working on a small set of user stories at any time. Thus these user stories wish to enforce a specific method by which the next task to work

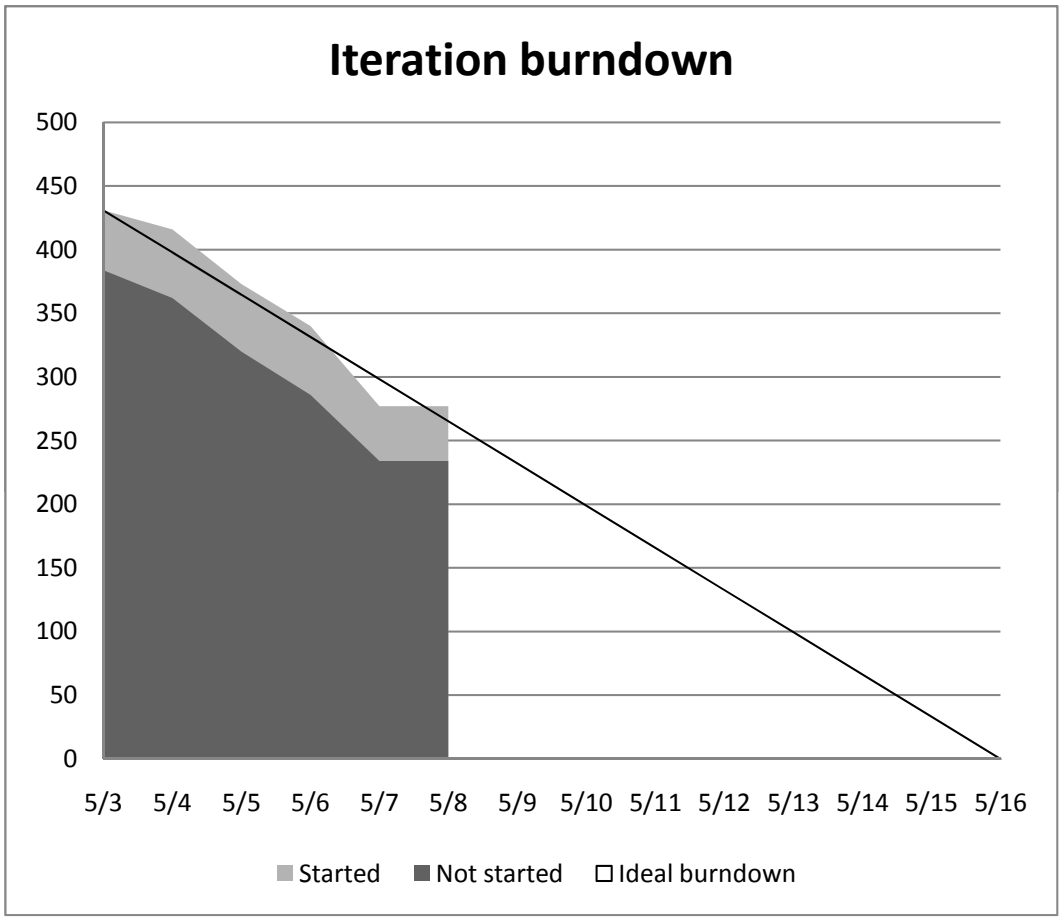

Fig. 23. Enhanced iteration burndown diagram, which differentiates started tasks from not started tasks.

on is chosen (3.2.3). Instead of making the system to enforce it, the developers could be trained to concentrate on completing started stories first. The status meetings (3.2.6) can be used to spot deviations from the practice, and to monitor the number of started stories. The practice of selecting of tasks from already started stories is also supported by the consolidated task and story list feature (6.1.1).

The last story, “Minimizing tasks in progress”, requests that an individual team member should be able to minimize number the tasks he is working on, and optionally to visualize the amount of started tasks he has been responsible for during the iteration. This is a reference to the maintain focus practice. The first half of the story is supported by the work queue feature (6.1.2), because it is easy to monitor the number of started tasks on this view. As such, the first part of the fourth story can be considered fulfilled, whereas the second, optional part, of limited functionality.

However, at the iteration level, the iteration burndown could be enhanced so that it would display the effort left in started tasks separate from the remaining effort in not-started tasks (Fig. 23). To address the need of monitoring the number of open stories, another similar burndown diagram could be added at the iteration level that could either show the number of remaining stories and number of started stories, or their story points.

6.2.8 Stand-up support

The "Stand-up support feature" contained one story, "Viewing Done / Impediments / What's next for stand-ups". It describes functionality that would aid a team member when answering the questions in the status update meetings (3.2.6). It specifically requests that the workaround using the "ready" task status should not be needed.

To fulfill this user story, a view should be provided that would display all those tasks for which the team member is responsible and which have been marked "done" since the team member last attended a status-update meeting; it should also display the work queue of the team member, if that feature is provided, and also all current impediments the team member has reported.

6.2.9 Impediment handling

The "impediment handling" feature was represented by two user stories: "Logging impediments" which concerns an easy way to write down impediments, be they related to work items (a task, a story), containers (an iteration), the team, or the general organization, so that they can be recalled easily in status update meetings or in sprint retrospectives. The other story, "Impediment radiation" describes the need of a team member of keeping the Scrum master aware of the current impediments so that he can remove them.

The full implementation of this functionality depends on other features. Foremost, to facilitate logging team impediments, as distinct from iteration level impediments, the tool should support teams as first-class entities. Because an organizational impediment can seldom be resolved within an iteration, the impediment entities should not be forced to be contained within an iteration backlog.

As in Scrum the impediments form the highest-priority work of the Scrum masters, and impediments can be also prioritized, the implementation could handle them as a special class of tasks. The Scrum master could see them in the consolidated task and story list, and be able to prioritize them in his work queue.

Until a tool has support for the stable feature team concept, the workarounds described in Subsection 4.8 can be used to mimic proper impediment handling. If the tool supports feature teams, and especially if the users can assume team roles, the implementation of impediment handling and impediment backlogs should be considered.

6.2.10 Monitoring spent effort limits

The last feature to fit under the 80 % limit was "Monitoring spent effort limits". This feature would notify the Scrum product owner as soon as an effort limit on a work item is exceeded. This was represented by one user story that requires the product owner to be notified "if effort spent on a task has exceeded the original estimate", thus demanding that it is a task specifically, not a larger item, on which the effort is monitored. However, the original estimates on the tasks are not upper limits, but instead they are supposed to be a tool for tracking the status of the iteration and to comprehend the load. Larman & Vodde (2008) call this confusing estimates with *commitments*, and cite the Merriam-Webster dictionary: an estimate is "a rough or approximate calculation". Furthermore, in order to be useful during the iteration planning, and tracking the iteration status, the original and updated estimates on the tasks should be *accurate*, but they do not need to be *precise* (Cohn 2004). If the team is very skilled in estimating, their original estimates must be

exceeded exactly in 50 % of the cases.

As such, this feature will also be the least likely one to be implemented in Agilefant. If there is a need to complete some task within some effort limit, then it can be agreed so within the team. In that case, the team member working on the task would then have to report in a status-update meeting if he could not finish the task before the time limit.

6.3 Conclusions

Of the analyzed features, 4 were fully implemented in Agilefant, and thus a proof-of-concept of these features is included in Agilefant 2.0; the source code for Agilefant is available under the MIT license. All implemented features also matched requirements or practices indentified in the literature.

The remaining 10 were not implemented at all. For many use cases in the remaining 10 stories, a usable workaround exists. Some of the features that were valued by the interviewees match very closely with both Extreme Programming and Scrum: “Stand-up support”, “Team view” and “Individual load balancing”. “Impediment handling” matches Scrum directly. Of the remaining features, the “Visibility into calendar”, “Task deadlines” , “Strategy-to-action (top-down)” and “Support for WiP reduction” matched neither Scrum nor Extreme Programming method directly but they could be implemented in a tool while remaining mostly compatible with Extreme Programming and Scrum.

The “Notifications” could improve intra-team communication in distributed teams, and coordination among teams that have dependencies to each others’ work. Finally, the “Monitoring spent effort limits” practice, that is, considering estimates as commitments, or managing against fixed effort limits is considered an antipattern by many sources, such as Leffingwell (2007), Larman & Vodde (2008), Beck (1999) and Schwaber & Beedle (2001). None of these 14 features addressed establishing cadence, or measurements that were not derived directly from the task or story statuses and estimates.

7 Discussion

This chapter contains discussion on the thesis. First, the book review is discussed in Section 7.1, followed by discussion on the acquisition of empirical data in Section 7.2. The analysis, design and implementation phase is discussed in Section 7.3. The chapter concludes with discussion on the validity of the research in Section 7.4.

7.1 Book review

The book review for daily work practices and tools was performed on a wide set of sources on purpose. All daily work management aspects are not discussed in detail in any of the books, and there are opposing viewpoints to certain practices in the literature. Most notably, Cohn (2004), a source that was not included in the book review, promotes the fill-your-bag method for task allocation; however in a later book, Cohn (2005) apologizes for this and now recommends only one-at-a-time. Also, Larman & Vodde (2008) recommend that senior management should especially be present in the status update meetings, to promote the Genchi Genbutsu principle of Toyota Production System; however, the "Scrum Primer" appendix included in the very book discourages this as the authors fear that the team is not willing to communicate freely if senior management is present.

Based on an quick survey using Google Scholar, Schwaber & Beedle (2001) and Schwaber (2004) were most cited sources for Scrum, as are Beck (1999) and Beck & Andres (2004) for Extreme Programming; the latter ones were cited in June 2010 ten times as much as Beck & Fowler (2000), and over 50 times as much as Auer & Miller (2001). However, none of these sources are particularly prescriptive; in fact in the foreword, Beck (1999) acknowledges that the first Extreme Programming book is not a how-to guide, and thus does not contain checklists of specific practices. And later Beck & Andres (2004) state that the reason for writing the second edition of Extreme Programming Explained was that the first book was in Beck's opinion still too prescriptive; the second edition further distances itself from the practices, being more of a journey to the philosophical background of Extreme Programming. Thus to review only these books for possible management practices would be incorrect.

To the author's knowledge this thesis contains by far the widest book review on daily work management practices on agile methods in the practitioner guidebooks, but there are many more books that could not be covered within a limited timespan, so the results of the review do not necessarily reflect the entire body of knowledge. The reason why research articles on daily work management practices was not included was the scarcity of sources: Dybå & Dingsøyr (2008) found in their systematic review only 36 empirical research

papers on agile methods until the end of 2005; only 4 of them were studies on mature teams, and they were all in Extreme Programming. Scrum, on the other hand, was studied in exactly one paper, and the practitioners were beginners. Arguably, however, for the tool to not be obstacle for development, only the studies that concern mature teams should be considered.

7.2 Discussion on the acquisition of empirical data

7.2.1 The daily work management workshop

The daily work workshop was used to elicit features for a tool from current practitioners, in the requirements workshop format (Leffingwell & Widrig 2003). The approach is unusual for research, but usual practice in software development. According to Leffingwell & Widrig (2003) it is essential that the key stakeholders to the project participate. In this case, they were the liaisons from ATMAN participating companies and ATMAN researchers. As such, there might be a selection bias, as the project might attract companies with a specific profile and specific interests, and thus the workshop results cannot be generalized to all agile software development organizations.

The success of a requirements workshop also requires that the participants share a common understanding, thus in the beginning there was an initiation session, where the idea of daily work management was explained, and then everyone wrote down in a 5-minute timebox what efficient daily work management should contain. Everyone then represented their ideas to all participants, succeeded by a short discussion. This initiation was not sufficient for building a shared mindset, however, as several user stories created were out-of-scope from the mini-milestone cycle and even the iteration cycle. Also, even though it was emphasized that the target of the workshop is to gather requirements for a software tool, several of the produced user stories were not translatable to software requirements, as they were addressing organizational culture. Despite these deficiencies a comprehensive set of stories was elicited. However, the 2.5-hour timebox that was allocated for the research was too short, and the number of produced user stories was too big to be prioritized in the workshop.

7.2.2 Prioritization

At first it was supposed that the prioritization of the user stories could be accomplished in the workshop. As the participants were from different cities, another workshop was not an option, so an alternate method for gathering the data for prioritization had to be chosen. Furthermore the number of user stories proved to be too big for prioritization. In the end, the hundred dollar method was chosen. To maximize the coverage of prioritization, the questionnaire was sent to liaisons in other Finnish software companies that were using Agilefant, in addition to workshop participating companies.

7.2.3 Results of prioritization

The results of the prioritization are mostly in line with books; however, features that were directly opposed to the self-organizing teams were among the 80 % set. Furthermore, only one feature was not considered totally worthless by some participant; at least one

interviewee cast no votes for each of the remaining 25 features. This might suggest that the needs of current agile software organizations vary greatly. In a private conversation, the interviewee who cast 800 votes for the "team view" told that the lack of it is the only reason for them to not adopt Agilefant in their organization, as they consider it essential for self-organizing teams. However, this feature did receive only 10 votes from another interviewee, which suggests that they do not consider self-organizing teams at all important, thus questioning their commitment for truly agile methods like Scrum or Extreme Programming.

7.2.4 Conclusions on the empirical data acquisition

In retrospect, the workshop and prioritization combination was not entirely satisfying approach for eliciting requirements for daily work management features in a tool; in total, each participant to the workshop had spent 2.5 hours at the workshop, and more time traveling to the meeting place. According to the reports, answering to the prioritization questionnaire took almost one hour approximately.

A structured 2.5 hour interview on each of the ATMAN participating companies could have produced more reliable data on their needs, and taken less time from the stakeholders as they would not have had to travel. Also, an interview would have provided data on their current development process, on the interviewees' stances towards agile software development and on their history of practing agile software development. The interview could have further be used to assess the maturity level of the agile practices in their organization (Leffingwell 2007). All in all, the workshop approach did result in less work for the researcher due to the participants processing the data themselves, but an ordinary structured interview could have worked better.

7.3 Discussion on the design and implementation

The designs of features were based on the book review and the sole judgment of the author; thus they are the product of one mind. For the 10 features that were designed but not implemented, it is not yet proved that the approaches work. The proposed workarounds have been tested in a real-world setting: the ATMAN project has used Agilefant for governing its research work, and the presented workarounds have been all successfully employed by the ATMAN project. Also, the implemented features have been all extensively tested when managing the work in ATMAN project. The implementation of proof-of-concept features was done on top of an existing system, Agilefant. Although Agilefant has a large user base, due to an almost complete rewrite of the codebase being done on it at the time of writing this thesis, the software organizations were not willing to experiment with it and thus no data on real-world performance could be gathered.

7.4 Discussion on the validity of the research

The validity of the research, and threats to its validity, is discussed in this section. Each separate phase of research is discussed in its own section.

7.4.1 Validity of the results of the book review

The main threat to validity of the book review results is that a book would not reflect the current best practices adequately and thus relevant practices were omitted or outdated practices retained. To avoid this, a wide selection of the books was included in the book review. The additive nature of the review still constitutes a threat to validity, as the newer books may omit mentions to outdated practices, which still are present in older books — but to concentrate on newer sources, such as Larman & Vodde (2008) or Leffingwell (2007), would itself be a threat due to their focus in large-scale setting, which might compromise their interest in the lowest cycle of control.

In addition, the nature of both Extreme Programming and Scrum is empirical, thus variations of the methods are likely to be widespread; furthermore, as there is no prescriptive book for daily work management in agile methods, it is not easy to tell if the successful teams are doing the practices by the book, or not. Furthermore, the books still are a collection of success stories and works of opinion; the failed cases are seldom mentioned. Unfortunately there is no remedy to this; while the Extreme Programming practices have been studied widely, Scrum has not received much attention in empirical studies (Dybå & Dingsøyr 2008). And the Extreme Programming research tends to be leaning towards adoption phase and beginner teams (Dybå & Dingsøyr 2008).

One threat to the validity was the long time span spent on the literature review; the author feels that during the course of book review, as knowledge was gained, the appreciation of things in the literature might have changed. All evidence of daily work management practices in the books were not always present on those pages sections where they could have been based on table of contents; thus the books had to be read thoroughly. This combined with the large number of books, the book review process could not be as systematic as it would have been with a lesser number; however as the sources were considered to complement each other, this would require that the missed practice was not identified in any of the reviewed books.

7.4.2 Validity of the results of the requirements elicitation

The participants in the requirements workshop were volunteers, invited from a small set of case companies. The selection of participants was without doubt biased, as the external participants were from ATMAN funding companies. Most of the participants had been followers of ATMAN research since its inception; this history needs to be also considered as a threat to internal validity. As a part of the workshop, there was an orientation practice, whose purpose was to make everyone to have a shared mindset of the problem; while this might have made the user stories more coherent, it is also a threat to validity, as this orientation phase might have caused the interviewees to be hesitant of expressing needs and ideas that deviated from the consensus. Arguably, more eccentric ideas might still not have passed the prioritization phase.

Even though the instructions for the user story creation were clear, and it was emphasized that the purpose of the workshop is to elicit features for a software tool, one participant had produced a set of stories mostly concerning organizational culture; none of his stories were possible to capture in a software product. This proves that the workshop method did not provide the best possible results in his case, as his contributions were eliminated from the set of user stories before prioritization. The fact that researchers participated actively in the workshop does not per se invalidate the results of the workshop, as the researchers had also been members of software development organizations, and

working on products, such as Agilefant. Furthermore, researchers did not participate in the prioritization.

7.4.3 Validity of the prioritization

The 100-dollar method seems to imply that every stakeholder is considered to belong to the same market segment. Because the prioritization results varied widely, it can be argued that this was not the case among the interviewees. Indeed, the literature suggests that the needs of actual organizations can be wildly varying. Scotland (2003) provides evidence of a fringe user of agile software development; a small team developing eTV services for the British Broadcasting Corporation (BBC). The requirement of producing software based solutions quickly to demand is quite different from the original area of Scrum (Schwaber 1995), which was purely the new software product development to competed markets. Thus it might not be valid to assume that all different kinds of organizations could be satisfied with a single set of features, and the generalization of the priority order is not possible, as the mean of a highly volatile variable was used for prioritization. Thus the prioritization results are only relevant to this set of interviewees and their background organizations.

The prioritization process relied on the interviewees' knowledge of the agile software development methods and terminology, as they had to cast their votes based on the textual descriptions of the features and their constituent stories. However, the extent of their knowledge was unknown, and the assumption was that they would have enough knowledge of agile methods to answer the prioritization. Also, only after the prioritization process was concluded, it appeared to the author that the prior version of Agilefant (before and including 1.6.x) might have affected the mindset of the interviewees with its conceptual model that substantially deviates from both Extreme Programming and Scrum (Heikkilä 2008), which further compromised the validity of the prioritization results.

Even though the original invitation to the workshop was given by e-mail and reacted timely by the liaisons, the original 2-week deadline given for the interviewees for prioritization was exceeded by the workshop participants, and a grace period had to be given. Eventually, only a part of workshop participants answered the prioritization questionnaire, along with some respondents from companies that did not participate in the workshop. The small number of respondents (n=11) makes it difficult to do any further analysis on the data.

7.4.4 Validity of the designs of new features, implementations and workarounds

The validity of the designs, implementations and workarounds are a two-fold issue. First the constructions need to be of practical relevance. The practical relevance order of these constructions was evaluated in the prioritization phase. The other is the credibility of these constructions. As the linkage from the implemented features to the existing theory have been shown, they can be considered credible constructions. They have been also preliminarily validated by being employed for the project management at ATMAN. A research project is different from the basic setting of Scrum or Extreme Programming practice of sharing the common goals, in that in a team of researchers there are many specialists, and some of them are possibly pursuing a post-graduate degree, which makes

such teams easily neglect the team aspects of the tool.

The validity of the implemented features cannot in this case be simply determined by an unmodified weak market test (Lukka 2001) as they cannot function without the underlying construction, that is, the backlog management tool. Instead they are considered supplementary components to it. The question would rather be “is there any manager, financially responsible for his unit, who is not willing to use the base construction in his unit, yet is willing to use the modified construction.” It clearly is more difficult to pass than the original weak market test as it requires exceeding a threshold, but also harder to fulfill. Such kind of weak market testing of this research should be conducted in future.

8 Conclusions

The conclusions to the thesis are represented in this chapter. First are represented the answers to research questions (Section 8.1). The contribution of this thesis has been discussed in Section 8.2. Finally, further research proposals are discussed in Section 8.3.

8.1 Answers to research questions

As an answer to research question Q1, "What are the specific practices of managing the daily work of a self-organizing team and of its individual members in the selected agile software development methods?" 10 daily work practices were identified in the literature review. The practices are

1. Visual management of work
2. Assigning responsibles
3. Selection of next task
4. Status tracking
5. Lightweight measurements
6. Status-update meetings
7. Measuring and balancing load
8. Impediment handling
9. Stable teams with dedicated members
10. Maintaining focus and establishing cadence

Evidence for these practices is present in both Extreme Programming and Scrum literature, except for impediment handling which is not at all discussed in Extreme Programming literature. While there seems to be convergence in the literature on practices 1 and 4—9, two competing methods for "assigning responsibles" are present in the literature — "one at a time" and "fill your bag", and no consensus on what should be the implementation order of tasks seems to be present.

The research question Q2, "What tools does the literature suggest for facilitating the implementation of aforementioned practices?" was answered by identifying all tools related to the aforementioned practices. The proposed tools are surprisingly simple: the original Scrum sources advocate spreadsheet files for planning artifacts, and measurement diagrams drawn on whiteboards, the Extreme Programming sources suggest the use of index cards or post-it notes, with an optional special taskboard for presenting the cards visibly, and using a whiteboard to draw transient quality measurement graphs. The use of specialized computer tools is specially discouraged by authors such as Cohn (2005) and

Larman & Vodde (2008), whereas they are recommended by Leffingwell (2007) especially in large-scale agile organizations and for distributed teams.

The research question Q3, "What kind of support does unmodified Agilefant provide for these practices?" was answered by analyzing the features present in Agilefant without the new features that are introduced in this thesis. According to the results, Agilefant deviates from the conceptual model of Scrum and Extreme Programming by necessitating that the iteration backlog is part of the project backlog, thus not allowing work for two projects / products be aggregated into the same iteration backlog. Another difference is its lack of first-class team concept; the team concept in Agilefant is simply a list of individual users; furthermore, only users, not teams, can be assigned artifacts.

Of the identified practices, unmodified Agilefant supports Visual management of work, Assigning responsibles, and Status tracking fully; Selection of next task is complicated if the user is assigned work from several products or projects simultaneously. Of the Measuring and balancing load, Agilefant supports only individual load metrics; the only measurements supported by Agilefant are the estimates for tasks and stories, and optional logging for effort spent. No proper support for Status-update meetings, Impediment handling, Stable teams with dedicated members and Maintaining focus and establishing cadence exists.

The research question Q4, "What features are proposed by the key stakeholders of software development organizations be implemented in agile software development management tool for managing daily work?" was answered through a requirements workshop. The workshop resulted in a set of 69 user stories. Of these stories, duplicates were removed, as well as those stories that were out of scope of daily work management, and the stories that would be naturally supported in any of the identified tools, including spreadsheets, index cards. The remaining stories were grouped under 26 features, which were sent for software development organizations for prioritization.

The research question Q5, "What is the priority order of these features according to the personnel at these software development organizations?" was answered through a multi-stakeholder prioritization process using modified hundred dollar method. The combined results show that the most wished feature is a "Team View" feature, which along with 4 other features — "Consolidated task and story list", "Individual load balancing", "Task deadlines" and "Work queues" constituted nearly 50 % of value appreciated among the features; 80 % of the value was achieved with 14 out of 26 features.

The research question Q6, "How a top-priority subset of these features could be supported in an agile software development management tool while ensuring compatibility with existing agile software development methods?" is answered in detail in Chapter 6; the 14 highest priority features were analyzed and a proposed design, along with a possible workaround to use in a tool if the feature does not exist, was provided. 4 of these 14 highest-priority items were also implemented as a proof of concept in Agilefant 2.0. The results of prioritization indicate two notable deviations from reviewed literature: Monitoring spent effort limits that was included in the 80 % subset is considered by the literature to be an organizational antipattern and thus incompatible with Scrum or Extreme Programming. The emergence of "Task deadlines" as the fourth important feature is also deviation from the prior theory, as none of the reviewed books have discussed of the possibility of having optional deadlines for tasks.

8.2 Contribution

The literature review in this thesis on the daily work management in agile software methods is groundbreaking due to its wideness and its approach, as this kind of analysis of daily work management practices is new. Furthermore, the books by Larman & Vodde (2008) and Leffingwell (2007) have not been subjected to literature review on daily work management practices. The list of identified daily work management practices shall form the basis for analyzing the tool support and team practices in the daily cycles of agile software development methods.

Even though the workshop approach had its weaknesses, new information on the daily work management was gained in it, and the prioritization process shows further interesting results; among them is the need for deadlines for tasks among the top-five features. The elicited list of features, and the prioritization results can be used as a roadmap for analyzing the features of agile tools further.

Analysis for 14 of proposed 26 features were provided, along with possible workarounds that could be employed in a software tools in case these features are not present in that tool. This will help the agile organizations to successfully use a tool that is lacking in functionality. Furthermore, 4 of the proposed 14 features were implemented as a proof of concept in Agilefant, thus bringing Agilefant a more viable tool for the software companies and helping further in validating the ATMAN research on Agilefant as more organizations would be willing to adopt it.

8.3 Future research

As only preliminary validation of the constructions could be provided in this thesis, further research should be made on the effectiveness of the proposed and implemented constructions in real organizations; including a controlled study of the efficiency of the use of specialized information systems for management over the paper and ink, or spreadsheet files. The remaining daily work management features should be implemented in Agilefant to further validate them. Also, as the empirical knowledge on daily work management is scarce, empirical studies on daily work management practices in real work settings should be conducted. In particular, the work of mature agile teams and their interactions with information systems should be observed to acquire a clearer picture of the needed interactions.

References

Auer K & Miller R (2001) Extreme programming applied: playing to win. Addison-Wesley, Boston, MA, USA.

Beck K (1999) Extreme Programming Explained: Embrace Change. Addison-Wesley, Reading, Massachusetts, USA.

Beck K & Andres C (2004) Extreme Programming Explained: Embrace Change (2nd Edition). Addison-Wesley, Boston, MA, USA.

Beck K & Fowler M (2000) Planning extreme programming. Addison-Wesley, Boston, MA, USA.

Beedle M, Devos M, Sharon Y, Schwaber K & Sutherland J (1999) SCRUM: An extension pattern language for hyperproductive software development. Pattern Languages of Program Design 4: 637–651.

Cirillo F (2006) The Pomodoro Technique. World Wide Web electronic publication. Cited Jun 30th 2010 from: http://www.pomodorotechnique.com/resources/cirillo/ThePomodoroTechnique_v1-3.pdf.

Cohn M (2004) User Stories Applied: For Agile Software Development. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

Cohn M (2005) Agile Estimating and Planning. Prentice Hall, Upper Saddle River, NJ, USA.

Dybå T & Dingsøyr T (2008) Empirical studies of agile software development: A systematic review. Information and Software Technology 50(9-10): 833–859.

Fowler M & Highsmith J (2001) The agile manifesto. Software Development 9(8): 28–35.

Hatton S (2007) Early Prioritisation of Goals. Lecture notes in computer science 4802: 235.

Heikkilä V (2008) Tool Support for Development Management in Agile Methods. M.Sc. Thesis, Helsinki Univ Tech, Software Business and Engineering Institute.

Jeffries RE, Anderson A & Hendrickson C (2000) Extreme Programming Installed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Jensen R (1996) Management impact on software cost and schedule. Crosstalk The Journal of Defense Software Engineering July 1996: 6.

Karlsson J (1996) Software requirements prioritizing. Proc. Requirements Engineering, 1996., Proceedings of the Second International Conference on, 110–116.

Karlsson J, Olsson S & Ryan K (1997) Improved practical support for large-scale requirements prioritising. Requirements Engineering 2(1): 51–60.

Karlsson J, Wohlin C & Regnell B (1998) An evaluation of methods for prioritizing software requirements. Information and Software Technology 39(14-15): 939–947.

Karlsson L, Höst M & Regnell B (2006) Evaluating the practical use of different measurement scales in requirements prioritisation. Proc. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, ACM, 335.

Karlsson L, Thelin T, Regnell B, Berander P & Wohlin C (2007) Pair-wise comparisons versus planning game partitioning—experiments on requirements prioritisation techniques. Empirical Software Engineering 12(1): 3–33.

Kasanen E, Lukka K & Siitonen A (1993) The constructive approach in management accounting research. Journal of Management Accounting Research (5): 243–264.

Larman C & Basili V (2003) Iterative and incremental development: A brief history. Computer 36(6): 47–56.

Larman C & Vodde B (2008) Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum. Addison-Wesley, Boston, MA, USA.

Leffingwell D (2007) Scaling Software Agility: Best Practices for Large Enterprises (The Agile Software Development Series). Addison-Wesley.

Leffingwell D (2008) Agile Enterprise Requirements Information Model – Subset for Agile Project Teams. World Wide Web electronic publication. Cited May 25th 2010 from: http://scalingsoftwareagility.wordpress.com/2008/12/17/agile-enterprise-requirements-information-model-

Leffingwell D & Widrig D (2000) Managing software requirements: a unified approach. Addison-Wesley, Reading, Massachusetts, USA.

Leffingwell D & Widrig D (2003) Managing software requirements: a use case approach. Addison-Wesley, Boston, MA, USA.

Lehto I & Rautiainen K (2009) Software development governance challenges of a middle-sized company in agile transition. Proc. SDG '09: Proceedings of the 2009 ICSE Workshop on Software Development Governance, IEEE Computer Society, Washington, DC, USA, 36–39.

Liker J (2004) The Toyota way: 14 management principles from the world's greatest manufacturer. McGraw-Hill Professional, USA.

Liu C & Layland J (1973) Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM (JACM) 20(1): 46–61.

Lukka K (2001) Konstruktiivinen tutkimusote. World Wide Web electronic publication. Cited May 19th 2010 from: http://www.metodix.com/fi/sisallys/01_menetelmat/02_metodiartikkelit/lukka_const_research_app.

McBreen P (2002) Questioning extreme programming. Addison-Wesley.

Rautiainen K, Lassenius C & Sulonen R (2002) 4cc: A framework for managing software product development. Engineering Management Journal 14(2): 27–32.

Rothman J (2007) Manage it! Your guide to modern, pragmatic project management. The Pragmatic Bookshelf, Raleigh, NC, USA.

Royce WW (1970) Managing the development of large software systems. Proc. Proceedings of IEEE Wescon, 26(1): 9.

Saaty T (1990) How to make a decision: the analytic hierarchy process. European Journal of Operational Research 48(1): 9–26.

Schwaber K (1995) Scrum development process. Proc. Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA, 117–134.

Schwaber K (2004) Agile project management with Scrum. Microsoft Press, Redmond, WA, USA.

Schwaber K (2007) The Enterprise and Scrum. Microsoft Press, Redmond, WA, USA.

Schwaber K & Beedle M (2001) Agile Software Development with Scrum. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Scotland K (2003) Agile Planning with a Multi-customer, Multi-project, Multi-discipline Team. Proc. Extreme programming and agile methods: XP/Agile Universe 2003: third XP Agile Universe Conference, New Orleans, LA, USA, August 10-13, Springer, NY, USA, 18.

Sutherland J, Viktorov A, Blount J & Puntikov N (2007) Distributed Scrum: Agile Project Management with Outsourced Development Teams. Proc. Sprague J RH (ed) Proceedings of the 40th Annual Hawaii International Conference on System Sciences, Big Island, Hawaii, January 3-6., ACM, IEEE Computer Society Press, Los Alamitos, CA, USA, 274.

Takeuchi H & Nonaka I (1986) The new new product development game. Harvard Business Review 64(1): 137–146.

Vähäniitty J & Rautiainen K (2008) Towards a conceptual framework and tool support for linking long-term product and business planning with agile software development. Proc. Proceedings of the 1st international workshop on Software development governance, ACM, 25–28.

VersionOne, Inc (2010) State of Agile Development Survey 2009. World Wide Web electronic publication. Cited May 27th 2010 from: http://www.versionone.com/Resources/Whitepapers.asp.

Wiegers K (1999) First things first: prioritizing requirements. Software Development 7(9): 48–53.

Appendix I Proposed new features and their constituent user stories

Proposed features for a tool and the names of constituent stories for these features. The detailed contents for these stories are available in Appendix III.

Feature	Constituent stories
Consolidated task and story list	Consolidated task list Seeing the tasks' context Convergence of tasks according to priority
Filtering tasks	Filtering tasks by effort left Filtering tasks by context
Work queue	Planning and maintaining my own work queue What-to-do-next planning Highlighting queued items in iteration view
"Working on now"-task	What I am working on "right now" Viewing everybody's Now-tasks Automatic effort logging for the Now task
Do not disturb -sign	Do-not-disturb -sign
Impediment handling	Logging impediments Impediment radiation
Making resource stealing visible	Making within-sprint resource stealing visible Ratio between sprint work and other tasks
Load balancing	Balancing work queue load based on calendar appointments & holidays Load shifting according to work queue
Monitoring spent effort limits	Monitoring spent effort limits
Newsfeed	Newsfeed
Notifications	Notifications on task completed Assignment notifications
Over-/undertime slider	Over-/undertime slider Automatic slider adjustment
Periodic tasks	Periodic tasks

Feature	Constituent stories
Pomodoro support	Countdown timer Timer synchronization Pomodoro effort logging Reset circuit
Reminders	Reminders
Stand-up-support	Viewing Done/Impediments/What's next for stand-ups
Strategy-to-action (bottom-up)	Tracing to higher level objectives from daily work Navigating to higher level objectives from daily work Logging effort to higher level contexts
Strategy-to-action (top-down)	Relating tasks to their higher level context Monitoring of epics and reacting to changes Progress monitoring from business perspective
Support for WiP reduction	Minimizing features in progress for a team Story horizon Blindfolds Minimizing tasks in progress
Task deadlines	Task deadlines Sorting by deadline
Task quick-add	Set-aside tasks/queuing interruptions/add task easily Tasklets
Task splitting	Task splitting
Exporting tasks as appointments	Creating calendar appointments from tasks
Team view	Daily work for a team Team's degree of being done with assigned stuff Team workload Spotting potential assignees Team load balancing Working on the most important stuff
Ticket/mail notification integration	Ticket integration
Visibility into calendar	Visibility into calendar Awareness of upcoming appointments

Appendix II Prioritization instructions

Information on your organization

Name of your organization	
Your position	
Is your organization using Agilefant (yes/no) *	
If yes, how many users there are in your organization *	

(= required field)*

Instructions

On the next sheet, you have a total of 1000 credits to distribute among the features that you find would be the most important for your organization. You can divide the credits in any way, by putting a number of credits to the blue cells representing different features. Each feature also contains one or more user stories. These all might be implemented based on the value of the feature, or then only some. However, you cannot bid on single stories, only on complete features.

Feel free to discuss the voting with your colleagues. Also, if you have any questions, feel free to ask them on our forum!

When you are done, send the filled spreadsheet to team@agilefant.org

Appendix III Complete list of features and stories / Prioritization spreadsheet

Story name	As...	I want...	so that...	Cr.
Consolidated task and story list				0
Consolidated task and story list displays the collection of all assigned work from all ongoing iterations for a single user, so that one can see at once where the user is assigned.				
Consolidated task list	as a person with many responsibilities	to have a single consolidated list of tasks I have to do	I don't have to keep peeking at different backlogs in order to find the next thing I have to do	
Seeing the tasks' context	a team member with lots of task related to different stories and products	to see the context path of the task, including story and iteration	I can better identify and prioritize tasks by understanding their context	
Convergence of tasks according to priority	as a person with many responsibilities	to have my tasks listed in sensible priority order	I am assured that I am working on the tasks in the correct order	
Filtering tasks				0
Allows the developer to easily see only a subset of his assigned work that match the given criteria when selecting what kind of tasks he is working on during the day.				
Filtering tasks by effort left	a person with only a small amount of work time left before a meeting/going home/etc.	to see those tasks from the to-do list that best fits the time I have left and pick one of them	I by end of the timeslot I have gotten completed whatever I have started and I need not remember where I left the things when I come back to work tomorrow.	
Filtering tasks by context	a software developer with many duties	to select tasks from a single set only during a day	I won't be overwhelmed by the sheer amount of tasks to do in my backlog	
Work queue				0
A work queue is a view, where the user can plan the work he is going to do in the very near future by dragging and dropping tasks in order.				

Story name	As...	I want...	so that...	Cr.
Planning and maintaining my own work queue	a team member	to see what I have in my work queue, and taking into account the relative importance and urgency of items, to be able to plan in what order to complete them	I can concentrate on finishing the tasks in the designed order; and keep stakeholders updated of my progress and situation, and the order of planned execution.	
What-to-do-next planning	a team member with multiple duties	to be able to have a separate to-do-list (which may include some personal stuff as well) to order & manage my tasks	I can figure out & keep track of what I want to do next without messing the real priorities of tasks	
Highlighting queued items in iteration view	a dev team member	to see in the iteration view on what tasks my team members are working	I ensure that my team is delivering the right things	
”Working on now”-task				0
”Working on now” task is a feature, where the developer can put a task in ”working on now” state, which can be indicated to the other team members and in backlogs in various ways. The feature can be used for automated effort logging too.				
What I am working on ”right now”	a developer	to denote which task I am working on right now	I can catch up quickly after an interruption	
Viewing everybody’s Now-tasks	a developer	a view that tells what my team (or some other group) are working on right now	I can let my team members know what I’m doing at the moment, as well as see what others are doing	
Automatic effort logging for the Now task	somebody who has to log spent effort	a timer to automatically start recording spent effort when I denote that I am now working on some task	I can log my spent effort without unnecessary clicking & typing	
Do not disturb -sign				0
An electronic do not disturb sign, that signals to other team members and users of Agilefant that the user should not be bothered.				

Story name	As...	I want...	so that...	Cr.
Do-not-disturb - sign	a person who is performing an important task that requires concentration	to be able to switch on an electronic do-not-disturb sign for a period of time which is visible to others using Agilefant	so that others can know without leaving their seat - or room, if they are not co-located - that they'll save their questions/comments/other stuff for later since I now need to concentrate	
Impediment handling				0
Impediment handling refers to tracking known impediments in Agilefant, keeping them visible on daily work view and for easy logging of impediments.				
Logging impediments	a developer	to easily log impediments I have run into, be it related to the story, iteration, or the organization	I do not have to specifically recall them during the daily stand-ups, or sprint reviews.	
Impediment radiation	a scrum team member	I want to keep my scrum master aware of my impediments	so he can remove them promptly	
Making resource stealing visible				0
Features for highlighting resource stealing.				
Making within-sprint resource stealing visible	a product / solution owner	the tool to make visible if "my" resources are stolen for other tasks / projects during the sprint	I can get the product out in time	
Ratio between sprint work and other tasks	a scrum team member	to have a clue about how much time I must spend on tasks from the sprint backlog	I can say no to misc tasks when they would put the sprint goal in jeopardy	
Individual load balancing				0
Features for helping a single user to better balance and visualize his daily workload.				

Story name	As...	I want...	so that...	Cr.
Balancing work queue load based on calendar appointments & holidays	a team member	I want to be able to take into account holidays and appointments in my calendar when organizing my work queue	I can see myself if I can finish my assigned work without going overtime, and keep stakeholders updated of my progress and situation. This automatically communicates to the product owner/project manager I am committed / not committed to complete the work assigned to me.	
Load shifting according to work queue	a person with many concurrent backlogs to deal with	my load to shift according to my personal work queue (i.e., the stuff I pick for today is added to my load for today and subtracted elsewhere)	in order to better comprehend my real load which helps me plan ahead what I really should do and what I should dump	
		Monitoring spent effort limits		0
Effort limit monitoring refers to a feature where the product owner is notified as soon as an effort limit is exceeded.				
Monitoring spent effort limits	a product owner / project manager	to be notified if effort spent on task has exceeded the original estimate	I can react as needed	
		Newsfeed		0
Produce a filterable newsfeed of all things a user or a set of users has done lately.				
Newsfeed	a Scrum team member	to have a filterable newsfeed of all things a selected group of people (can be only one as well) have done in Agilefant (task/story state changes, prioritization, estimation, creation, changing descriptions, etc.)	I can get an understanding of what has been happening lately	
		Notifications		0
Notify interested parties when a task is completed automatically, and users of them being assigned to tasks.				

Story name	As...	I want...	so that...	Cr.
Notifications on task completed	a product owner or a project manager	to see when a particular task has been completed and get a notification	so that I can get things further without disturbing myself with constant polling (”has it yet been done? has it yet been done?”)	
Assignment notifications	a employee with many teams and ”hats”	to see, via some kind of newsfeed (or an email) when a task is assigned to me by someone else than me	I stay informed of the tasks that are assigned to me when I am not present	
		Over-/undertime slider		0
Easy tracking of personal over-/undertime on Agilefant.				
Over-/undertime slider	someone who has to keep track of his over-/under time	to have a slider with adjustable scale which I can also move freely	to have light-weight means of keeping track of how much over-/undertime I currently have and satisfy my company’s requirements for tracking work time	
Automatic slider adjustment	someone who has to keep track of his over-/under time	the slider to automatically be adjusted according to the effort I log	I don’t have to manually adjust the slider unless necessary	
		Periodic tasks		0
Periodic task support in Agilefant - e.g. time allocations that repeat regularly.				
Periodic tasks	a team member with multiple duties	to express my ongoing work as ’periodic tasks’ instead of using the baseline load	I remember to attend to the periodic stuff at the intervals I want to as well as to better understand and express my load	
		Pomodoro support		0
Support Pomodoro timers in Agilefant, e.g. a timer that alerts you after a certain period (usually 25 minutes) so that you are able to concentrate only on the task at hand but be notified when the time elapses				
Countdown timer	As somebody practicing the pomodoro technique (pomodorotechnique.com)	I want to have a pomodoro timer in Agilefant	I have my focus support instruments in the same system and others can easily see the time left if they are interested	

Story name	As...	I want...	so that...	Cr.
Timer synchronization	As somebody practicing the pomodoro technique (pomodorotechnique.com)	I want to synchronize my timer with the other people using Agilefant (for example, my team)	so that that our pomodoros end simultaneously and can chat without interrupting each other and don't have to constantly poll each other whether their pomodoros will end soon	
Pomodoro effort logging	As somebody practicing the pomodoro technique (pomodorotechnique.com)	I want my completed pomodoros to be logged as spent effort (possibly prompting me to edit the entry) without unnecessary clicking around	so I don't have to disturb my flow to go clicking around to log effort spent	
Reset circuit	a developer	an "alarm clock" ringing when I have exceeded the time planned for some task + mechanism to resolve the situation (put on hold, split, abandon, ...)	I do not remain stuck (or "frozen", or "adrift") for a long time in a task that was estimated to be short (or planned to be taken quickly)	
		Reminders		0
Reminder notifications for tasks.				
Reminders	a scrum team member	to set, and then get reminders about certain tasks	I would not forget especially smaller tasks that still are important and must be taken care of	
		Stand-up-support		0
Supporting standup meetings by having a specialized view for newly done tasks, and impediments.				
Viewing Done/ Impediments/ What's next for stand-ups	an attendee for a daily stand-up	to have lists of my Done tasks, my current work queue and possible logged impediments shown	so that I don't have to remember them all AND I don't have to use the current 'implemented/ready' -workaround in Agilefant	
		Strategy-to-action (bottom-up)		0

Story name	As...	I want...	so that...	Cr.
Easy traceability from the tasks and leaf stories to the features and epics and their contexts from the daily work view.				
Tracing to higher level objectives from daily work	a product owner	to see the higher-level objectives that a task is addressing	the outcome of the implementation provides a real solution to customer's high-priority need(s) or contributes to our other higher-level goals	
Navigating to higher level objectives from daily work	a developer	to easily navigate the story hierarchy	I can easily get the best possible information about the higher-level objectives that a task is addressing, and that the outcome of the implementation provides a real solution to customer's high-priority need(s) or contributes to our other higher-level goals	
		Strategy-to-action (top-down)		0
Traceability from the top of story hierarchy to the grassroots level on daily basis.				
Relating tasks to their higher level context	a team member	to easily see how the context where a particular task assigned to me belongs, is progressing on the whole	I can experience (and ensure) that my efforts lead to results on higher levels too	
Monitoring of epics and reacting to changes	an owner or a project manager	to see how the higher level stories (epics and features in Leffingwell's terms) I am interested are progressing in terms of lower level stories and tasks	so that I can easily follow what is happening without me having to ask routine questions continuously and I can react earlier to arising issues	
Progress monitoring from biz perspective	a product manager in a far-away product mgmt department	to see how the nuts-and-bolts-level tasks the developers are working on contribute to my Next Big Epic	so that I can view progress in terms that I understand and possibly help the poor developers scope out hard stuff that is not really that important to me	
		Support for WiP reduction		0

Story name	As...	I want...	so that...	Cr.
A feature to specially visualize and limit work/features/tasks in progress				
Minimizing features in progress for a team	As a product owner/business owner	to minimize the minimum marketable features in progress	ensure that we work only on those features we are going to ship at the end of the iteration	
Story horizon	As a product owner	my team to see only a set of minimally-marketable features to help them better concentrate on the stories at hand...	we can use kanbanish scrum	
Blindfolds	As a product owner	the team members to be able to see only the topmost ”started” stories by default and only in the iterations they are responsible for in the daily work	I can ensure that they concentrate on the stories that should be completed first, and not get sidetracked.	
Minimizing tasks in progress	As a team member	to minimize the number of tasks I have concurrently open on, and possibly see a burndown of my open tasks during this iteration	I do not overwhelm myself by starting more tasks than I could possibly finish in reasonable time frame. I can also follow how well I am performing in getting the tasks completed	
		Task deadlines		0
Optional deadlines for tasks, differing from iteration/sprint boundaries				
Task deadlines	an employee	tasks to have optional deadlines, which also affect load calculation	I can more easily see what really has to be done next and my load better reflects the real situation	
Sorting by deadline	a developer working on separate projects/iterations	set deadlines for certain tasks	I do not spend my time working on tasks that are not needed yet, and be able to concentrate on those that really need to be done now.	
		Task quick-add		0
Adding tasks easily on daily work view to ongoing sprints, stories.				

Story name	As...	I want...	so that...	Cr.
Set-aside tasks/queuing interruptions/add task easily	a team member who receives unexpected tasks	to quickly record the any interrupting requests, incoming tasks, or tasks that I spontaneously come up with on some to-do list in the same system where I am managing my duties tasks, and find them easily later	I don't need to interrupt my flow for a long time when a unexpected task arrives but I still can process them properly later	
Tasklets	a scrum team member	to have an easily accessible place holder for very small "tasks" (less than 5 minutes) that are not really tasks	I can still remember to do these tiny tasks in priority order	
Task splitting				0
Split big tasks or resource allocation placeholders into smaller ones easily				
Task splitting	someone who notices that a task is actually a quite big	to be able to split it so that both the resulting task inherits all the attributes of the original task (i.e. assignee, to which story/epic this task contributes, etc.)	I can split big tasks into more manageable units without too much extra clicking around & typing	
Exporting tasks as appointments				0
Exporting selected tasks as appointments to calendaring application and updating them there.				
Creating calendar appointments from tasks	a person who has lots of stuff in his calendar	to have selected tasks appear in my calendar as appointments, and to be able to update them from either the backlog mgmt tool or the calendar	all my important stuff can be managed via a single system	
Team view				0
Display the combined assigned work of a team, along with combined load estimates.				
Daily work for a team	As a product owner/scrum master/team member	to see what all team members are doing currently, and, optionally their pomodoro timers (see pomodorotechnique.com)	so that I know who are doing what and can easily see with whom I can discuss some issue soonest	

Story name	As...	I want...	so that...	Cr.
Team's degree of being done with assigned stuff	a product owner	to see how many unfinished tasks my team members have	I can better estimate how much we can do during the next sprint because of our other commitments	
Team workload	a project manager	to see the work load of the team and assign tasks taking that into account	the work load of the team is balanced and we can ensure that we do things in the right order.	
Spotting potential assignees	a team leader, a product owner or a project manager	to see to whom I should preferably assign or offer the task	assigning tasks for developers to commit to - if they can - would be convenient.	
Team load balancing	a scrum master	to see that my team members load and capacity is balanced	I can ensure that my team has the capability to deliver the planned scope.	
Working on the most important stuff	a product owner	to easily see which tasks my team members are working on even if they have stuff to attend to in different iterations	I am assured that my team is doing the high-priority tasks	
		Ticket/mail notification integration		0
		Expandable integration for pushing notifications onto users' daily work view		
Ticket integration	a developer with lots of support requests that interfere with my development efforts	to have notifications of incoming requests (e.g. via email, bug reporting, instant messaging, etc.) to show up on my task list	I need not interrupt whatever small task I am doing currently to address the issue, but still remember to answer it as soon as it is convenient for me.	
		Visibility into calendar		0
		Reflect calendar information from external calendaring applications on daily work view.		

Story name	As...	I want...	so that...	Cr.
Awareness of upcoming appointments	someone who uses an electronic calendar to organize his time	to see a tiny version of the calendar on the Daily Work page and be able to import selected appointments as tasks	I know what appointments I have upcoming without switching between systems, and don't have to do unnecessary clicking & typing if I want to create tasks for certain appointments	
			Total points	**1000**

Appendix IV Stories not included in prioritization

Already implemented stories

The following stories suggested features already present in the tool, and thus were not included in prioritization.

Story name	As...	I want...	So that...
Adjusting effort estimates	a developer	to be able to refine the tasks' effort estimates as work progresses.	everybody can see the progress and can reschedule the backlog to be realistic as needed
Assigning tasks	a person with many responsibilities	that I can be assigned tasks without interruption, and vice versa.	I can concentrate on one task for a moment, and only after that is completed, or my time slice has been used, I can start working on it, delegate it or bounce it back
Degree of sprint completion	a product owner	to see how many unfinished tasks my team members have in the current sprint.	I can better estimate how much we can do during the next sprint
Delegating tasks	a scrum team member	to be able to delegate tasks assigned to me/volunteered by me to somebody else.	so I will not spent too much time with the task I cannot proceed or I'm stuck with
Logging spent effort	a scrum team member	to easily log the effort spent for a task I'm working on.	logging effort spent would be effortless
Logging spent effort	a scrum team member	to easily log the effort spent on the relevant context, i.e. to the story, project, product, etc.	logging effort spent on whatever level of granularity appropriate to the context is easy
Managing personal to-dos	a daily work management tool user	to log whatever personal tasks I would have to worry about during my work as they pop up, and forget them for time being.	they do not trouble me until needed.
My tasks	a team member	to see the tasks within a sprint in priority order and possibly reorder them.	I can ensure that I am working on those items that are crucial for the successful execution of the sprint
Prioritizing tasks within a sprint	a Scrum team member	to prioritize my doings myself within a sprint.	all backlog items are finished during the sprint.
private tasks	a scrum team member	to have just one place where to store and manage all my tasks, even tasks that do not belong the product development.	as a whole I can manage better and get the feeling of better work control and getting things done

Story name	As...	I want...	So that...
Reacting to overwhelming load	a developer	to realize as early as possible that I cannot cope with the tasks like I have planned. I can also then try to reorganize these tasks myself, or if that does not succeed, I can go to discuss the issue with project manager/team leader.	I myself, the product owner / project manager or the team leader are able to plan the work in advance accurately, and we can also decide easier what items we shall complete when the situation changes.
Realistic tasks	a scrum team member	to see that the work left in my tasks and assignments is realistic to get done in the time planned.	I am able to do them in the assigned time without screwing up the whole plan, and being embarrassed
Real-time burndown visibility	a scrum master	to have instant visibility to the sprint burndown.	I can react promptly if incoming misc tasks are hurting our velocity and becoming a threat to the sprint goals
Updating estimates	a scrum team member	to change the effort left estimates for a task I am working on now.	re-estimating effort left is easier

Duplicates and stories that were out of scope

The following stories were either duplicates of existing stories, or out of scope of daily work management, or tool support.

Story name	As...	I want...	So that...
Commitment management	an owner and team leader	to efficiently and handily (without heavy protocol) agree with developers on how we can achieve our goals and ensure that they are committed to the tasks.	developers explicitly commit themselves to the items, or say that they are not able to do them, to avoid false optimism and tasks that are not done.
Desirable workplace	an R&D manager	the system to support a culture where people can be efficient without feeling overloaded	Our company maintains a good spirit, good mental health and a reputation as a desirable workplace.
Flexible work hours	a scrum team member	to have a flexible work time policy	I can take care of personal tasks during office hours and work late to compensate
Historical time-break-down	a scrum team member	to compare the actual efforts spent of done tasks vs. the original estimates	the actual efforts of tasks can be compared to the planned ones to improve planning

Story name	As...	I want...	So that...
Historical time-break-down	the portfolio owner	see what I (or anyone) have been spending time to on a given day / week / month	to see where the time is actually spent so I can prune & put on hold the stuff that is not that vital to our company
Investment effort tracking	a product owner	to track the investment (in terms of effort) and changes to it	I can approve or discuss the scope (or even terminate it, if no solution is found)
Lunch-time management	a user with dietary requirements	to have my fixed lunch/coffee breaks on the work queue/calendar	I can more effectively manage my daily work and even finish the tasks I have been working on, before having lunch, and still have them timely, to avoid any health complications etc.
Making jeopardized focus visible	a scrum team member	the tool to make deviations to the principles of scrum visible to promote a culture that respects my right to focus according to scrum	I can use most of my time to work on the sprint backlog and not misc stuff that is pushed to me from various sources
Notifications on task completed / owner	a scrum team member	to get notification about, for example, completed tasks done by others (can be notifications also other type of event like task cancelled, delayed, etc.)	I can continue my other work without unnecessary polling for status
Priority guidance	a scrum team member	to get guidance when in doubt about priorities	I can continue without doubting whether I'm working on the right task
Relating a task to team's efforts	a team member	to see how a particular task assigned to me is related to the work assigned to other team members in the sprint	I can experience (and ensure) that my efforts lead to results on higher levels too
Splitting epics	an owner or a project manager	to be able to split epics into features and further as stories so that they retain the memory of their origins	we can work on more manageable units of work
Working hours report	a resource manager	to get full report of all the work that a resource has done each month	I can ensure that working hours are within the agreed range (not too much overtime, ...)

Appendix V Companies and individuals answering the prioritization

The companies that answered the prioritization questionnaire

Company ID	*Company size*[*]	*Number of answers*	*Agilefant in use?*	*Number of users*
C1	S	1	Yes	15
C2	L	1	Yes	N/A
C3	L	2	Yes	8
C4	XS	1	Yes	2
C5	L	1	No	-
C6	L	1	Yes	8
C7	M	1	No	-
C8	S	1	Yes	N/A
C9	S	2	Yes	28

[*] *Number of employees in the organization: XS: –10 persons, S: 10–50, M: 50–250, L: 250– persons*

The individual interviewees

Interviewee Identifier	*Company*	*Role*
I1	C1	Director of Software Development
I2	C2	Technology Manager
I3	C3	Technical Product Manager
I4	C4	Software Engineer / Web Developer
I5	C3	Agile Coach
I6	C5	Project Management Officer
I7	C6	Scrum Master & Area Product Owner
I8	C7	Quality Manager
I9	C8	Quality Manager
I10	C9	Consultant
I11	C9	Chief Executive Officer

Appendix VI Individual voting results

Feature name	*Interviewee*										
	I1	I2	I3	I4	I5	I6	I7	I8	I9	I10	I11
Consolidated task and story list	100	100	70	80	150			150	158	15	80
Filtering tasks			60	50	25				60	50	10
Work queue	100		40	50	125	50		150	22	100	90
Working on now-task			10	60	15	50			15	50	60
Do not disturb -sign			10	50		100			15		10
Impediment handling		50	20	30	25	50			22	30	80
Making resource stealing visible			10	30	10	20			22	30	10
Individual load balancing	100		150	60	25		100	50	60	200	80
Monitoring spent effort limits			200	20	10				37		
Newsfeed			15	40					22	30	
Notifications	200		10	70	75				22		30
Over-/undertime slider			10	10					7		40
Periodic tasks			40	40		30		50	15	30	40
Pomodoro support			30						7		60
Reminders			30	40	5				15		10
Stand-up-support	100		15	10	15	30	100		22	10	40
Strategy-to-action (bottom-up)			50	60	95	150			37	10	15
Strategy-to-action (top-down)			50	70	100	50		150	37	10	25
Support for WiP reduction		100	15	50	25				60	20	
Task deadlines	100	150	10	40		50		150	60	175	90
Task quick-add			15	10	75	30		50	15	10	50
Task splitting	100	100	30	50	75	50		50	60	20	30
Exporting tasks as appointments			20	5		100			15	10	50
Team view	200	500	50	40	150	100	800	150	158	15	10
Ticket/mail notification integration			20	30		40			22	10	10
Visibility into calendar			20	5		100		50	15	175	80