HELSINKI UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SOFTWARE BUSINESS AND ENGINEERING INSTITUTE

Ville Heikkilä

Tool Support for Development Management in Agile Methods

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology

Espoo, December 1, 2008

Supervisor: Tomi Männistö, DrSc(Tech)
Instructor: Jarno Vähäniitty, LicSc(Tech)

HELSINKI UNIVERSITY OF TECHNOLOGY Department of Computer Science and Engineering		ABSTRACT OF MASTER'S THESIS	
Author	Ville Heikkilä	Date	December 1, 2008
		Pages	vii + 118
Title of thesis	Tool Support for Development Management in Agile Methods		
Professorship	Software engineering	Code	T-76
Supervisor	Tomi Männistö, DrSc(Tech)		
Instructor	Jarno Vähäniitty, LicSc(Tech)		

The goal of this research was to find out what requirements the most prominent agile methods and the case company have for a development management tool and to determine if an existing tool that sufficiently fills the requirements can be found. Many tools have been developed to facilitate the management of work and requirements in agile software development projects since the mainstream breakthrough of agile software development methods following Agile Manifesto in 2001. However, it is not known whether the tools that are publicly available actually meet the requirements that the agile methods have for managing work and requirements. Pre-existing research on the topic has concentrated on generating a set of requirements and building a new tool which is based on those requirements.

Based on an analysis of popular articles on agile software development, Extreme Programming and Scrum were found to be the most prominent methods. Practitioner guidebooks on each method were selected and then reviewed for requirements. Interviews were performed in the case company and the results were analyzed to identify the requirements. The resulting requirements belonged to two groups: conceptual requirements, which describe what kind of information needs to be saved in the tool, and functional requirements, which describe what kind of functionality the tool must have. The open source tool Agilefant and the commercial tools Mingle, Rally Enterprise Edition and ScrumWorks Pro were selected. Each tool was then separately reviewed against the requirements.

Agilefant had severe deficiencies concerning conceptual requirements in both requirement sets. These conceptual deficiencies also resulted in severe deficiencies in functional requirements. Mingle filled well the conceptual requirements of both sets. It also successfully met the functional requirements from the books. However, Mingle failed to fulfill the important functional requirements from the case company that concerned work-hour recording and reporting. Rally Enterprise Edition adequately fulfilled the conceptual and functional requirements from the book review, but the tool had severe conceptual and functional deficiencies regarding the work-hour recording and reporting requirements from the case company. ScrumWorks Pro had many deficiencies concerning the conceptual and functional requirements from the book review and the case company.

The results of this research show that although tools for agile software development by-the-book exist, realworld software development companies have additional requirements which may not be fulfilled by the current tools. In this case, the majority of the additional requirements concerned work-hour recording and reporting. Tool developers should take into account the real world needs of software development organizations when creating tools. In order to enable the development of tools that better match the realworld requirements, further research into other agile software development organizations should be conducted to identify what kind of additional requirements the organizations have.

Keywords

Agile software development, project management, software development tool support

TEKNILLINEN KORKEAKOULU Tietotekniikan osasto			DIPLOMITYÖN TIIVISTELMÄ
Tekijä	Ville Heikkilä	Päiväys	01.12.2008
		Sivumäärä	vii + 118
Työn nimi	Ketterän ohjelmistokehityksen hallinnan työkalutuki		
Professuuri	Ohjelmistotuotanto	Koodi	T-76
Työn valvoja	Tomi Männistö, TkT		
Työn ohjaaja	Jarno Vähäniitty, TkL		

Tämän diplomityön tavoite oli selvittää vaatimukset, jotka tärkeimmät ketterät ohjelmistokehitysmenetelmät ja tapausyritys asettavat työkaluille, joilla hallitaan ohjelmistokehitystyötä. Näiden vaatimusten avulla pyrittiin selvittämään täyttääkö jokin tällä hetkellä saatavilla oleva työkalu vaatimukset riittävän hyvin. Ketterien menetelmien tutkimus ja käyttö on yleistynyt nopeasti sen jälkeen, kun Agile Manifesto julkaistiin vuonna 2001, ja lukuisia työkaluja on kehitetty työn ja vaatimusten hallintaan ketterässä ohjelmistokehityksessä. Tästä huolimatta tutkimustietoa siitä, miten hyvin olemassaolevat työkalut täyttävät ketterien menetelmien tarpeet, on vähän. Aiheeseen liittyvä tieteellinen työ on keskittynyt uusien työkalujen kehitykseen ja uusien vaatimusten etsintään.

Ketterää ohjelmistokehitystä käsittelevien populaariartikkelien analyysin perusteella Extreme Programming ja Scrum olivat kaikkein tärkeimmät ketterät menetelmät. Menetelmiin liittyvät vaatimukset hankittiin menetelmäohjekirjallisuudesta. Tapausyrityksen vaatimukset hankittiin haastattelujen avulla. Tuloksena saadut vaatimukset voitiin jakaa kahteen erityyppiseen ryhmään. Ensimmäinen ryhmä sisälsi konseptuaalisia vaatimuksia, jotka kertovat millaista tietoa työkaluun on voitava tallentaa. Toinen ryhmä sisälsi toiminnallisia vaatimuksia, jotka kertovat mitä työkalulla pitää olla mahdollista tehdä. Avoimen lähdekoodin työkalu Agilefant ja kaupalliset työkalut Mingle, Rally Enteprise Edition ja ScrumWorks Pro valittiin katselmointia varten. Tämän jälkeen työkalut katselmoitiin yksi kerrallaan.

Agilefant-työkalussa oli vakavia puutteita konseptuaalisiin vaatimuksiin liittyen. Näistä puutteista seurasi myös vakavia toiminnallisia puutteita. Mingle täytti konseptuaaliset vaatimukset ja menetelmäkirjallisuuden toiminnalliset vaatimukset erittäin hyvin. Se ei kuitenkaan täyttänyt tapausyrityksen tuntiseurantaan ja -raportointiin liittyviä tärkeitä vaatimuksia. Rally Enteprise Edition täytti menetelmäkirjallisuuden konseptuaaliset ja tominnalliset vaatimukset hyvin. Työkalussa oli kuitenkin vakavia konseptuaalisia ja toiminnallisia puutteita koskien tapausyrityksen tuntiseurantaan ja -raportointiin liittyviä vaatimuksia. ScrumWorks Pro:ssa oli vakavia puutteita liittyen menetelmäkirjallisuudesta ja tapausyrityksestä hankittuihin konseptuaalisiin ja toiminnallisiin vaatimuksiin.

Tämän diplomityön tulokset osoittavat, että ominaisuuksiltaan riittäviä työkaluja menetelmäkirjallisuuden mukaisen ketterän ohjelmistokehityksen hallintaan on olemassa. Ohjelmistoyrityksillä on kuitenkin lisävaatimuksia, joita nykyiset työkalut eivät välttämättä täytä. Tässä tutkimuksessa nämä lisävaatimukset käsittelivät pääosin tuntiseurantaa ja -raportointia. Työkaluja kehittävien tahojen tulisikin ottaa paremmin huomioon ketterää ohjelmistokehitystä tekevien yritysten todelliset tarpeet. Jatkotutkimuksen tulisi selvittää, millaisia vaatimuksia muilla ketterää ohjelmistokehitystä tekevillä yrityksillä on ohjelmistokehityksen hallintatyökalulle.

Avainsanat

Ketterä ohjelmistokehitys, projektinhallinta, ohjelmistokehityksen työkalutuki

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter contains an introduction to the thesis. Section 1.1 describes the motivation for conducting the research that is described in this thesis. Section 1.2 presents the high-level research question of the research. It also describes the three lower-level research questions that are used in answering the high-level question. The common aspects of agile methods and further sub-research questions that were used in the research are also presented in this section.

1.1 Motivation

Following the creation of the Manifesto for Agile Software Development [Beck et al., 2008], agile software development methods have accumulated interest, both within the academic community and in the software industry. One aspect that is common to many agile methods is the management of work as small pieces of work such as stories in Extreme Programming (XP) and backlog items in Scrum. Agile methods are iterative and incremental: work is planned and done in short iterations which build upon the previous iterations. In more traditional software process, requirements will pass through multiple steps before they are implemented in software source code [Schach, 2001, p. 66]. The trail from elicitation to implementation is shorter in agile methods and the requirements are easily altered or reprioritized during implementation. Another common aspect in agile methods is the empowerment of the software development team. Each development team member is expected to select the next thing to do without explicit instructions from a manager. This practice also differs from the traditional way to manage work.

Traditional project management tools such as PERT charts, Gantt charts and work breakdown structure charts could be used with agile methods. Unlike in traditional projects, the expectation in agile software development is that only a small part of the requirements is known when the project starts and that new requirements will constantly emerge during development. This makes it unfeasible to follow the progress of the development work with these traditional tools. Recreating the traditional charts whenever a new requirement emerges would take resources out of development work. Therefore new lightweight tools for management of work are needed.

The existing research on the subject has concentrated on tools for Extreme Programming [Pinna et al., 2003a, Melis et al., 2004, Angioni et al., 2006, Kääriäinen et al., 2004, 2003] or more specifically managing user stories in a XP context [Breitman and do Prado Leite, 2002, Rees, 2002]. Furthermore, the goal of the existing research has mainly been to create new tools for managing agile software development which are augmented with some kind of additional functionality which, based on his own experiences, the author of the research has deemed necessary. None of the found related research literature has used an external real world agile software development organization as a source of requirements. None of the existing research systematically reviews the existing tools for managing agile software development. While the results from the related literature can be used to validate the results of this research, a more general approach toward the selection of the included agile methods and a case organization study are required in order to elicit the requirements for reviewing the existing tools for managing agile software development. See Appendix A for a more detailed description of research literature that is related to managing agile software development.

The exact requirements for a development management tool depend on many factors, such as which agile method is used and how much the method has been customized. However, the requirements from a few most prominent agile methods, combined with requirements from a real world agile development organization, will provide an idea of the current status of the available tool support for agile software development management. The results from this thesis will allow any interested party to identify tools for closer investigation according to the organization's special needs.

1.2 Research goal and questions

The goal of this thesis is to find out if the publicly available tools for managing the daily work of an agile software development organization have sufficient features for that purpose or whether they lack some functionality that prevents their efficient use. The following high-level research problem is based on this research goal:

Does any tool available for free evaluation have sufficient functionality for managing the daily work of an agile software development organization?

The search for an answer to the high-level research problem is accomplished by using the three research questions which are presented in the list below. The majority of the research for answers to questions one and two is independently performed. However, the answers to the first question are used in the formulation of the research process for answering the second question. The answers to the first two questions are subsequently used in the research for answers to the third question, where the tools available for evaluation are compared to the identified requirements.

1 What are the requirements of the most prominent agile methods for a tool that manages the daily work of an agile software development team, according to agile software development methodology textbooks?

2 What are the requirements of a real world agile software development organization for a tool that manages the daily work of an agile software development team?

3 Does any tool which is available for evaluation fill those requirements sufficiently well?

To help elicit of the most relevant requirements to answer the questions one and two, aspects that are characteristic of agile software development are used to formulate more specific research sub-questions. These aspects are briefly introduced in the thesis motivation in Section 1.1 and are summarized as follows.

- Work is conducted and planned in iterations which build upon previous iterations.
- Shorter time horizons are planned in greater detail and longer time horizons are planned in less detail.
- Work is split into small pieces which are individually managed.
- The development team autonomously selects the execution order of work during a development iteration.
- The progress of work is monitored with lightweight indicators.

The following more specific research sub-questions are used to answer research questions one and two. All questions in the list are used to answer both questions.

(a) What can be identified as "pieces of work"? That is, something that needs to be done.

(b) What information do the "pieces of work" contain?

(c) How is "the work that needs to be done" recorded?

(d) How is "the work that needs to be done" managed?

(e) How is "the next thing that a developer should do" selected?

(f) What planning time horizons are used?

(g) What information do these time horizons contain?

(h) How is the progress of work monitored during the development?

Chapter 2

Research process

This chapter provides a detailed description of the research process of this thesis. Figure 2.1 contains a flowchart that shows the overall structure of the research process. The research was started by performing a mini-research to determine the scope of the research on agile methods. At the time of writing, many agile methods were actively discussed and researched. The selection process for the agile methods for inclusion in this research was not trivial. A high number of practitioner guidebooks have also been written on agile software development methods and it would have been impractical to read them all.

The first step in the mini-research was the selection of the most prominent agile methods. The second step was the selection of a set of books which cover the most prominent agile methods. As the first research goal of this thesis is to determine the requirements for a tool that supports by-the-book agile software development, conference papers and journal articles were excluded from the search. In the third step of the mini-research two of the four initially selected most prominent agile methods were excluded from the research. The scoping mini-research is described in detail in Section 2.1.

The first research question in this thesis is, “What requirements do the most prominent agile methods have for a tool that manages the daily work of agile software development team, according to agile software development methodology textbooks?” This research question was answered by conducting a book review of the selected authoritative practitioner guidebooks for the agile methods that were included in the research. Section 2.2 describes the book review process.

The second research question of this thesis is, “What requirements does a real world agile software development organization have for a tool that manages the daily work of a software development team?” Answers to this research question were searched by conducting a case study in a real world agile software development organization. The selection of organization is described in detail in Section 2.3 and the selection of interviewees is described in detail in Section 2.4. Section 2.5 describes the software development organization requirements elicitation process.

The third research question of this thesis is, “Does any tool available for evaluation sufficiently fulfill those requirements?” The requirements which have been referenced here are the requirements elicited in the book review process and case company study. The answer to this research question was investigated by reviewing a selection of open-source and commercial agile software development management tools. The tool selection process is described in detail in Section 2.6. As a result, one open source tool and three commercial closed source tools were selected for review. Detailed introductions to the selected tools can be found in Chapter 6. Section 2.7 describes the process used in reviewing the tools.

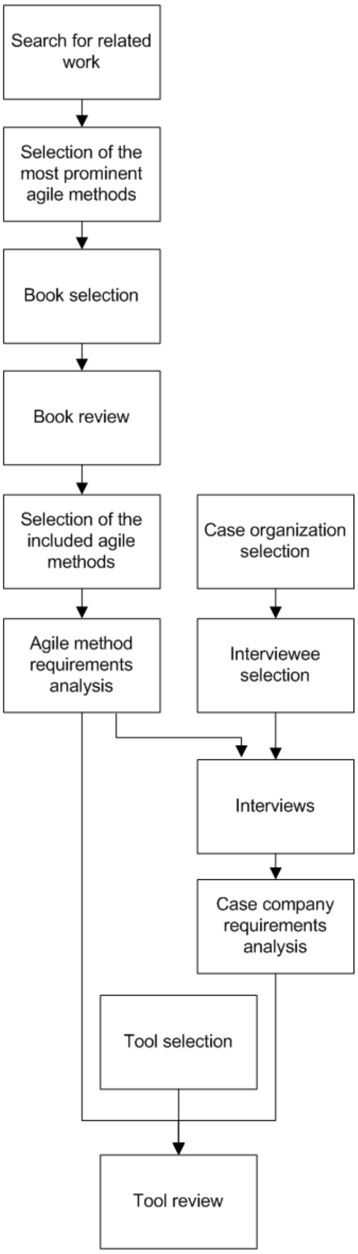

Search for related work
Selection of the most prominent agile methods
Book selection
Book review
Selection of the included agile methods
Case organization selection
Agile method requirements analysis
Interviewee selection
Interviews
Case company requirements analysis
Tool selection
Tool review

Figure 2.1: Overall structure of the research

2.1 Selection of the included agile methods

The selection of the agile methods that were included in this research is described in this section. The search for the most prominent agile methods was started by searching for Web pages which contained information on agile software development. The two following sources were identified in this search:

Agile Alliance is an umbrella organization of different entities that are interested in agile software development. The two main sources selected for the preliminary agile software development method search from the Agile Alliance website were articles that were linked from the website and the Agile Narratives Program, which aims to gather personal stories which are relevant to agile software development.

InfoQ is an online community website that covers several topics including agile software development. Books about Agile and Articles about Agile sections were included in the research.

Articles from these sources were browsed and an article was included in the review if its title was somehow related to agile project or requirements management. The publication year of each included article was recorded to form a preliminary image of the evolution of agile methodology. No strict search terms, selection criteria, or a search protocol were used in this phase, as the goal was to form an idea of the prominence of the different agile methods instead of extracting statistically significant quantitative information.

The selected articles were read and any mentions of agile methods were recorded along with the article's year of publication. Any references to books on agile software development and to computerized tools for agile software development work management were also recorded.

34 articles were selected for closer inspection. Only one of these articles was excluded for not containing relevant material. No relevant material was found on the Agile Narratives website.

Two criterion were used in the agile method selection: The total number of articles which referenced the method and the time span in which the articles which mentioned the method were published. The latter criteria was expected to reflect the long-term importance of the agile method in question; for example, whether the method received attention for a few years and then faded into obscurity or if it managed to remain relevant until the present day.

Amazon.com Web-page was used as the source for books that cover the selected methods. The names of the methods were used as search strings and the few first pages of search results were browsed.

Several books were selected to be read during this stage. After reading the books, a decision was made to exclude two of the four included methods. The first excluded method was Lean Software Development. It concentrates on providing guidance for making a software development processes more lean and does not offer a practical software development process. The second excluded method was excluded Feature Driven Development (FDD). The method is notably different from the two included methods and a separate tool review would have been required in order to evaluate the tools' compliance with FDD. The inclusion of the method would have provide little value over the results from the two methods that ultimately were included.

As a result of these steps, Scrum and Extreme Programming were selected for detailed analysis in this research.

2.2 Agile software development book requirements extraction

This section presents the protocol which was used to extract the requirements of the book review. The goal of the book review was to obtain information on how the selected agile methods work as described by the authors of the methods. The research sub-questions which were presented in Section 1.2 were used to guide the review of the selected books. However, the recorded results were not limited to contain only answers to the questions.

The results from the selected books were recorded in a mind map. The table of contents of each selected book was read and the chapters which potentially contained answers to the research sub-questions were noted. The chapters were then read and any answers to the sub-questions were recorded in a mind map with a reference to the page which contained the answer.

After the book was read a the mind map was used to develop a concept map of the agile method as it was described in the book. If concepts or relations between them were unclear, then the book was referenced for clarification.

The concept maps were analyzed to identify the discrete types of concepts that existed in each map. The identification of the different concept types was based on the differences in the function of the concepts. The four identified concept types serve different purposes in the methods. Each concept in each map was then mapped into the set of identified concepts (see Section 4.1).

There were two notable decisions which were related to the concept types. An “actor” concept type, which could contain, for example, person or team concepts, was not included in the concept types. Scrum and XP were considered single team-centric methods in the scope of this thesis. Only one concept which required a team member as a value was identified in both methods. Other information concerning actors is implied by the context of the project and does not need to be explicitly recorded in the methods. This makes separate concept type to contain actor type concepts unnecessary, as the information can be recorded equally well with the other concept types. The second notable decision was to include the container of work concept type as a separate concept type. The identified containers of work could have been included in the conceptual models as parts of the related time horizons. However, there are notable differences between the relationships among the containers of work and the relationships between the time horizons to which the containers belong. The separate container of work concept type makes illustrating these differences much simpler.

One new concept map for both of the two selected methods was created by merging the previous two concept maps that were created based on the books. By using the new concept map, the two original concept maps, the mind map, and the source books, an explanation was written for each concept and for the relationships between concepts. These results can be found in Chapter 4.

In addition to the individual concept maps of the two methods, a map which combines the concepts from the two methods was created. Many real world users of Scrum and XP have customized the methods to some extent [Salo and Abrahamsson, 2008] and the comparison to the exact conceptual requirements from the two methods is less valuable than a comparison to the combined conceptual requirements. Any tool that fills all combined conceptual requirements also fills the individual conceptual requirements from both methods.

The combined conceptual model was not just a mechanical combination of the concepts from the two methods. Connections between concepts were added in cases where it seemed appropriate, considering the consistency of the model as a whole. For example, a connection between a project and release has been added (see Figure 4.4), as such connection exists in Extreme Programming between a release and its iterations (see Figure 4.3).

The concepts and the relationships in the conceptual models were not prioritized or refined to create the requirements for a tool. The books which were used as sources give little indication of the relative importance between the different concepts which exist in the agile methods. The concepts in the combined conceptual model were directly used as data model requirements for the tools since they indicate what type of information needs to be saved within a tool. The inclusion of all identified conceptual and functional requirements in the tool reviews facilitated the formation of informed opinions about the sufficiency of the tools.

Requirements for selection of work and monitoring progress were also elicited from the books. These requirements were created by first extracting answers to the related questions from the books and then recording them in a mind map. Next, the requirements were created according to the mind map. A combination of the requirements from the both methods was also created in a similar manner.

2.3 Selection of the case company

The software development methods and processes in software development organizations vary widely according to many different factors, such as the size of the organization, the type of the software developed and time-to-market requirements. Conducting a large-scale inspection of requirements in many different software development organizations is out of the scope of this thesis. However, taking the requirements from a single software development organization provides some insight into the kinds of requirements that real world software development organizations have.

Large software development organizations are usually divided into many subdivisions and each might have its own software development process. In addition, a large organizational structure creates a set of separate issues which are mostly related to communication and are out of the scope of this thesis. On the other hand, very small organizations can do without much process at all and the requirements from a such organization would not be very interesting considering the goals of this thesis. Medium-sized organizations are the best candidates because they may have defined processes for software development in place, yet they are still small enough that it is possible to obtain an understanding of the organization's processes by conducting a set of interviews.

For the purposes of this thesis, a medium-sized software development company that was willing to participate was selected. Because the company was looking for a new tool to manage the daily work of its employees, it was motivated to contribute to this thesis since it stood to benefit from the results. The company offered both software products and services and employed agile software development at least at some level within their software development. As an added benefit, the company's official language is Finnish, which is also the native language of the interviewees and the author. Therefore, there were no language barriers to be overcome.

2.4 Selection of the interviewees in the case company

Three different roles that have to do with software development were selected for interviews within the case company. The first selected role was senior software developer. A senior software developer spends most of his or her time developing software. In addition to writing program code, the duties of this position include tasks such as testing, low-level design and participation in the architectural design of the software. A senior software developer works daily with the requirements or features that need to be developed into the software. Considering the research questions of this thesis, the needs for a tool that manages a senior software developer's daily work are very important.

The position of software development team leader is the second selected role. A software development team leader also participates in software development tasks, but some of his or her time is spent managing software development. This includes clarifying requirements for the developers, analyzing and prioritizing requirements, monitoring progress of work and communicating the project's status to stakeholders. Because the software development team leader is responsible for managing the requirements and monitoring progress of the software development, his or her needs for a tool to manage the daily work must also be considered.

The position of project manager was the third selected role. A project manager is responsible for planning and managing projects, which requires an awareness of the status of each project that he or she is managing. During the planning of a project, a project manager also plans at such level of granularity, that the team leaders and developers can begin to develop the project according to the plan. This means that a project manager needs to be aware of the daily work of the developers. Therefore, a project manager has his or hers specific needs for a tool for managing daily work.

One person per each role was identified at the company and asked to be interviewed. Each agreed to be interviewed. Each interviewee was contributing to several different customer projects. The senior software developer and the team leader were also participating in documentation and bug-fixing and the project manager was participating in training activities.

2.5 Case company requirements elicitation

The second research question (see Section 1.2) of this thesis was answered by conducting a series of interviews in a real world agile software development organization. This section describes how the interviews were conducted and analyzed. Section 2.5.1 describes the practical arrangements of the interviews. Section 2.5.2 describes how the results from the interviews were analyzed.

2.5.1 Interview methodology

Two different sets of interview questions were created for the interviews. The first set of questions targeted interviewees whose jobs mainly constitute software development activities such as coding. The second set of questions was aimed at interviewees with more of a management type of job. The questions were created created according to the research sub-questions that were presented in Section 1.2 and based on the results from the book review.

The interview was constructed in a semi-structured fashion. The questions were arranged in a tree-like structure in which general questions were placed closest to the trunk and more specific questions formed the branches. If the answer to a question was expected to be either yes or no, two alternative branches were formed to account for follow-up questions. The questions were formulated according to the author's knowledge on general software engineering and on project management practices. In addition to the questions, an introduction for the interviewees was produced. For the most part this introduction contained statements that were intended to make the interviewee as comfortable as possible with the interview situation. The original introduction and the two sets of interview questions can be found in Appendix B. All questions were in Finnish.

Interviews were conducted in one of the company's meeting rooms. In addition to the author and the interviewee, one other person participated in the interviews. His purpose was to take notes during the interview to facilitate the analysis of the interviews. He was also allowed to make questions to the interviewees when it appeared that the author had missed an opportunity to ask something of importance. The interviews were also recorded to facilitate the analysis.

When the introduction was read to the first interviewee, it became apparent that, instead of putting the participant at ease, he became less relaxed. Therefore, only the introduction's most important elements regarding privacy and confidentiality were read during the following two interviews. The interviews were conducted in semi-structured fashion. The interviews were started by asking the first question in the trunk of the question tree structure. After the interviewee had answered the question, follow-up questions were asked. The follow-up questions were either branches of the previous question or free-form questions based on the previous answer. The follow-up questions continued until the whole branch was exhausted, whereupon the next question in the trunk of the tree was asked. If the interviewee had already answered a question that came later in the question tree, then the question was skipped. The interview continued until all questions in the structure were exhausted and no more relevant free-form questions came to mind.

2.5.2 Interview analysis

The analysis of the interviews was conducted after all three interviews had been performed. The analysis was performed by looking at the notes that the second interviewer had taken during the interview and by listening the recordings of the interview. The analysis results of each of the interviews were recorded in separate mind maps. Instead of the tree formation which organized the questions, the analysis mind maps were grouped through an association technique. Data elicited from the interviews was recorded in the mind map and grouped and organized in an appropriate manner. If later data revealed that the previous organization was inappropriate, then the data was reorganized. All three interviews were separately analyzed in this fashion.

Based on the three analysis mind maps, a concept map of the concepts that are involved in managing the daily work of the software developers was created. Most of the concepts in this map were mapped to the concept types which were identified in the agile software development book review (see Section 4.1). However, the concept map included one concept type that could not be mapped to the four previously identified types. As a result, one of the original four concepts was broadened to include the new concept type.

Based on the three original mind maps, the concept map, and the interview recordings, a new mind map was constructed which consisted of the requirements for a tool based on the interviews. The new mind map was primarily constructed to enable the voting and the verification described in the next paragraph, as the conceptual map and the three original mind maps would have been too complicated to explain and employ in a such event. The requirements were divided into groups in the mind map based on their similarity. The individual requirements were also annotated to aid the verification and voting. The annotations were intended to show the current state of the requirements considering the tools that the company had in use when the interviews were conducted.

A two-hour meeting was arranged with the company for the prioritization and verification of the requirements. The three interviewees participated in the meeting. In addition, a wide variety of employees who fill different roles within the company were present. These included customer support personnel, programmers, and managers. The prioritization of the requirements was performed by giving each participant a set of cards with numbers from 1 to 4 printed on them. Each number represented a priority class. Number 1 meant that the requirement was critical, 2 meant that the requirement was useful, 3 meant that the requirement was not needed and 4 meant that the voter could or would not comment. These classes and the annotations used in the requirements mind map were explained to the participants before the voting commenced. During the voting, each requirement was explained by the author. If the participants agreed that the author had correctly understood the requirement, then the voting commenced. If the participants agreed that a requirement needed to be altered, then it was discussed by the participants until an acceptable alternative could be formulated.

The voting was conducted in the following way. All company employees that were present individually selected a priority card without showing his or her selection to others. When all voters had made their selection, they simultaneously revealed their votes. If the statistical mode of the votes was larger than the number of other priorities, then the priority was accepted and recorded on the mind map. Otherwise the participants whose votes differed from the mode were asked to discuss why they voted they way they did. After the discussion, a second round of votes was cast. The process did not require a second round of discussion or third round of voting.

The only significant exception to the voting process were the requirements that were annotated as being previously implemented in a tool that was already in use at the company to manage the developer's daily work. If a discussion was not raised on a such requirement, it was marked as priority level 1 without voting. If the need to vote the priority of the requirement became apparent during the discussion, the priority of the requirement was voted as described in the previous paragraph.

A second meeting about the requirements for reporting work hours was arranged a few days following the discussion and voting meeting. Work-hour reporting is a broad topic and could not be handled in detail during the voting and verification meeting. One senior developer and one development team leader participated in the work-hour reporting requirements specification meeting. The author of this thesis was not the moderator of this meeting. Instead, it was moderated by a researcher from the TKK SoberIT laboratory. The goal of the meeting was not to prioritize the work-hour reporting requirements, but to explore the company's needs for work-hour reporting functionality in general. The SoberIT researcher presented his views on how the reporting functionality should be implemented. The topic was then discussed among the participants. Voting was not conducted in this meeting. The requirements for work-hour reporting were based on this meeting and the interviews the author carried out.

2.6 Selection of the agile software development tools

It was not possible to answer the third research question by reviewing every publicly available agile software development management tool, because there was no guarantee that all existing tools were found. Furthermore, it would have taken an impractical amount of time to review each of the found tools. This issue was mitigated by creating a set of criterion to select the tools that were to be reviewed. A decision to was made to limit the scope to two open-source and two commercial closed-source tools. This number was considered large enough to give a good picture of the current status of the tool support.

The first step in the tool selection process was to find as many agile software development management tools as possible. The methods listed below were used in the search for tools.

- During the selection of the most prominent agile methods (see Section 2.1), all references to tools that were found in the articles were recorded.
- A keyword search was conducted within the SourceForge open source development repository. The searched description field keywords were "story", "backlog", "scrum", and "extreme programming". Each keyword was used in isolation. The search was limited to "Software Development“ and "Project Management" categories. Tools were selected based on their descriptions from the results.
- A Google search of two different search phrases was performed. These search phrases were "scrum backlog management tool" and "extreme programming story management tool". Results were compiled from these results and from sponsored links.
- Several tools were identified through information from informal sources, such as tips from colleagues of the author.

All identified tools were collected in two lists. The first list contained 53 open-source tools and the second list contained 24 closed-source commercial tools. The complete lists can be found in Appendix C.

Second step in the tool selection process was to apply exclusion criteria to the identified tools. The exclusion criteria are explained in the following list. The numbers in the list correspond to the headings in the tables of tools that are presented in Appendix C. The exclusion criteria were then applied to the lists of tools.

1. The tool must be in a mature development stage. The version number of the tool should be at least 1.0. This excludes, for example, release candidates and beta versions. The tools used in managing the daily work of agile software development store

crucial information about the development process. Such tools should be relatively bug-free and mature enough to prevent productivity losses caused by a malfunction in a tool.

2. The development of the tool must be alive. A software development organization using a tool must be able to expect that bugs and other issues in the tool are going to be fixed in the future by the developer of the tool. The users themselves cannot be expected to fix any issues. This excludes abandoned or inactive open-source projects and discontinued commercial products. Open-source projects with only one author or contributor are also excluded on the basis that the risk of discontinuation of the project is too great if something happens to the author. To avoid seemingly alive, but inactive projects, the latest version of the tool must have been released after January 1, 2008. Incremental releases, such as bug fixes and security updates, are accepted as releases.

3. The tool must be primarily directed toward managing the daily work of agile software development organizations. A tool that does not reference agile software development in its intended use or feature list is probably directed toward more traditional projects and is cluttered by features that are not useful in agile context. Tools that contain features that can be used to manage agile software development, but that also contain significant amount of unrelated functionality are also excluded, as the extra functionality clutters the tool. An important part of the agile software development philosophy is to use the simplest tool that works and the extra features and customization options beyond what is required are considered harmful. An example of this kind of tool is a development suite that contains functionality in many aspects of software development, from document management and version control to automated testing.

4. The tool must be available for evaluation. Some commercial closed-source tools do not offer an evaluation version, a trial account, or a demo version. Evaluation of these tools would require purchasing a license for the tool which is not possible considering the resource constraints of this research.

Each of the selected open-source tools were then closely inspected to determine whether the tool could be included in the final review. At this point, all but one of the open-source tools were excluded from the final review. Because only one open-source tool was not excluded from the detailed review, the original goal to review two open-source tools and two commercial closed-source tools could not be reached. In order to keep the original scope of a total of four tools, the decision was made to review one additional commercial closed-source tool.

The third goal of this thesis was to find out if any tool sufficiently fulfilled the requirements of agile software development management. The closed-source tools that would be reviewed more deeply were attempted to be selected in a such way that the selected tools would reflect different philosophies toward project management. This consideration was expected to maximize the probability of finding a tool that would fulfill the requirements. If several similar tools were found, the tool which was most promising, based on its description and feature lists, was selected. Each of the three commercial closed-source tools that were initially selected were also included in the final review.

2.7 Agile software development tool reviews

The first step in the review of the selected tools was gaining access to the tools so they could be reviewed. Various access methods were identified depending of the type of the tool. Some tools ran as services on their developer's servers. They could be accessed with a Web browser and did not require installation in the testbed computer. For those tools, an online demonstration or trial accounts were acquired and used in the reviews. Other tools needed to be installed to the testbed computer. These tools were installed according to any available basic installation instructions.

The system information of the testbed computer used in the test is shown in Table 2.1. While the testbed system did not reflect the hardware configuration of a typical server machine, the reviews in this thesis were focused solely on the functionality of the tools and not on their performance or reliability. The basic functionality of the tools that were reviewed should not have been affected by the hardware configuration that was used. Thus, the hardware differences between a typical server machine and the testbed machine were consequently ignored.

Table 2.1: Testbed computer system information

Base system	Dell Latitude D530 (laptop PC)
Processor	Intel Core 2 Duo 2,2GHz
Memory	2048MB 667MHZ DDR2
Hard disk	7200 RPM Serial ATA
Operating system	Windows XP Professional SP3
Web browser	Mozilla Firefox 3

There were two sets of requirements that needed to be compared to the features in the selected tools. The first set was the requirements from the book review (see Section 4.8) and the second set was the requirements from the case company (see Section 5.10). A separate review was conducted for each of the two requirement sets.

The reviews were conducted by opening the map of the conceptual requirements. Each concept in the requirements was compared to concepts which were available in the tool. The concepts in the concept map were notated depending on how good match for the concept was found within the tool. If there was some ambiguity on which concepts to match, the conceptual requirements and the concepts in the tool were matched in a such way that maximized the number of matching concepts. All concepts in the requirements map were matched one by one. Any concepts in the tool that did not have a corresponding concept in the requirement concepts but had to be used for functional reasons were added to the concept map and marked as excess concepts. The breadth of the conceptual requirements that were filled by the tool was then evaluated based on the markup of the concepts in the concept map. Functional requirements that did not directly relate to concepts such as monitoring work or views in the tool were reviewed in an exploratory fashion.

When it was possible to configure a tool to better match the requirements, the configuration functionality was used. In the case of an extensive and complicated configuration process, the tool was only configured to the extent that an average tool user could be expected to be willing to do.

Chapter 3

Materials

This chapter contains descriptions of the materials that were used in this research. Intermediate results of the materials selection processes are also included. A description of the reason for exclusion is also provided with respect to the materials that have ultimately been omitted from this research. Section 3.1 contains the results from the most prominent agile methods selection process. Section 3.2 contains the results from the agile software development practitioner textbook selection. Section 3.3 contains the results from the agile software management tool selection.

3.1 Agile software development methods

This section presents the results from the selection of the most prominent agile software development methods. A description of how the selection was made is provided in Section 2.1. 14 agile software development methods were identified from the 33 articles that were read. The results concerning the prominence of the identified agile methods are summarized in Table 3.1. The first column shows the method's name and, if available, the generally accepted abbreviation of the method's name. The total number the method is referenced in the articles is located in the second column and the publication years for the articles that mention the method are shown in the third column.

Based on these results, four agile methods were identified as the most prominent. These methods are Extreme Programming, Feature-Driven Development, Lean Software Development, and Scrum.

3.2 Agile software development textbooks

This section introduces the six books that were initially selected according to the title and description that was available on Amazon.com (see Section 2.1). The list of the books that were selected to be read can be found in Table 3.2. The row that references inclusion displays whether the material from the book was included in the in-depth book review.

Agile Software Development with Scrum [Schwaber and Beedle, 2002] is the first official book about Scrum by the method's creators, Ken Schwaber and Mike Beedle. In addition to describing the Scrum process, the book also contains information on the philosophical and theoretical background of Scrum. A newer book, The Enterprise and Scrum

Table 3.1: Results from the agile method popular article search

Method	#	Years referenced
Adaptive Software Development (ASD)	3	2001, 2002
Agile Unified Process (AgileUP)	3	2007
Crystal	4	2001, 2002
Dynamic Systems Development Methodology (DSDM)	4	2002, 2006
Evolutionary Project Management (EVO)	1	2006
Extreme Programming (XP)	26	2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007
Feature-Driven Development (FDD)	5	2002, 2004, 2005, 2006, 2007
ICONIX	1	2005
Lean Software Development	4	2002, 2003, 2006, 2007
Open Unified Process (OpenUP)	2	2007
Scrum	11	2001, 2002, 2004, 2006, 2007
Spiral Development	1	2002
Trilogy's Fast Cycle Time	1	2001
Usage Centered Design	1	2002

Table 3.2: Books included in the book review

Book	Included
Extreme Programming Explained 1st ed. [Beck, 2000]	yes
Extreme Programming Explained 2nd ed. [Beck and Andres, 2004]	yes
A Practical Guide to Feature-Driven Development [Palmer and Felsing, 2002]	no
Lean Software Development: An Agile Toolkit [Poppendieck and Poppendieck, 2003]	no
Agile Software Development With Scrum [Schwaber and Beedle, 2002]	yes
The Enterprise and Scrum [Schwaber, 2007]	yes

[Schwaber, 2007], is mainly focused on using Scrum on enterprise level, but it also includes a brief introduction to Scrum. In the case of conflicting information, the more recently published book was used as the authoritative source.

The first edition of Extreme Programming Explained [Beck, 2000] is the first Extreme Programming book written by Kent Beck, one of the original authors of the XP method [Beck, 1999]. It provides a general introduction to the practices of XP. The second edition of the book [Beck and Andres, 2004] concentrates on the values, principles, and philosophy of XP, but it also includes some instructions on the practical side of XP.

Feature Driven Development and Lean Software Development were not included in the in-depth book review.

A decision to exclude Lean Software Development from the scope of the research was made after reading Lean Software Development: An Agile Toolkit by Poppendieck and Poppendieck [2003]. The book describes a set of thinking tools, "...to aid software development leaders as they develop the agile practices that work best in their particular domain." [Poppendieck and Poppendieck, 2003, p. 8] instead of describing a concrete development process and offers little relevant information for this research. This point of view was further fortified by Poppendieck [2007, p. 166]:

> "Lean Software Development provides the theory behind agile software development practices and gives organizations a set of principles from which to fashion software engineering processes that will work best in the context of their customers, their domain, their development capability, and their unique situation."

After reading and analyzing (see Section 2.2 for the book review protocol) A Practical Guide to Feature Driven Development by Palmer and Felsing [2002] the decision was made to exclude Feature Driven Development from the rest of the research. Feature Driven Development differs notably from Scrum and XP in its approach and philosophy to software development [Highsmith, 2002, p. 280-283]. While creating a combination model of Extreme Programming and Scrum was considered to be relatively easy due to the similarities between the two methods [Larman, 2004, p. 163], the requirements that FDD has for a tool are very different and would have required a completely separate tool review.

3.3 Agile software development management tools

As a result of applying the exclusion criteria to the list of all found tools (see Section 2.6), four open-source and nine commercial closed-source tools were left as potential candidates for the detailed review. The open source tools are listed in Table 3.3 and the commercial tools are listed in Table 3.4. The first column in the tables contains the tool names and the second column references the tools' authors.

The open-source tools to be included in the tool review were selected first. The first candidate was Agilefant, which was developed by the Software Business and Engineering Institute in Helsinki University of Technology. The tool was selected for inclusion in the in-depth review. A general introduction to Agilefant can be found in Section 6.1.1

The second open-source candidate for review was eXtreme Management Tool by Zest Software. The tool is primarily intended to manage development in an Extreme Program-

Table 3.3: Open-source tool candidates for review

Tool	Author
Agilefant	TKK / SoberIT
AgilePlanner	Frank Maurer et al.
eXtreme Management Tool	Zest Software
Project Dune	Gerard Toonstra et al.

Table 3.4: Commercial closed-source tool candidates for review

Tool	Author
DevPlanner	Fedorenko
Mingle	ThoughtWorks
OnTime	Axosoft
Rally Enterprise Edition	Rally Software Development
ScrumDesk	ScrumDesk
ScrumWorks Pro	Danube Technologies
VersionOne	VersionOne
VisionProject	Visionera

ming project. It is implemented as an plug-in on Plone content management system. eXtreme Management Tool was first selected to be the second tool to be reviewed. The web page of the tool does not offer any installation instructions. Installation of the tool was attempted several times with several different methods based on information in a general 3rd party plug-in installation guide for Plone. While the installation seemed to success each time, the result was always a broken Plone instance. Based on the lack of installation instructions and the troublesome installation process the tool was finally dropped from the detailed review.

The third candidate open source tool was AgilePlanner created by The Agile Software Engineering/e-Business engineering (ASE/EBE) group at the University of Calgary. AgilePlanner's functionality is geared toward providing virtual board for planning game with story cards for large and distributed projects. It provides little project management functionality and therefore it was not a good candidate and was therefore excluded from the detailed review.

The fourth open source tool candidate was Project Dune, which is an open source tool and project developed by a group of contributors headed by Gerard Toonstra. Most of the tool's features concentrate on managing issues and documentation. The tool's feature list includes managing Scrum tasks. However, managing tasks is implemented as an aside on the issue management; every task in the tool must be linked to an issue. Because of the focus on issue management, the tool is not suitable for general management of agile software development and was excluded from the detailed review.

Based on the findings described above, Agilefant was the only open source tool included in the detailed review.

The only commercial closed source tool excluded based on how the developer of the tool describes the the tool was DevPlanner by Fedorenko. The tool's description says it is a "personal day planning software" [Fedorenko, 2008]. DevPlanner's main purpose is to help an individual developer to plan and follow the usage of time with task estimation and daily schedule. The tool is capable of creating team reports, but it does not have sufficient features to be useful in managing work at a team level. DevPlanner could be used to manage the work of an individual developer during agile software development, but considering the goals of this thesis, it was not relevant. Therefore, it was excluded from the detailed review.

The rest of the commercial tools were divided into three groups that differ notably in philosophy toward software development. The first group represents tools that have flexible conceptual models and must be configured before they can be used. Mingle is the only tool that belongs to this group. The second group represents tools that are aimed toward a specific agile software development method. ScrumWorks Pro and ScrumDesk belong to this group. The third group represents tools that are intended for agile software development in general. These tools may offer some configuration options, but they do not require extensive configuration to be useful. Rally Enterprise Edition, OnTime, VersionOne and VisionProject belong to this group.

One tool from each group was selected to be reviewed in depth. The selected tools were Mingle, Rally Enterprise Edition and ScrumWorks Pro. General introductions to the selected tools can be found in Sections 6.2.1, 6.3.1 and 6.4.1, respectively. Mingle was the self-evident selection from the first group, as it was the only tool in the group. ScrumDesk's features are directed for providing a virtual board for story cards in a Scrum project, while ScrumWorks Pro is more project management oriented. Based on this, ScrumWorks Pro was selected for detailed review. No large differences in the coverage of interesting features were found in the four tools in the third group. There were however notable differences in the amount of features not related to the subject matter of this thesis. Rally Enterprise Edition was selected to represent the third group. It seemed to contain the least amount of unrelated functionality.

Chapter 4

Requirements for a tool according to the book review

This chapter describes the results of the book review. Each section contains requirements for Scrum, XP and the combined model. Section 4.1 presents the four concept types that were identified in the analysis of the results. Most of the requirements resulting from this analysis are mapped to these four concept types. Conceptual maps of the two methods and the combined model can be found in Section 4.5. Descriptions of the identified concepts (disregarding properties, which only specify other concepts) are found starting from Section 4.2, which describes the identified time horizons. The followings Sections 4.3 and 4.4 describe the identified containers of work and the identified pieces of work, respectively.

There are two requirements that are related to every concept but that are not listed in this chapter. These requirements are the creation of a new instance of a concept type and the deletion of an old one. These operations are implicit requirements for every concept of every type and they are not repeated for each concept.

In addition to the requirements identified from the conceptual maps, the requirements for selection of work (Section 4.6) and monitoring progress (Section 4.7) are described in this chapter. Finally, Section 4.8 presents a summary of the requirements from the book review.

When Extreme Programming or Scrum is referenced as a source of requirements in this chapter, it only represents the method as described in the books used as sources. Other implementations of the methods exist and references to these two methods should not be interpreted as generalizations to all published implementations of the methods.

4.1 Identified concept types

Figure 4.1 shows the four different types of concepts that were identified in each included method. All concepts that are of importance in this research were mapped to these four types. The concept types are explained in the following list.

Time horizon is a part of temporal dimension that has a defined purpose in the method. From a planning perspective, a time horizon is a length of time which is planned ahead.

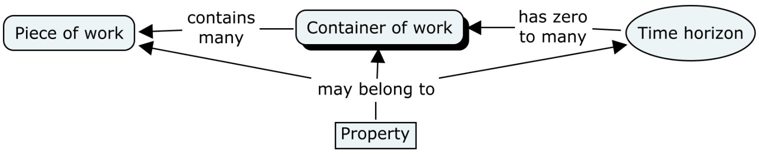


Figure 4.1: Concept types of the book review requirements

Container of work groups units of work. Most often the grouping is done based on the time horizon in which the piece of work is going to be done, but other groupings also exist.

Piece of work is an abstraction of the actual work that needs to be done. It may be presented in different ways, for example as a description of something the system must be capable of or as an actual description of something that needs to be done.

Property specifies the concept to which it is associated. All other concept types may have properties associated with them.

4.2 Identified time horizons

This section presents the time horizons identified in the book review. Four time horizons were identified in Scrum and two time horizons were identified in XP. The time horizons identified in Scrum are described in Section 4.2.1 and the time horizons identified in XP are described in Section 4.2.2. In addition, the combined model of the time horizons from both methods is described in Section 4.2.3.

4.2.1 Time horizons in Scrum

This section describes the four time horizons identified in Scrum. These time horizons were the *product*, *project*, *release* and *sprint*. Based on the sources used, no direct relationship exists between the product, release and sprint time horizons. However, the three time horizons do have indirect relationships between them, as their associated containers of work have direct relationships (see Section 4.3.1). Unlike the other three time horizons, the project time horizon does not have a container of work associated with it. However, it has a direct relationship with the product time horizon.

Product

The product time horizon represents the complete life cycle of a product and it is the highest level time horizon in Scrum. Each product time horizon has a *product backlog* attached to it. The product has a vision of the system, or a *system vision* property. The vision of the system to be built is the highest level planning tool in Scrum. The vision may be vague and stated in market terms at first, but becomes clearer as the development of the product proceeds. Implementing all *product backlog items* results in realization of the system vision.

Project

A product is developed by one or more Scrum *projects*. All projects which develop the same product share the same product backlog. The project is not a well defined time horizon in Scrum. It is more of a context in which Scrum is used. Scrum defines two properties specific to the project time horizon: the *complexity factor* and the *project goal*. The complexity factor increases the effort estimates of product backlog items based on the characteristics of the project that decrease the productivity of the developers in the project. Project goal is the reason the project exists. When the project goal is reached the project is completed.

Release

In Scrum, the release is the time horizon that is used in long term planning. The length of a release is not defined in Scrum, but several sprints are usually completed during one release. Each release has a date and a *release backlog* (see Section 4.3.1). The division of product backlog items into release backlogs conveys the long term development plan of a product. If more or less work than originally planned is completed in a release, the contents of the release backlog may be altered or the date of the release may be changed.

Sprint

The sprint is the development iteration level time horizon in Scrum. The sprints belonging to the same release are executed in sequential order. One month is the normal length of a sprint, but a shorter duration may be used. Regardless of the length, each sprint in a Scrum project must be of equal length. There are two containers of work (see Section 4.3.1) related to the sprint: A *set of product backlog items selected to be developed in a sprint* and a *sprint backlog*. In addition the sprint has a *sprint goal*, a *sprint plan* and a *team capacity*.

The sprint goal is a subset of a project goal and it describes the purpose of a sprint. A sprint goal is crafted based on the product backlog items that have been selected to the sprint in a such way that the completion of the items will result in reaching the sprint goal. Tasks in a sprint backlog are devised in a such way that completing them will convert the product backlog selected to the sprint into the sprint goal. If the scope of a sprint needs to be reduced, product backlog items can be removed from the sprint and re-entered into the product backlog or the scope of the items can be reduced, as long as what remains meets the sprint goal.

The sprint plan is a tentative plan of how a sprint will be carried out. This includes a plan of what needs to be done, how it will be done and who will do it. The sprint backlog can be considered to be a part of the sprint plan.

The team capacity tells how much work a team is capable of doing in a sprint. Sprint's team capacity is derived from the effort estimates of product backlog items that a team managed to complete in a preceding sprint. In the first sprint the team capacity is an estimate.

4.2.2 Time horizons in Extreme Programming

Extreme Programming defines two time horizons: the *release* and the *iteration*. Beck and Andres [2004, p. 91] mentions the project time horizon, which has a collection of stories and a deployment date as properties. The relationship between the project time horizon and the rest of the XP method is not explained in the sources used and, therefore, project time horizon is excluded from the results concerning XP.

Release

The release is the long term planning time horizon in Extreme Programming. The release time horizon has *stories selected to the release* as a container of work. The details of a release in XP are planned by picking the stories that are going to be implemented into the release. The suggested length of a release is a quarter of an year, but other lengths can be used. The date of a release can also be calculated from the effort estimates of the stories that have been selected to be implemented in the release. The release has one or more *themes*. The themes are textual descriptions of how the release fits in to the big picture. No detailed information on how to write themes is given in the sources used.

Iteration

The iteration is the development level time horizon in Extreme Programming. Each release contains several subsequent iterations. The iteration time horizon has a *collection of tasks*, *stories selected to the iteration* and *"done" stories* as containers of work. The collection of tasks in an iteration acts as a detailed plan for the iteration. The suggested length of an iteration is one week, but other lengths can be used. After the completion of an iteration the *actual work that was accomplished* is recorded. This value is then used as a guideline when selecting stories into the following iteration.

4.2.3 Combined time horizons

XP and Scrum both have nearly identical release and iteration time horizons. In addition to these two time horizons, Scrum describes two higher level time horizons: the project and the product. The time horizons form a hierarchical structure, where a higher level time horizon may contain one or more lower level time horizons. All time horizons except project have a backlog (see Section 4.3.3) containing backlog items (see Section 4.4.3) associated to them. The names of the combined time horizons follow the names in Scrum with the exception of the lowest level time horizon, where the more general *iteration* is used instead of the Scrum specific sprint.

Product

The product time horizon represents the complete life cycle of a product from the product's inception to its discontinuation. Apart from a name or other identifier, the product has no properties associated with it. Each product has a product backlog which contains all backlog items that have been created for the product. Regardless of whether the backlog items have been selected to be developed or not, a product's backlog contains all work that should be done for a product.

Project

The project time horizon is the only time horizon which has no backlog associated with it. Project has two properties: a complexity factor and one or more project goals. The complexity factor reflects the complexity of a project and increases the effort estimates of backlog items related to the project accordingly. The project goals are textual representations of the high level goals of a project. A project may be associated with a product that is being developed in the project, but projects that are not associated with a product may also exist.

Release

The release time horizon is used to plan the content of product releases. In addition to a name or an identifier, each release has a set date. A release may also have several release goals, which are textual representations of the purposes of the release. A release may be associated with a project which is the context of the release, but releases that are not associated with any projects may also exist. If a release is associated with a project, each release goal may be associated with one project goal. Each release has a release backlog containing all backlog items selected to be developed in the release.

Iteration

The iteration is the development level planning time horizon. An iteration may be associated with a release, but iterations not associated with a release may also exist. Each iteration has a set length and may have one or more iteration goals. An iteration has a capacity property which is the amount of work that is estimated to be completed in the iteration. The actual amount of work accomplished in an iteration is also recorded after the iteration is over. Iteration goals are textual representations of the goals of an iteration. If an iteration is associated with a release, each iteration goal may be associated with one release goal. Each iteration has an iteration backlog associated with it. The iteration backlog contains all backlog items that have been planned to be implemented in an iteration.

4.3 Identified containers of work

This section presents the containers of work identified in the book review. The different backlogs in Scrum and collections of stories in Extreme Programming are quite similar in purpose. One notable difference is that, in Scrum, the backlog items picked from a higher level backlog to a lower level backlog are still part of the higher level backlog, while in XP stories are moved between containers. In addition to the containers of backlog items or stories, both methods have a container for a different type of piece of work called task. The containers of work in Scrum are presented in Section 4.3.1 and the containers of work in XP are presented in Section 4.3.2. In addition, a combination of the two methods' containers is presented in Section 4.3.3.

4.3.1 Containers of work in Scrum

There are two kinds of containers of work in Scrum. One type contains product backlog items and the other tasks. Product backlog items are contained in *product backlogs*, *release backlogs* and in a *product backlog selected into a sprint*. Selecting an item into a release backlog or sprint does not remove it from the product backlog. The two lower level backlogs are kinds of temporal dividers of the whole product backlog and are containers only from a logical point of view. The *sprint backlog* is the only container for tasks. Tasks exist only in the sprint backlog. The sprint backlog is completely distinct from the three containers of product backlog items.

Product Backlog

The product backlog is a prioritized list of product backlog items that must be done for a product to realize its system vision (see Section 4.2.1). All known items that belong to a product are listed in the product backlog. A product backlog constantly changes as the priorities of items change and new items are added. A product backlog exists from the beginning of the development of a product with Scrum-process to the termination of the product.

Release Backlog

The release backlog is a part of the product backlog that has been selected for a release of a product. A product backlog is divided into several proposed release backlogs. The closer the date of a release is, the more detailed are the release backlog items in the release backlog. A product's release backlogs are altered during the development of the product to adapt to the changing environment and requirements.

Product Backlog in Sprint

The product backlog in a sprint is the part of a product backlog that is selected to be implemented in a sprint. Only such limited number of product backlog items are selected that they can be turned into an increment of product functionality during the sprint. This selection is affected by the team capacity of the team that is developing the product in the upcoming sprint. A team capacity is derived from the effort estimates of the product backlog items that the team completed in the previous sprint. If the sprint is the first sprint, then the team capacity is an estimate.

Sprint Backlog

A sprint backlog contains all tasks that have been devised for a sprint. In addition to normal tasks, a sprint backlog can contain reminders of probable future work. These reminders are special tasks that don't have any specifications and are marked as being remainders. Reminders of future work are created when all tasks for a sprint cannot be formulated before further investigation or design. The sprint backlog can only exist in the context of a particular sprint.

4.3.2 Containers of work in Extreme Programming

Extreme Programming has two kinds of containers of work. The first type contains stories and the second type tasks. Unlike the backlogs in Scrum (see Section 4.3.1), the containers in XP are not explicitly defined in the sources used. Each XP time horizon (see Section 4.2.2) has a set of stories selected into it. In addition, the iteration has tasks that have been devised for it. These are the main containers of work in XP. XP also has a container which contains "done" stories.

Stories not in a release

The first container of stories in XP contains the stories that have not yet been selected into a release and the stories that have been removed from an ongoing release. The container is not explicitly named in the sources used and will be called the *stories not in a release* in this thesis.

Stories in a release

The second container of stories in XP is the stories that have been selected into a release from the collection of stories not in a release. This container will be called the *stories in a release* in this thesis. Only an ongoing release or a release that is currently being planned have a such container. Future releases after the next one are not planned beforehand using stories. The only stories selected to a release are those that can be implemented in the time available.

Stories in an iteration

The third container of stories in XP is the stories that have been selected for implementation in an iteration. It will be called the *stories in an iteration* in this thesis. The container contains the most valuable stories that can be implemented in the iteration. The stories are selected from the collection of stories in a release.

"Done" stories

In addition to the other containers described in this section, Extreme Programming Explained 2nd edition has a picture [Beck and Andres, 2004, p. 40] that shows a collection of "done" stories. The exact role of the collection of "done" stories is not explained. This leaves open several questions: Is the "done"-status a property of a story or is there a container of work for "done" stories? Should the "done" stories be discarded after an iteration or a release is completed or kept as an history log of the project? For the purpose of this research the container of "done" stories is assumed to contain the completed stories of the ongoing iteration which were moved to the "done" container from the stories in an iteration container. The container will be called the *"done" stories* in this thesis.

Tasks of an iteration

The only container in which tasks can exist is the collection of tasks that have been devised for an iteration. This container will be called the *tasks of an iteration* in this thesis. Optionally the tasks of an iteration can be kept in a stack so that the topmost task on the stack is always the first to be selected for completion. If tasks are not used (see Section 4.4.2) in a XP project, this container does not exist.

4.3.3 Combined containers of work

The five containers of work in the two methods can be mapped almost exactly into five combined containers of work. The only exception is the collection of "done" stories in Extreme Programming. This collection is left out of the combined concepts, as a separate "done" flag conveys the same information.

Because Extreme Programming terminology does not give explicit names for the collections of stories or tasks, the names for the combined containers of work have been taken mostly from Scrum terminology with two exceptions: Scrum terminology does not explicitly name the list of product backlog items selected to be developed in a sprint and the collection of tasks in a sprint is named sprint backlog. As all other backlogs in Scrum contain backlog items instead of tasks, this is a potential source of confusion in the terminology. To increase consistency and avoid confusion between the different types of backlogs, the list of backlog items in an iteration is named *iteration backlog* and the set of tasks in an iteration is named *task list*.

The backlogs in the combined model keep the backlog items belonging to them in a priority order. If relative priority is used in the backlog item prioritization (see Section 4.4.3), the backlogs keep items in a relative priority order. If priority classes are used, the backlogs keep items grouped into classes and the groups are kept in a relative priority order. If a combination of the two is used, the backlogs keep their backlog items prioritized relatively inside the priority class groupings and the groups are also relatively prioritized.

Product backlog

The *product backlog* is the highest level container of work in the combined concept map. Each product has one product backlog. All work that is going be done, is currently under work or has already been completed for a product is listed in the product backlog. As such, the product backlog acts also as a history log for the development of the product.

Release backlog

The *release backlog* contains the backlog items that have been selected to be implemented in a release. A release backlog is a subset of a product backlog if the release in question is associated with a product. A product backlog may be divided into several subsequent release backlogs, but a backlog item can only belong to at most one release backlog at a time.

Iteration backlog

The iteration backlog contains the backlog items that have been selected to be implemented in an iteration. An iteration backlog is a subset of a release backlog if the iteration in question is associated with a release. A release backlog may be divided into several subsequent iteration backlogs, but a backlog item can only belong to at most one iteration backlog at a time.

Task list

The tasks that have been devised for an iteration are listed in the task list. Tasks can only exist in the context of an iteration. While tasks in Scrum are not directly connected with backlog items, this connection is available in the combined concept map. Unlike tasks, backlog items can be moved between iteration, release and product backlogs. This may result in a situation where tasks that are associated with a backlog item are orphaned in an iteration when the associated backlog item is moved to a different backlog. To avoid the possible confusion caused by this situation, the task list should clearly indicate the tasks that are associated with backlog items that are not in the same iteration's backlog.

4.4 Identified pieces of work

This section presents the pieces of work identified in the two agile methods. In addition to the individual methods, a combined model of the two methods' pieces of work is presented in Section 4.4.3. The pieces of work in Scrum are described in Section 4.4.1 and the pieces of work in XP in Section 4.4.2. Both methods have two different types of pieces of work. The first type is called backlog item in Scrum and story in Extreme Programming. The second type is called task in both methods.

4.4.1 Pieces of work in Scrum

Scrum has two different pieces of work. The first one is the *product backlog item* and the second one is the *task*. The two pieces of work serve different purposes and their progress is monitored on different time horizons (see Section 4.2.1). Excluding work such as the daily scrum, which is part of the Scrum process itself, the two pieces represent all work that is done in a Scrum software development project.

Product Backlog item

The first piece of work in Scrum is the product backlog item. Table 4.1 summarizes the properties of the product backlog item. A product backlog item is a requirement or a feature and may be, for example, functionality, issue, bug fix, technology or enhancement. Product backlog items always reside in a product backlog. See table 4.3.1 for more info on the product backlog.

The only required operations on the product backlog item are setting and changing its properties. This includes changing the priority order between product backlog items in a product backlog.

Task

The second piece of work in Scrum is the task. A sprint backlog contains all work a team plans to accomplish during a sprint. The items in a sprint backlog are called tasks. By

Table 4.1: Scrum product backlog item properties

Property	Definition
Name	A short descriptive identifier
Description	A description of what the item consists of
Initial estimate	An amount of effort in days that was estimated to go into the development of the item before the development started
Adjusted estimate	An initial estimate adjusted with a complexity factor (see Section 4.2.1); The value is used in the selection of new items into a release or a sprint
Work remaining	Amount of work in days that is estimated to be remaining in the beginning of each Sprint; The value is used in the selection of unfinished items into a release or a sprint
Relative priority	The priority of the item related to the other items in a product backlog; The priority is used in the selection of product backlog items into releases and sprints
Issue status	An item may be denoted as an issue; An issue needs to be refined and turned into a regular item or items before work on it can begin

completing the tasks in a sprint backlog the development team reaches the sprint goal or goals of a sprint (see Section 4.2.1). Tasks are not direct decompositions of product backlog items selected into a sprint but rather things that need to be completed to turn the items into shippable product functionality. In terms of estimated development effort and description detail, the task is more fine grained than the product backlog item. Table 4.2 summarizes the properties of the task. The only required operations on the task are setting and changing its properties.

Table 4.2: Scrum task properties

Property	Definition
Description	A detailed description of the task
Originator	The team member who created the task
Responsible	The team member who is responsible of doing the task
Work remaining	An amount of work in hours that is estimated to be remaining for each day of a sprint; that is to say, there is one value for each bygone day of the sprint; this value is used during a sprint to monitor if the fulfillment of the task is feasible before the end of the sprint
Status	The progress status of the task; May be “not started”, “in progress” or “completed”; this property is used to monitor how the fulfillment of a task is proceeding

4.4.2 Pieces of work in Extreme Programming

Two different pieces of work are used in Extreme Programming. The first piece is the *story* and the second is the *task*. Usually tasks are direct decompositions of one or more stories, but tasks that are not related to any story can also exist. The two units of work have similar purpose, but their progress is monitored on different time horizons (see Section 4.2.2).

Story

The first piece of work in XP is the story. It is described in the sourced used as an piece of customer visible functionality, as something the system needs to do and as a customer requested feature. Table 4.3 summarizes the properties of the story.

In addition to setting and changing the properties of a story, splitting a story into two or more stories must be possible. This is done if the whole story cannot be estimated or a part of a story is more important than the rest.

Table 4.3: Extreme Programming story properties

Property	Definition
Name	A short name for the story
Description	A short paragraph describing the purpose of the story
Effort estimate	The development effort estimated to go into implementation of the story; can be in real time (hours or days) or in abstract story points
Value class	The value class to which the story belongs; is one of: (1) system will not function without, (2) provides significant business value or, (3) nice to have; used in the selection of stories to be implemented in a release (see Section 4.3.2)
Risk class	The estimation imprecision risk class the story belongs to; is one of: (1) can be estimated precisely, (2) can be estimated reasonably well or, (3) cannot be estimated at all; used in the selection of stories to be implemented in a release (see Section 4.3.2)
Priority class	The priority class can be one of the following: (1) work on exclusion of all else, or (2) low value; only used when all selected stories cannot be implemented in a release or an iteration to direct effort into the most crucial tasks

Task

The second piece of work in Extreme Programming is the task. A task describes something that a programmer knows the system must do. Tasks are usually direct decompositions of one or more stories, but tasks that are not directly related to any stories can also exist. Table 4.4 summarizes the properties of the task.

In addition to changing the properties of a task, the operations that are required for the task are splitting a task if the task is too large (more than few days) and combining several tasks that are too small (hour or less each).

According to the sources used, the need for tasks can be eliminated by using tasks that are small enough. However, the sources used do not explain how the properties of the story and the task are to be combined or define how small is small enough.

Table 4.4: Extreme Programming task properties

Property	Definition
Description	The description of the task
Estimate	The programming effort estimate of the task estimated in ideal programming days
Effort spent	Records how much time the responsible person has has spent implementing the task; the unit used is not specified in the sources used
Effort left	Records how much time is estimated to be left in the implementation of the task in ideal programming days
Responsible	The person who is responsible for estimating and performing the task
Related stories	One or more stories the task is decomposed from; may be empty if the task is not related to any stories

4.4.3 Combined pieces of work

Both XP and Scrum have two different pieces of work which are conceptually quite similar. The first one is a high level planning tool that is used in planning which requirements and features should be implemented and when they are going to be implemented. This combined high level piece of work is called *backlog item* in the combined concept framework presented in this chapter. The second piece of work in both methods is called *task* and that name will be also used in the combined concept framework. Tasks are used in the operational level planning of iterations.

Backlog item

The naming of the first piece of work does not imply that Scrum is favored over Extreme Programming. It is rather an effort to avoid the implied writing format that the name “story” conveys. In addition, the term backlog is often used to represent the collection of work that needs to be done: “an accumulation of tasks unperformed or materials not processed” [Merriam-Webster, 2008]

Table 4.5 describes the properties of the combined backlog item. Instead of the XP-style container of “done” stories, a flag is raised when the backlog item is “done”. This simplifies the monitoring of progress in the combined model (see Section 4.7.3). The only operations that are required for the combined backlog item are changing the properties it has and selecting it to different backlogs.

Task

In XP a task is usually a direct decomposition of a story into smaller parts. However, in Scrum tasks describe something that needs to be completed by a developer instead of describing a requirement or feature for the system. Apart from this difference, the two pieces of work are quite similar. Both are used in iteration level planning to plan the more detailed execution of an iteration. Table 4.6 shows the properties of the combined task. In

Table 4.5: Combined backlog item properties

Property	Definition
Name	A short descriptive and unique name
Description	A short paragraph describing what the item consist of
Initial estimate	An initial development effort estimate of the backlog item; the unit is a point which can for example represent days, hours or abstract estimation points
Priority	The priority of the backlog item; can be either relative to other backlog items, based on at least three different priority classes or a combination of the two: relative priority order inside each priority class
Risk class	The estimation precision class to which the backlog item belongs; is one of (1) can be estimated precisely, (2) can be estimated reasonably well, or (3) cannot be estimated (issue)
Value class	Value class to which the backlog item belongs; Is one of (1) system will not function without, (2) provides significant business value, or (3) nice to have
Adjusted estimate	If the project complexity factor is used (see Section 4.2.3), the adjusted estimate of the backlog item is calculated based on it; when present, the adjusted estimate is used in all calculations instead of the initial estimate
Urgency flag	Backlog items that have the urgency flag raised must be worked on in exclusion of all other backlog items
Done flag	Completed backlog items are marked as “done” by raising the done flag

Table 4.6: Combined task properties

Property	Definition
Description	A detailed description of the task
Effort left	The amount of work that is estimated to be remaining after each day of an iteration, a.k.a. there is one value for each bygone day of the iteration; The unit is a point which can represent for example hours, days or abstract estimation points
Responsible	The person who is responsible of the execution of the task
Originator	The person who created the task
Status	The progress status of the task; Is one of (1) not started, (2) started or (3) completed
Effort spent	The amount of work that the responsible person has spent in the implementation of the task; The unit is a point which can represent for example minutes, hours or days
Related backlog items	The backlog item or items the task is related to; may be none if the task is not related to any backlog items

addition to changing the properties that the task has, splitting and combining tasks should be possible.

4.5 Concept maps of the identified concepts

This section contains figures of the three concept maps that were created. See Section 4.1 for an explanation of the four different concept types used in the maps. Detailed descriptions of the concepts in the maps and their relationships can be found in Sections 4.2–4.4. Figure 4.2 shows the concept map of concepts in Scrum, Figure 4.3 shows the concept map of concepts in Extreme Programming and Figure 4.4 shows the concept map of concepts in the combined model.

4.6 Selection of work

This section describes how a developer in a Scrum (Section 4.6.1) or XP (Section 4.6.2) project selects the next item he or she is going to complete. In addition, a combination of requirements for selection of work from the two methods is presented in Section 4.6.3.

4.6.1 Selection of work in Scrum

The next thing a developer should do during a development iteration in Scrum is selected based on the best judgment of the developer during a daily scrum meeting. If there are tasks that the developer is responsible for, the developer might select one of the tasks and work on it. The selection of the task is again based on the best judgment of the developer. Product backlog items have a relative priority order, but as tasks are not direct derivates of backlog

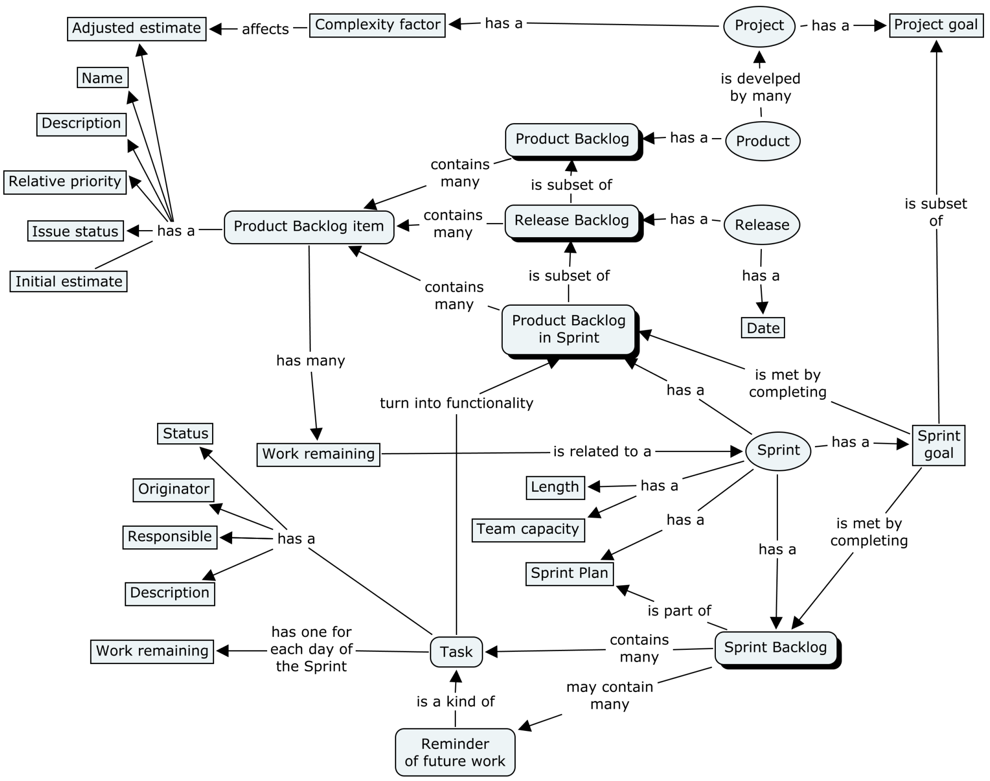


Figure 4.2: Scrum concept map

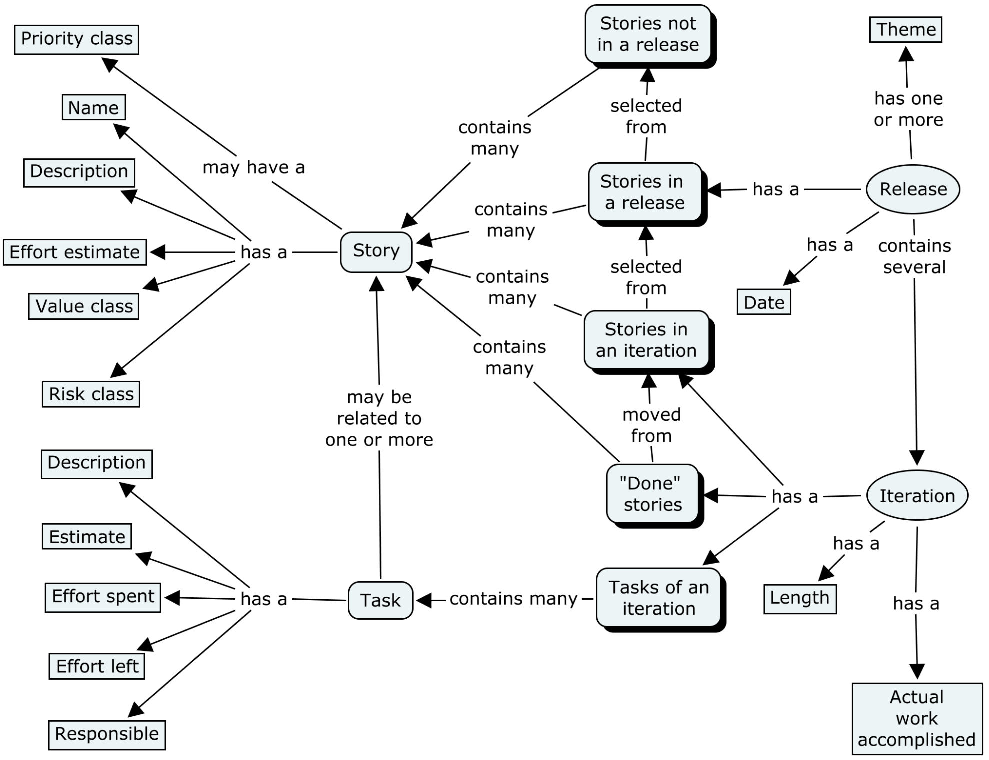


Figure 4.3: Extreme Programming concept map

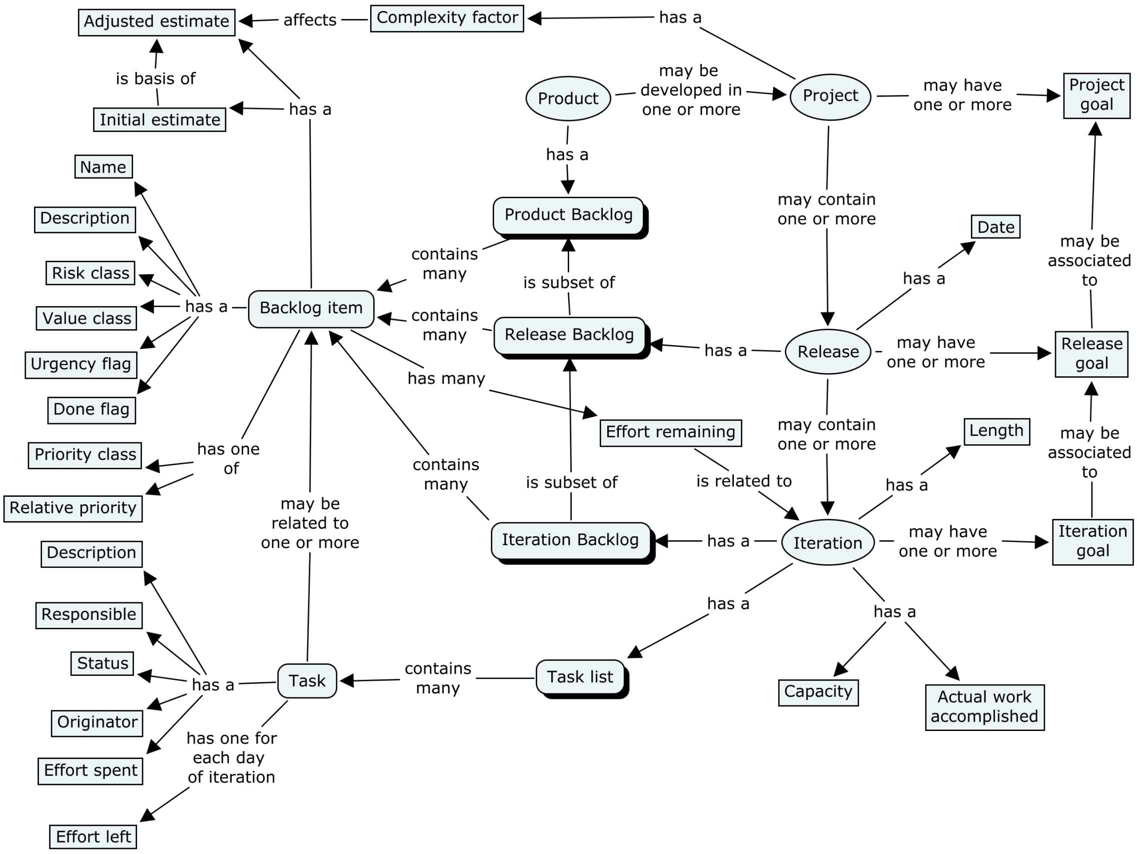


Figure 4.4: Combined concept map

items they can only be used as a reference. In addition to completing a task, a developer may need to attend a design session, refine an issue or perform some other action related to software development with Scrum.

4.6.2 Selection of work in Extreme Programming

During Extreme Programming's iteration planning each programmer signs up for a set of tasks. During iteration a programmer may select one of the tasks he is responsible for and find a pair programming partner to implement it with or he can act as a pair for one of the other programmers. The selection of a task and the pairing are done based on the best judgment of the programmer in question.

An alternative for the iteration planning time task sign up is to use a stack of tasks. Task selection is done by picking the topmost task for implantation and then finding a pair to implement it with. After the task has been implemented, the programmer picks the next task from the top of the stack.

4.6.3 Combined selection of work

Two combined selection of work requirements can be devised based on the requirements from Scrum and XP. The first requirement is a way for developers to see the tasks that have been assigned to them. As tasks have no priority order, the selection of the next task is also based on the best judgment of the developer. The second requirement is the implementation of the stack based task selection from Extreme Programming. It can be implemented for example by keeping an ordered list of tasks, from which the developer picks the topmost unassigned task.

4.7 Monitoring progress

This section describes how the progress of work in a Scrum (Section 4.7.1) or XP (Section 4.7.2) is monitored. Example charts that are used for monitoring progress in the two methods are also provided. Combined progress of work monitoring requirements are described in Section 4.7.3.

4.7.1 Monitoring progress in Scrum

The progress of work is monitored in Scrum on both Sprint and Release time horizons using *burn-down charts*. On the sprint level, a burn-down chart displays the amount of work remaining in hours in the sprint backlog. One data point summed from the effort estimates of the tasks in the sprint is shown for each day of the sprint. Figure 4.5 shows an example of a *sprint backlog burn-down chart*. The date of a sprint backlog completion can be estimated by comparing the reduction speed of remaining work of the previous days to the amount of remaining work calculated by summing the remaining work of all tasks. This comparison can be done in several ways, for example by using the average daily reduction in remaining work.

On the release time horizon, a release backlog burn-down chart is used. The *release backlog burn-down chart* is similar to the sprint backlog burn-down chart, but the time axle is in Scrum sprints and the remaining work is recorded as days of work. The amount of remaining work is calculated by summing the remaining work of all product backlog items that belong to the release backlog.

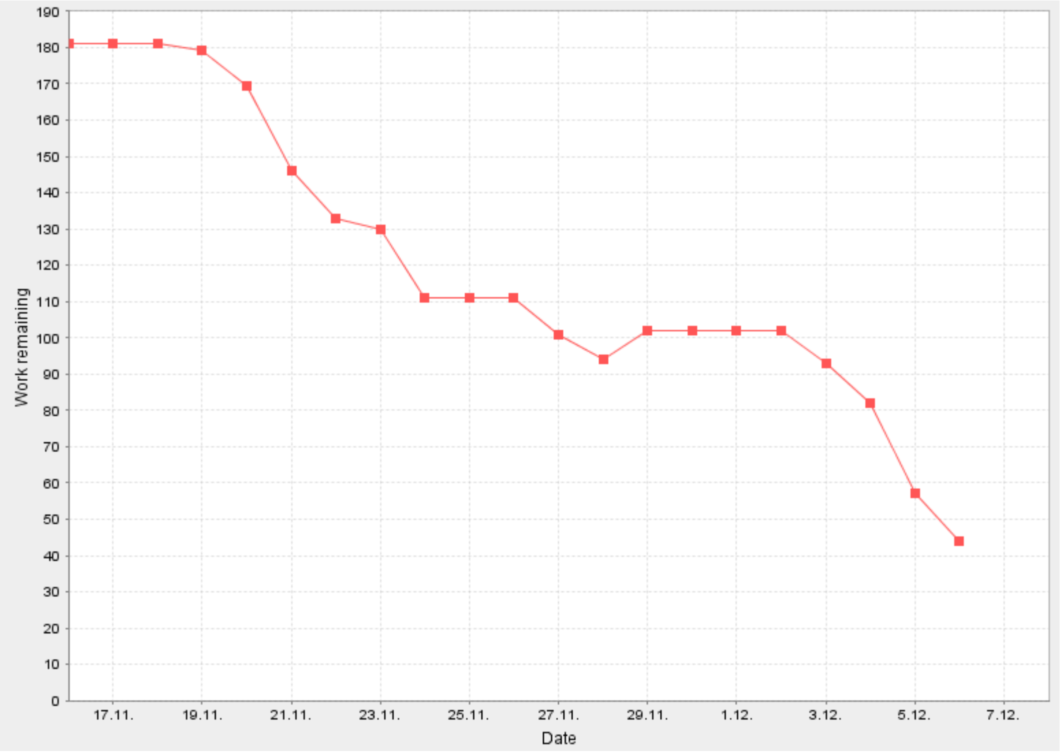


Figure 4.5: Example of a sprint backlog burn-down chart

4.7.2 Monitoring progress in Extreme Programming

In Extreme Programming, the progress of work is monitored on the release, iteration and individual programmer levels. On the release level, the progress of work is monitored by monitoring the amount of stories in the stories in a release container of work.

On the iteration level, the progress of work is monitored by comparing the amount of stories that are in the stories in an iteration container to the amount of stories that are in the "done" container of work. For example, if the stories in an iteration container has more stories than the "done" container when the end of the iteration is near, it is likely that all stories cannot be completed before the iteration end. The sources used do not specify whether the estimated amount of work in the stories or only the number of stories should be used in monitoring work. Figure 4.6 shows an example of monitoring progress of work in XP. The gray boxes in the figure represent individual stories and the three squares represent different containers of work.

On the individual developer level progress of work is monitored from the effort left estimates of tasks. If the sum of a developer's tasks effort left estimates is larger than the

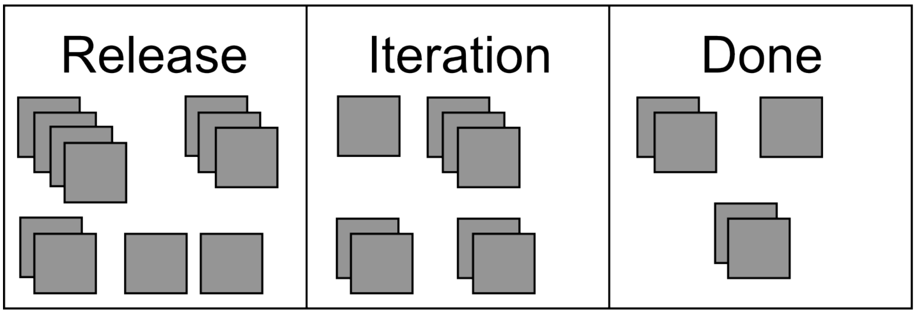


Figure 4.6: Example of monitoring work in Extreme Programming

work time left in the iteration, the developer is probably over committed and cannot finish all tasks.

4.7.3 Combined progress monitoring

The method of monitoring the progress of work on the release and iteration levels is quite similar in both methods. The largest difference is that the burn-down charts in Scrum also display historical data, while the amount of stories in the different stages of development used in Extreme Programming displays only the current situation. In addition, the progress of work on backlog items is not monitored during a sprint in Scrum, while in Extreme Programming the progress of work on stories during an iteration is monitored by comparing the amount of "done" stories to the amount of stories in the iteration that are not yet done.

Both methods' progress monitoring requirements are included in the requirements of the combined model. The first requirement is the existence of Scrum style release and iteration burn-down charts. The release burn-down chart shows the sum of effort left estimates of all backlog items in a release backlog in the beginning of each iteration of a release. The iteration burn-down chart shows the sum of effort left estimates of all tasks in an iteration backlog for each day of an iteration. The second requirement is a Extreme Programming style numerical or graphical display which allows the comparison of "done" stories to the rest of the stories in an iteration or release.

One requirement for monitoring progress on the individual programmer level is taken from XP. An individual developer must be able to see the sum of effort left estimates of the tasks that are assigned to him in a selected iteration.

4.8 Summary of the requirements from the book review

This section summarizes all combined model requirements. For more detailed information on each requirement, see the corresponding section in this chapter. The summary of requirements is presented in figure 4.7. The requirements are presented in a tree form. The first level of branching groups the different types of requirements together, while the second and third level branches present the concrete requirements. Some of the requirements that describe connections between concepts are described as optional, which emphasizes the fact that the connection is not always mandatory. This does not imply, that all other concepts must always contain some information. For example a backlog item with blank description can exist. The backlogs and the task list are not listed as separate main requirements, but as sub-requirements of the corresponding time horizons.

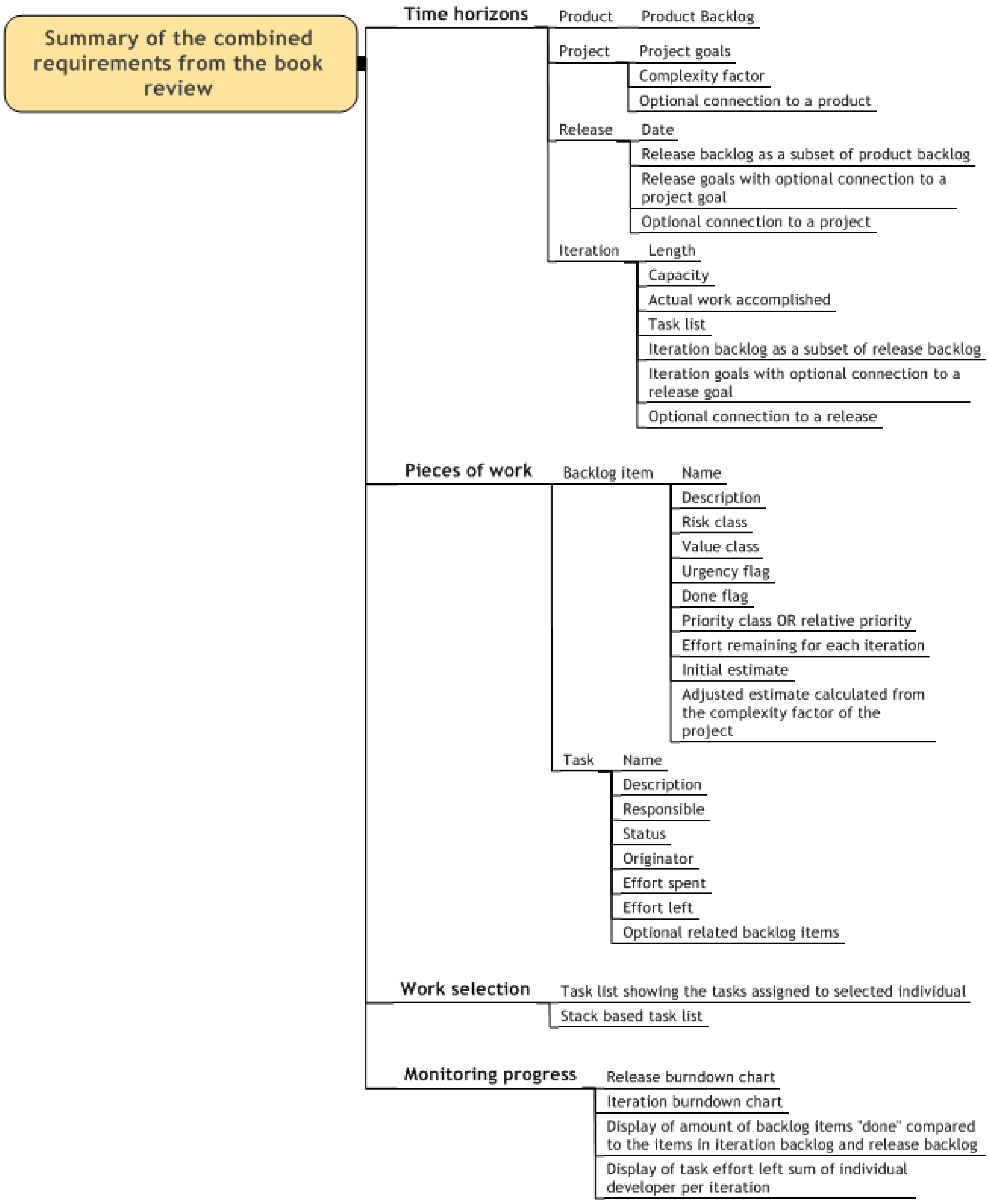

Summary of the combined requirements from the book review
Time horizons
Product
Product Backlog
Project
Project goals
Complexity factor
Optional connection to a product
Release
Date
Release backlog as a subset of product backlog
Release goals with optional connection to a project goal
Optional connection to a project
Iteration
Length
Capacity
Actual work accomplished
Task list
Iteration backlog as a subset of release backlog
Iteration goals with optional connection to a release goal
Optional connection to a release
Pieces of work
Backlog item
Name
Description
Risk class
Value class
Urgency flag
Done flag
Priority class OR relative priority
Effort remaining for each iteration
Initial estimate
Adjusted estimate calculated from the complexity factor of the project
Task
Name
Description
Responsible
Status
Originator
Effort spent
Effort left
Optional related backlog items
Work selection
Task list showing the tasks assigned to selected individual
Stack based task list
Monitoring progress
Release burndown chart
Iteration burndown chart
Display of amount of backlog items "done" compared to the items in iteration backlog and release backlog
Display of task effort left sum of individual developer per iteration

Figure 4.7: Summary of the requirements from the book review

Chapter 5

Requirements for a tool for the case company

This chapter presents the requirements for a tool for the case company. See Section 2.3 for information on the case company. Section 5.1 presents the concept types that were identified in the case company. Section 5.2 describes the time horizons that were identified in the case company. Section 5.3 describes the containers of work identified in the case company. Section 5.4 describes the pieces of work identified in the case company. Section 5.5 presents the conceptual map of the identified case company concepts. Section 5.6 describes the selection of work requirements from the case company. Section 5.7 describes the work monitoring requirements of the case company. Section 5.8 describes the requirements the case company has for views in a tool. Section 5.9 describes the requirements the case company has for work-hour reports. Finally, Section 5.10 presents a summary of the requirements from the case company.

As in Chapter 4, creating a new instance of a concept and deleting an instance are implicit requirements and are not repeated for each concept described in this chapter.

5.1 Identified concept types

The concepts in the conceptual model based on the case company are mostly identical to the concept types presented in Section 4.1. There are however two additions to the concept types. The first one allows accommodation of the two priority classes that the concepts are divided into. These two priority classes are *mandatory* and *nice to have*. For the purpose of making the rest of this chapter easier to read, these two priorities will be denoted by 1 for mandatory requirements and by 2 for nice to have requirements. In the concept map concepts that belong to the priority class 1 have solid borders and colored background and concepts that belong to the priority class 2 have dash line borders and plain background. The second addition is an extension to the piece of work concept type. The original concept type presented in Chapter 4 covers pieces of work that need to be done in the future. This concept type is extended to also cover pieces of work that record work that has been done in the past. There is only one concept of this extended type in the map of the identified concepts of the case company; It is the *effort spent entry* piece of work. This extended concept type is identified by colored background with no borders.

5.2 Time horizons identified in the case company

Two priority 1 time horizons were identified in the case company. The first identified time horizon is the *project* and the second is the *iteration*. Both of the identified time horizons share many aspects, but serve a different purpose in the case company.

The project time horizon is a long-term planning tool. As the case company produces both software products and software services, the projects vary notably in type. Some types of projects are not projects in the traditional sense of the term, as they might not have a fixed set of resources or a deadline. Such projects include semi-continuous bug fixing in the product platform or continuous enhancement of documentation. Grouping such activities under the term project is practical as it removes the need for a separate container for work that is not related to any traditional project. The project time horizon is also used in the traditional sense of the term for projects such as customer specific software development projects, which have a fixed set of resources and a deadline.

The project time horizon has a set of responsible persons, a deadline, a number of sold hours and a number of planned hours as priority 1 properties. All of these can be empty. If the project was sold to a customer, the number of sold hours contains information on how many work hours were sold to the customer of the project. The number of planned hours shows how many hours were actually planned for the project. Projects usually last for at least several months and can last for more than a year. The project time horizon has also one priority 2 property, the project type. This property makes it possible to classify the project into one of the types defined in the case company. The selection of types is not fixed and can changed when required. The project has two priority 1 containers of work attached to it. The first one is the requirements in the project, which contains the requirements that must be implemented during the project. The second container is the tasks in the project, which contains the tasks that have been generated for the project.

The iteration time horizon is a short-term planning tool. Like the project, the iteration has a fixed deadline as a priority 1 property. Iterations are always done in the context of a project and projects are usually divided into several iterations. The length of an iteration varies depending on the project, and can be as long as four months. The end of an iteration usually marks some important date, such as an agreed delivery point of the software developed in the project.

The iteration time horizon has one priority 1 container of work and one priority 2 container of work. The priority 1 container of work is the tasks in an iteration, which contains all tasks that should be performed in an iteration. These tasks are a subset of the tasks in a project to which the iteration belongs. The priority 2 container of work is the requirements in the iteration. The requirements in this container are a subset of the requirements of the project the iteration is a part and will be implemented in the iteration.

5.3 Containers of work identified in the case company

A total of four containers of work divided in to two different types were identified in the case company. Three of the containers are priority 1 requirements and one is a priority 2 requirement. The first type of container contains tasks and both of the containers of this type are of priority 1. The *tasks in a project* contains tasks that have been devised to be implemented in the project to which the container belongs. The *tasks in an iteration* container contains tasks that have been selected from a tasks of a project container to be implemented in an iteration or are independently devised to be implemented in the iteration. If tasks have priorities, which is priority 2 requirement, then the tasks in either container must be ordered in relative priority order.

The second type of container contains requirements. The *requirements of a project* is a priority 1 container of work. It contains all requirements that must be implemented in a project. The *requirements of an iteration* is a priority 2 container of work. It contains requirements that are selected from a project to be implemented in the iteration to which the container belongs.

Later in this chapter the reference to the container of work a task or requirement belongs to is often omitted for the sake of readability. As each time horizon has only one of each type of container of work, referencing a container of work by the name of the time horizon to which it belongs does not cause ambiguity.

5.4 Pieces of work identified in the case company

Three different pieces of work were identified in the case company. All three are priority 1 requirements. The first piece of work is the *requirement*, the second piece of work is the *task* and the third piece of work is the *effort spent entry*.

The requirement is a piece of work that describes something the system or software must be capable of doing. Requirements are described on a high abstraction level and their actual content may vary greatly in style and detail level. Most often a requirement describes some feature that the software needs to have. The requirement has three priority 1 properties and two priority 2 properties. The properties are summarized in the Table 5.1. The only operations required for requirements are moving them between projects and iterations (if they can reside in iterations) and changing the properties and relationships they have.

Table 5.1: Case company requirement properties

Property	Priority	Definition
Responsible	1	The person who is responsible for the implementation of the requirement
Priority	1	The priority of the requirement on scale of 1 to 3
Links to specification documents	1	Links to documents in external systems that describe the requirement more precisely
Originator	2	The person who is the originator or creator of the requirement
Attached specification documents	2	Links to documents kept in the tool that describe the requirement more precisely

The second piece of work is the task. The task describes something that needs to be done. The properties that belong to the task are summarized in Table 5.2. Tasks may be related to one or many requirements. This relation is priority 1 requirement. Having a task related to many requirements is required, for example, when implementing a task that enables some functionality that advances the implementation of several requirements that depend on that functionality. A priority 2 requirement for the task is relation to several projects besides the project or iteration container to which the task belongs. This enables the tracing of tasks that advance several projects in addition to the iteration or project to which they belong. Priority 1 operations required for the task are moving it between projects and iterations and changing its properties and relationships. The task has also one priority 2 operation: the tool should provide assistance in splitting one task into two or more tasks. The specifics of this requirement are left open. As priority 1 requirement a task also holds information on how much total effort has been spent on it. This information is calculated from the effort spent entries related to the task.

A task may also be related to an issue record in an external issue management system. This is a priority 2 requirement. Issues are special kinds of pieces of work, as they may vary greatly in size and essence. They may be for example bugs or feature requests. Issues may be reported directly by a client to an issue management system and a process completely separate from the normal software requirements management process can be used to manage the issues. While Scrum advocates storing all work that needs to be done in the product backlog [Schwaber and Beedle, 2002, p. 72], the sheer number of issues in a large project would overwhelm this container. For example, the current number of open items in the bug management system of the Firefox project [Mozilla Organization, 2008a] was over 15000 at the time of writing. Managing issues is a large topic in itself and outside of the scope of this thesis. It will not be explored further. For more information on issue management tools see, for example, an article written by Serrano [2005]. In some cases importing an issue from an issue management system to a tool managing the daily work of the software developers may be required. In the case company, the tool used to manage issues is Bugzilla [Mozilla Organization, 2008b]. There is one priority 2 requirement related to this functionality. The tool must be capable of importing bugs from Bugzilla as tasks in the tool. The exact implementation of this is left unspecified. For example it can be done by specifying a bug identifier number or by free text search in the bug database done from the tool.

As effort spent entries (see next paragraph) are always related to a task, a task named Miscellaneous is usually created in the case company for each iteration or project. All effort spent entries that are not related to any real task are related to this task. For example, effort spent reports for quick bug fixes which do not have corresponding tasks are related to this task. The case company did not see a need for any kind of automation or more general implementation of this functionality and no new requirements therefore stem from it.

The third piece of work is the effort spent entry. This piece of work does not describe something that needs to be done. Instead, it records work that has already been done or more specifically how much time the has been spent doing a particular task. The properties of the effort spent entry unit of work are described in Table 5.3. The effort spent entry piece of work also differs from the two other pieces of work in that it does not reside in any container of work. The effort spent entry is always related to a task. Usually creating a new effort spent entry means that some work has done to advance a particular task the entry in related to. Effort spent entries are primarily used to generate effort spent reports (see Section 5.9). The only required operations for the effort spent entry is changing its properties. Effort spent entry's target task can be changed after it has been created. This requirement simplifies correcting errors in task targeting.

Table 5.2: Case company task properties

Property	Priority	Definition
Description	1	A description of what the task consists of
One responsible	1	The person who is responsible for the implementation of the task
Many responsible persons with weights	1	Several persons who are responsible of the implementation of the task with different weights of responsibility; the weights reflect how deeply involved each responsible person is in the implementation of the task
Many responsible persons	2	Several persons who are responsible of the implementation of the task; this property is a subset of the many responsible persons with weights property, but here each person has an equal weight
Overdue flag	1	A task is marked overdue if the deadline of the iteration or project it belongs to has passed and the task has not been implemented
Implementation status	1	The implementation status of the task; status can be one of new, open, reviewable or closed, which are priority 1 requirements or blocked, which is a priority 2 requirement; setting closed status should set the remaining effort automatically to 0
Original estimate	1	The original estimate of the effort needed for the implementation of the task
Remaining effort	1	An estimate of the remaining effort needed for the implementation of the task
Percentage done	1	An automatically calculated estimate of the work completion percent; calculated from the remaining effort and the original estimate; can be over 100% if the original estimate is lower than the remaining effort
Billable flag	1	A task is marked billable if can be billed from a client
Deadline	2	The date when the task must be done
Priority	2	The priority of the task related to the other tasks in the same container of work
Sold effort estimate	2	The effort estimate that was sold to a customer
Creator	2	The person who created the task
Reviewer	2	The person who is responsible for reviewing the task
Reviewed flag	2	A flag that is raised when the task has been reviewed
Change history	2	A log of all changes that have been done to the task or its properties

Table 5.3: Case company effort spent entry properties

Property	Priority	Definition
Amount	1	A number of hours of work that has been performed
Date	1	The date the work was performed
Comment	1	Extra information about the performed work
Work type	1	One of the work types listed in a company wide list of possible work types; work types can be added to and removed from the list when required
Recorder	2	The person recording the performed work
Work performers	1	One or more persons who performed the work; the total amount of work is the amount times the number of persons; used when several people did the same work at the same time for example they participated in the same meeting

5.5 Concept map of the identified concepts in the case company

Figure 5.1 presents the concept map of the concepts identified in the case company. See Section 5.1 for explanation of the different types of concepts in the concept map. The individual concepts and the relationships between them are explained in Sections 5.2–5.4.

5.6 Selection of work in the case company

The selection of work in the case company is done on two levels. The first level is the selection of the next requirements which are going to be implemented. Information about the requirements is stored in a folder structure in a shared network drive. The selection of the requirements that are going to be implemented next is done in meetings with customers and in meetings among the different stakeholders in the case company. Usually the implementation order of the requirements is directly or indirectly based on the wishes of the customers, but dependencies between different requirements can affect the selection. Only requirements for a tool for managing the daily work of the software developers stemming from the selection of the next requirement to implement are views that show what is the current status of the requirements that are being developed. See Section 5.8 for descriptions of these requirements.

The second level of work selection is the task level. Usually the broader guidelines for task selection are laid out in a team meeting or in discussions with the team leader. The broader guidelines are based on many factors like what work is currently unfinished, what kinds of dependencies the tasks have, what the customer would like to have next and which tasks are most profitable to execute. The case company does not however see a need for the dependencies and customer wishes to be stored to the daily work management tool. Tasks that originate from customers may be stored in a spreadsheet and discussed by the

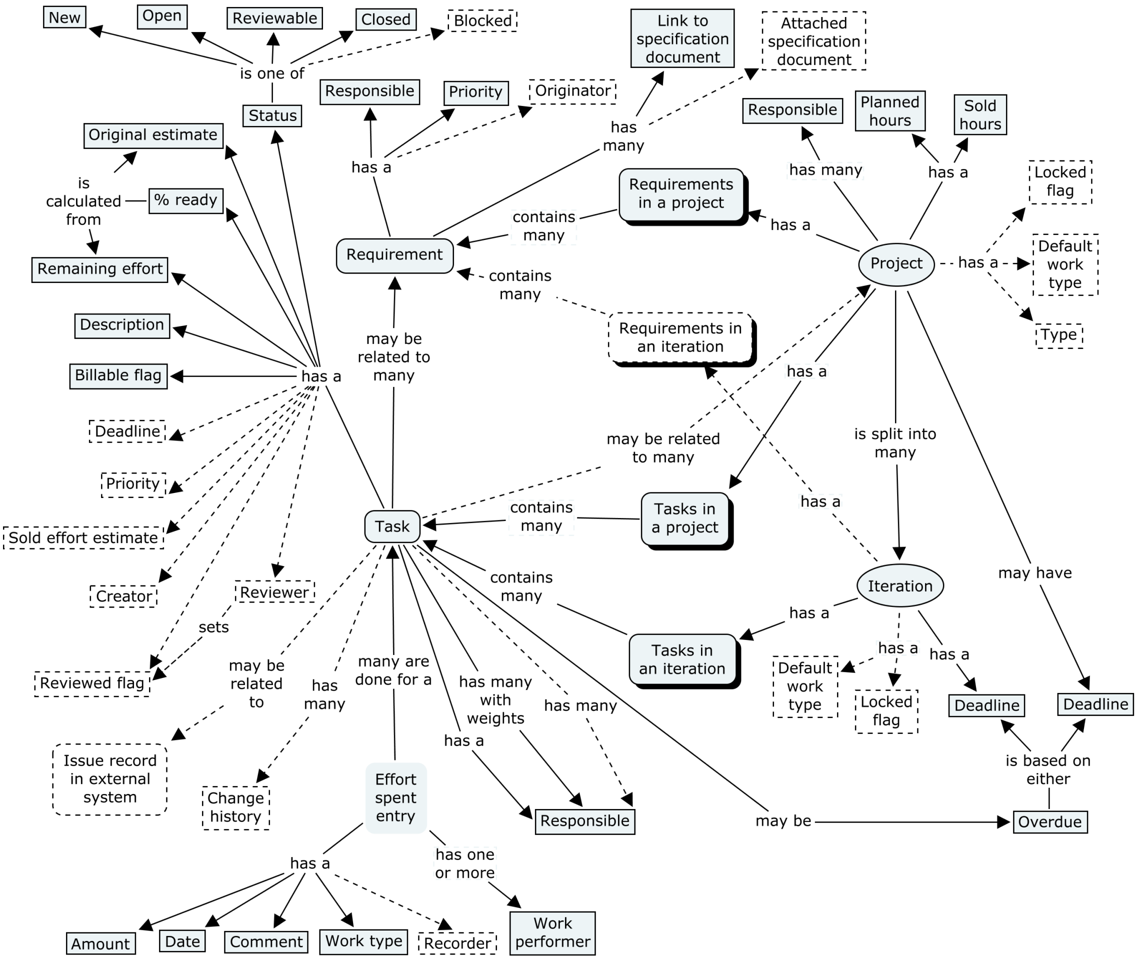


Figure 5.1: Case company concept map

team responsible of implementing them before they are stored into the current daily work management tool.

An individual programmer usually selects the next thing he should do from the list of tasks in the ongoing iterations. He may also check from the tool what he did during the previous day or week for reference. The current tool shows a developer all tasks that have been assigned to him and the developer picks the task to advance based on what tasks are started but incomplete, urgent or not started. The current tool has the capability of emphasizing urgent tasks, however this functionality is mostly not used and the urgency of a task is usually based on the best judgment of the developer. Due to the small size of the development teams in the case company, tasks do not have a priority. The best judgment of a developer is regarded sufficient for selecting the top priority task. Requirements related to the selection of the next thing a developer should do focus on the views that the tool has. These requirements are elaborated in Section 5.8. In addition to the views, one priority 2 requirement related to the selection of work exists. The tool must notify the person who is set as the responsible for a task by e-mail. Specifics such as whether the e-mail in sent immediately or as a digest and other notification preferences are left unspecified.

In addition to the tasks in the current tool, developers are also assigned work more informally. This work is most often assigned face to face, by phone or by e-mail. Usually this means that an urgent issue needs to be fixed as soon as possible. The developers do not usually create separate tasks for such work. If an effort spent entry must be created for such work, it is attached to Miscellaneous task in the iteration or project (see Section 5.4).

5.7 Monitoring progress in the case company

The main way to monitor the progress of work of an individual programmer in the case company is the regular meetings held by the development teams. In these meetings each team member describes what is the status of the tasks that he is responsible for, what he is going to do next and whether there are any problems that need to be solved. The past effort spent entries can be used to review what the programmer has done previously.

The progress of work on individual tasks can be monitored from the effort spent entries of the tasks. The current tool also emphasizes overdue tasks, which are tasks that are not done when their iteration's deadline has been passed. While the current tool supports effort left estimation on tasks, this functionality is mostly not used and thus cannot be used in monitoring the progress of work on tasks. The progress of a project as a whole is monitored informally in meetings.

The progress of work in an iteration is monitored with an *iteration burn-down graph*, which is a priority 1 requirement. The iteration burn-down graph is similar to the sprint burn-down graph used in Scrum and described in Section 4.7.1. The vertical axis shows the effort left, while the horizontal axis shows dates.

Requirements related to monitoring the progress of work in the case company are focused on the views the tool supports. These requirements are described in Section 5.8.

5.8 Requirements for views in the case company

Four priority 1 requirements for views a tool should have were elicited from the case company. The first view is required by software developers to manage their daily work. The view is per-person based; it shows information of one person at a time. The view has 4 priority 1 requirements and one priority 2 requirement. These requirements are summarized in Table 5.4. This view is closely related to the selection of the next thing a developer should do presented in Section 5.6. The requirements for this view do not include the emphasis of tasks that are overdue. The case company did not see a need to emphasize overdue tasks in the list of tasks.

Table 5.4: Requirements for the developer daily work view

Requirement	Priority	Definition
List of assigned tasks	1	A lists all tasks that are assigned to the selected person
List of tasks to review	1	If tasks are assigned reviewers (see Table 5.2), this list contains all tasks that the selected person is assigned to review
List of effort spent entries	1	A lists all effort spent entries the person has created during the current week and the previous week
Effort spent calendar	1	A calendar from which a date can be selected; all effort spent entries created for the selected date are then displayed
Effort left per iteration	2	A list of per iteration sums of effort left estimates for all tasks the selected person is assigned responsible to.

The second view shows a division of remaining work split in time horizons and persons. It is intended for project managers or team leaders. It shows the amount of work remaining for each person in each ongoing iteration or project. This is the only priority 1 requirement for this view. The amount of work remaining for each person is calculated by summing the effort left estimates of each task for which the person is responsible. Table 5.5 shows an example of how this view could look. The exact way the data should be presented in this view is left unspecified and the figure is strictly an example.

Table 5.5: Example of work division view

	Project 1			Project 2		
		Iteration 1	Iteration 2		Iteration 1	Iteration 2
Person A	190	150	40	30		30
Person B	30		30	150	40	110
Person C	230	100	130			
Person D				230	145	85
Person E	20		20	120	60	60
Total	470	250	220	530	245	285

The third view is meant to be used in status meetings of software development teams. The team status meetings in the case company are arranged so that each member of the team tells what they have done since the last meeting, what they are going to do next and what problems they have if any. This convention is very much like the daily scrum meeting practiced in Scrum [Rising and Janoff, 2000, p. 31], but done on a weekly scale instead of daily. The view has only one priority 1 requirement. The view is required to show the effort spent entries of the selected person that have been created for the tasks assigned to him during a time period. The time period must be freely selectable.

The fourth view is a management team view for the upper management of the case company. This view shows each of the ongoing projects, the number of hours budgeted for each of the projects, the number of hours sold for each of the projects and the number of hours that have been reported for each of the projects (in effort spent entries).

In addition to these particular views, views for viewing and editing different concept types must exist. For example, a view that shows all tasks in an iteration could exist. The exact contents and layouts of these views are not specified, as these requirements are covered by the requirements of editing the different concepts, properties and relationships.

5.9 Work-hour reports in the case company

A set of requirements for spent effort reporting functionality was elicited from the research in the case company. Since much of the information leading to this set of requirements originated from the time sheets specification meeting (see Section 2.5.2), these requirements are not prioritized like the other requirements in this chapter and should be considered less reliable. The time sheets specification meeting also introduced some new requirements to the concepts presented previously in this chapter. These requirements are handled separately in this section.

The first requirement for effort spent reporting is a report that shows the total amount of hours reported for a selected project or iteration. This number is calculated from the effort spent entries of tasks that are located in the iteration or project. In addition to the total number of hours, separate values should be shown for effort spent entries that are marked billable and for those that are not. For projects that contain iterations the totals for the project and for each iteration belonging to the project are shown. The second requirement is a report that shows the total number of hours for a selected person in a selected time period. The hours should be shown separately for different projects, iterations and work types. The third requirement is a report that shows how much a person has spent effort in each work type in a selected iteration or project.

There are two new requirements for concepts related to work-hour reports. The first requirement is a *default work type* for projects and iterations. Whenever a new effort spent entry is created, this work type is the default work type for that effort spent entry. This reasoning behind this requirement is, that iterations or projects in the case company some times have one work type that dominates the work done in the project or iteration. Selecting the same work type each time from a long list of possible work types quickly becomes tedious. The second requirement is, that it must be possible to lock projects and iterations. Effort spent entries cannot be changed or created to tasks in locked iterations and projects. Some times employees in the case company were not certain of which iteration or project work hours should be reported to, especially in the case of consecutive iterations in the same project. If a project manager can lock past iterations and projects, reporting work hours to wrong target becomes less likely.

5.10 Summary of the requirements for the case company

The requirements from the case company are summarized in Figure 5.2. The requirements are presented in a tree structure. The first level branches group different types of requirements together, while the deeper branches represent requirements and sub-requirements. See the corresponding sections in this chapter for more information on each individual requirement. The numbers preceding each requirement are equal to the priority numbers introduced in Section 5.1. The requirements identified in the work-hour reporting meeting (see Section 2.5.2 are identified by a clock icon. These requirements have no priority. The containers of work are not listed as separate main requirements, but as sub-requirements of the corresponding time horizons. As the containers presented here have no properties associated to them and are only related to the pieces of work and time horizons, this simplifies the presentation of requirements. Changing the properties and relationships the concepts have are not listed as separate requirements. They are fundamental implicit requirements and are expected to be implemented for every concept.

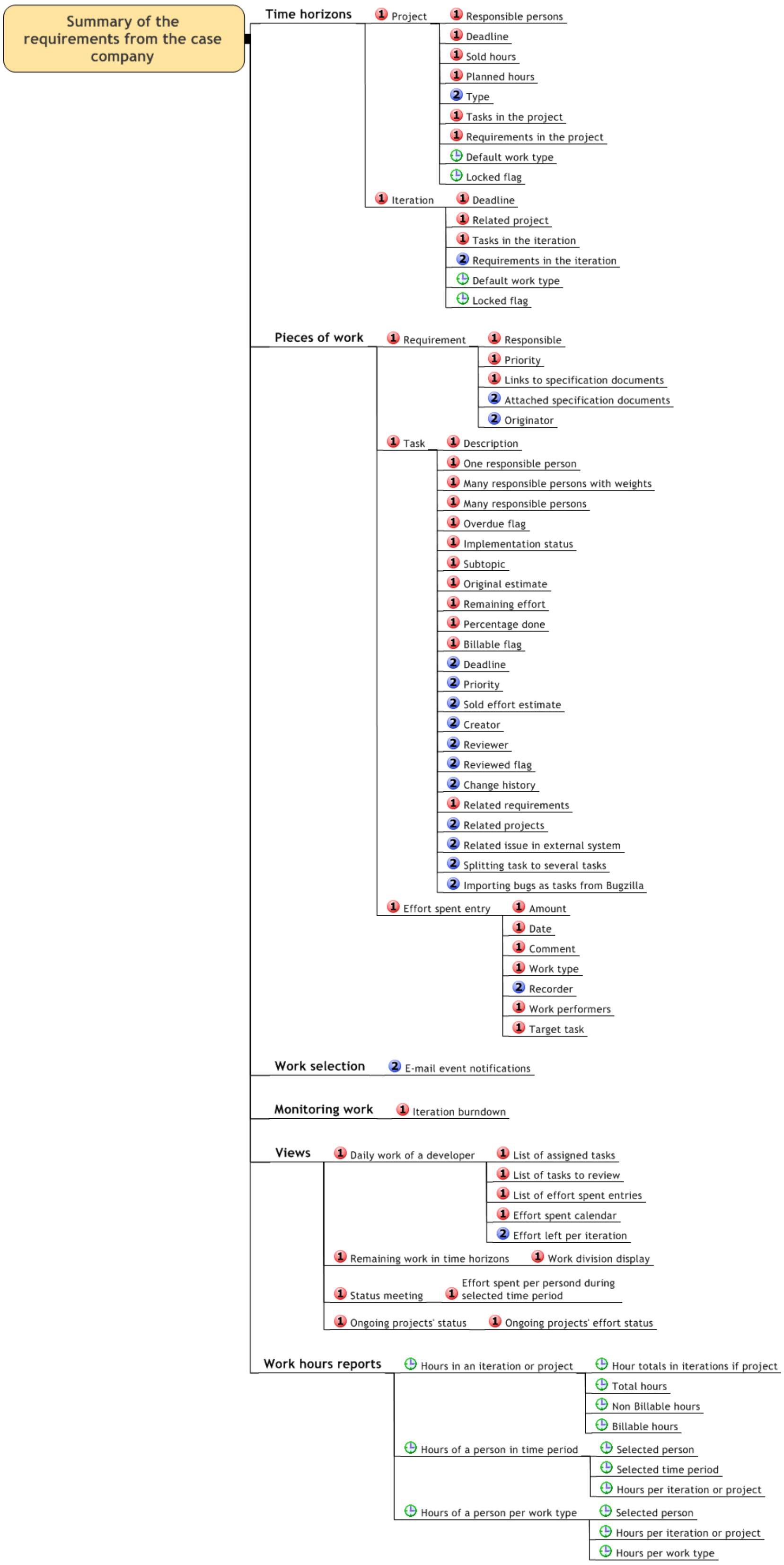


Figure 5.2: Summary of the requirements from the case company

Chapter 6

Results of the tool review

The results of the tool review are described in this chapter. When describing the features that a tool has, only the features that are relevant to the book review or case company are included. Every tool reviewed in this chapter has features in addition to those described, but the additional features provide no added value, as we are only interested in how well the tools fill the requirements from the book review or case company. When a tool is said to lack some feature, it should be interpreted in the context in which it is stated. The most common situation is that, in some sense, the tool in question might have the feature, but using the feature would require something that is in conflict with the requirements the tool is compared to. Section 6.1 presents the results for Agilefant, Section 6.2 presents the results for Mingle, Section 6.3 presents the results for Rally Enterprise Edition and Section 6.4 presents the results for ScrumWorks Pro.

Each section contains one conceptual for each set of requirements. See Section 2.7 for more information on how these maps were created. The concepts in the maps are color-coded depending on how good a fit for the concept can be found in the tool reviewed in the section. The color-coding used in these maps is presented in Figure 6.1. The blue-gray background concepts and black text relationships denote concepts that have a good match in the tool. Yellow background concepts and dark green relationships denote concepts that have an approximate or rough match in the tool. Light red background concepts and orange relationships have no match in the tool. Light blue background concepts and dark blue relationships denote concepts that had to be in order used to enable some required concept but were not a part of the requirements. Dotted lines and box borders denote second-priority case company concepts and a box without borders denotes the effort spent entry piece of work.

In addition, each section contains one additional concept map. These maps present the high-level concepts and relationships using the terms in the tool. The contents of the concept maps are explained in the sections following each of the concept maps. For Agilefant, ScrumWorks Pro, and Rally EE, the maps show how time horizons, containers of work, and pieces of work are arranged in the tool. The conceptual model used in Mingle is highly configurable and the meta-model used in the configuration is shown in the concept map.

When referencing the conceptual maps which contain the conceptual requirements from the book review and the case company, the notion of conceptual requirements is mostly omitted and terms case company requirements for case company conceptual requirements and book review requirements for book review conceptual requirements are used instead.

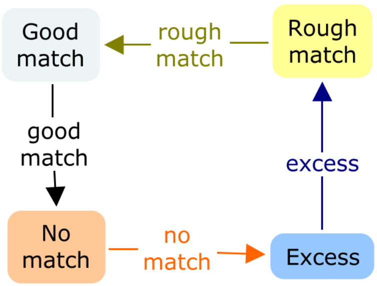


Figure 6.1: Colo-coding of the tool review result concept maps

Since the more general terms encompass both conceptual requirements and other requirements, this should cause no confusion.

When comparing the features of a tool to the requirements from the case company, the priorities of the requirements are omitted from the text. However, the priorities are included as superscript prefixes in the tables that summarize the features in Section 6.5. “^{1}” denotes a priority 1 requirement, “^{2}” indicates a priority 2 requirement, and “^{T}” represents a requirement that is related to the work-hour reporting functionality. See Section 2.5.2 for more information on the priorities.

6.1 Results for Agilefant

This section presents the results of the Agilefant review. For general information on the tool, see Section 6.1.1. Figure 6.3 presents the color-coded book review concept map and Figure 6.4 presents the color-coded case company concept map.

Figure 6.2 presents a high-level conceptual model of the concepts in Agilefant. The later sections describe the mapping of the conceptual and other requirements to the Agilefant features. Results of comparing the Agilefant features to the requirements from the book review and case company are described in Sections 6.1.2–6.1.6. Requirements for views and work-hour reporting are from the case company only and the results for these requirements are described in Section 6.1.7 and Section 6.1.8.

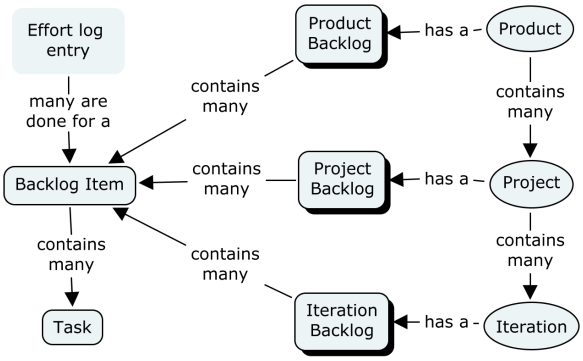


Figure 6.2: High-level conceptual model of Agilefant concepts

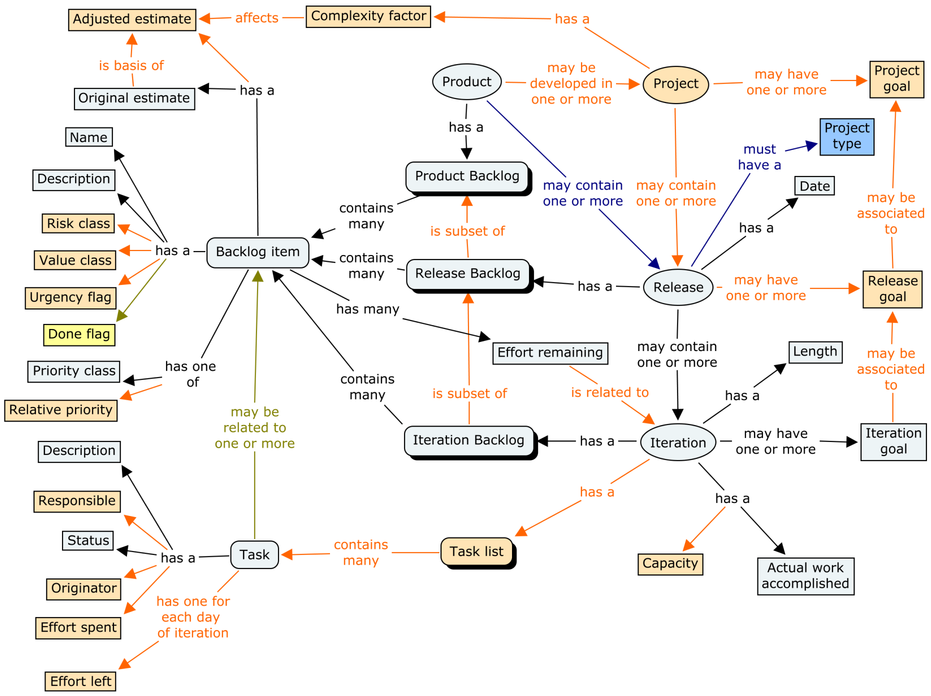


Figure 6.3: Conceptual map of the book review concepts colored by match with Agilefant

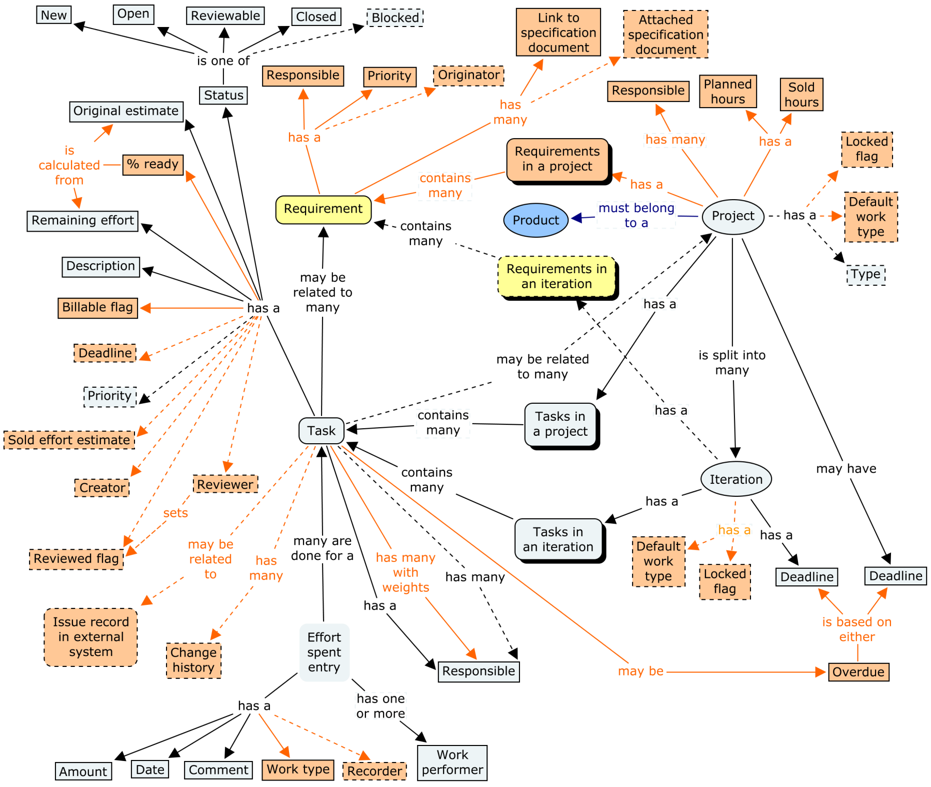


Figure 6.4: Conceptual map of the case company concepts colored by match with Agilefant

6.1.1 Introduction to Agilefant

Agilefant [Helsinki University of Technology, 2008] is a backlog management tool developed by the Software Business and Engineering Laboratory at Helsinki University of Technology. Agilefant is a Web-based tool running on a Web-server and accessed with a Web browser. Agilefant is installed on a Tomcat Web-application-server and it uses an external MySql-server as a database back end. Agilefant's installation guide recommends that given versions of the supporting software are installed and the tool has not been systematically tested in environments other than the recommended one. In practice, this means that when Agilefant is installed into an existing server infrastructure, the existing services in the environment are likely insufficient to run Agilefant and installation of new services is required.

The reviewed version of Agilefant contained features that encompass backlog management, work-hour reporting, work monitoring and short-term planning. Backlogs on product, project and iteration levels contain backlog items. Each product in the tool can have a set of development themes associated with it and these themes can be used to group backlog items. In addition, each iteration can have several goals and backlog items can be assigned to these goals. Backlog items offer such features as prioritization, status, and groups of responsible developers. Tasks in Agilefant represent more fine-grained work. Tasks are always created under backlog items. Work hours can be logged for each backlog item and work-hour reports can be created and filtered by backlog, user, and time span. Agilefant contains a daily work view for developers. This view displays the backlog items which a selected developer is responsible for and also shows how much assigned work the developer has. The product level backlog view offers a road map chart which shows the product's projects and iterations on a time axle.

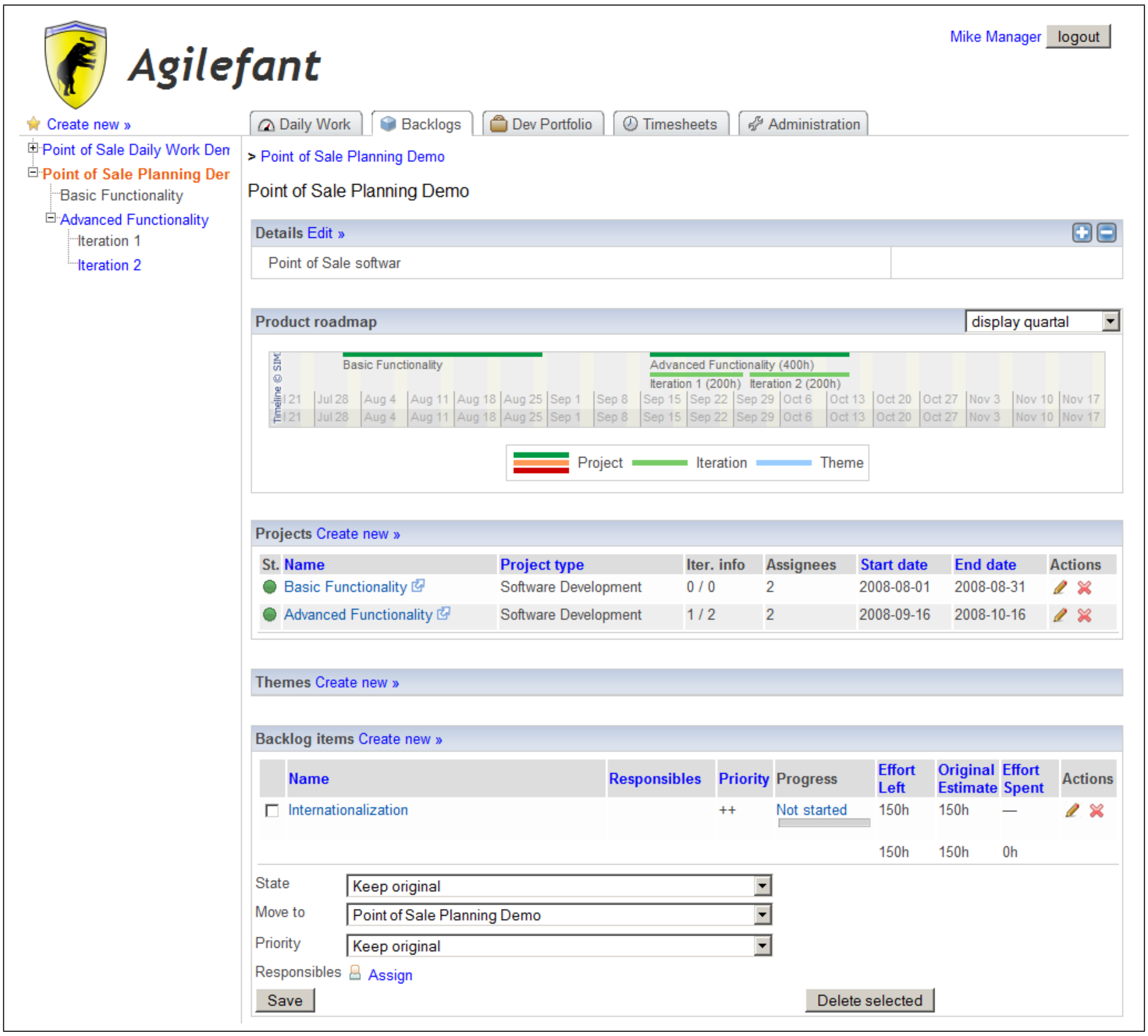
Figure 6.5 presents an example of Agilefant's product backlog view. The tree structure on the left side of the screen shows products, projects and iterations. The currently selected backlog is shown in orange color. The product backlog view covers most of the screen. The drop-down boxes in the bottom of the view enable editing multiple selected backlog items at the same time.

6.1.2 Time horizons in Agilefant

Agilefant has a tree-like structure of time horizons. The tree consists of three levels, which are the product, the project and the iteration. The product is the highest level time horizon and many projects can be attached to it. In turn, the project time horizon in turn may have several iterations attached.

Agilefant's product time horizon matches the product time horizon in the book review requirements. Each product in Agilefant has a product backlog attached to it, which matches the corresponding book review requirement. The product in Agilefant also has a list of projects attached to it. The highest-level time horizon in the case company is the project. However, the rigid tree structure in Agilefant forces the use of the product time horizon.

The project time horizon in Agilefant roughly corresponds to the release time horizon in the book review requirements. No project time horizon as it is presented in the book review requirements exist in Agilefant. The project time horizon in Agilefant is a good match



> Point of Sale Planning Demo

Point of Sale Planning Demo

Details Edit »

Point of Sale softwar

Product roadmap display quartal

Projects Create new »

St.	Name	Project type	Iter. info	Assignees	Start date	End date	Actions
●	Basic Functionality	Software Development	0 / 0	2	2008-08-01	2008-08-31	
●	Advanced Functionality	Software Development	1 / 2	2	2008-09-16	2008-10-16	

Themes Create new »

Backlog items Create new »

	Name	Responsibles	Priority	Progress	Effort Left	Original Estimate	Effort Spent	Actions
☐	Internationalization		++	Not started	150h	150h	—	
					150h	150h	0h	

State: Keep original

Move to: Point of Sale Planning Demo

Priority: Keep original

Responsibles: Assign

Save

Delete selected

Figure 6.5: Agilefant product backlog view

to the project time horizon in the case company requirements. Each project in Agilefant must belong to a product, as projects that do not belong to any product cannot exist in Agilefant. Projects in Agilefant must have a project type, which is selected from a list of available types. The list can be freely edited. This corresponds to the project type property requirement in the case company, but forces the creation of extra concept, when compared to the book review conceptual map. The Agilefant project also has a list of iterations attached to it. Projects in Agilefant must have an end date, which corresponds to the release date in the book review requirements and to the deadline in the case company requirements. While Agilefant mandates the use of the end date, unlike in the case company where it does not have to be set, but the end date can be set to such a distant future date that it does not practically exist. The project in Agilefant has a project backlog container of work attached to it. The project backlog is a rough match to the tasks in a project container in the case company and release backlog in the book review.

The iteration time horizon in Agilefant roughly corresponds to the iteration time horizon in the book review requirements and it is a good match to the iteration time horizon in the case company requirements. Each iteration must belong to a project in Agilefant, which matches the iteration in case company requirements, but does not match the book review requirements where iterations do not have to belong to any project. Iterations in Agilefant must have an end date, which is optional in the case company requirements. As with the project time horizon, this can be circumvented by setting the end date to a distant future date. The iteration in Agilefant has a set of iteration goals attached to it. This container roughly matches the requirements in an iteration container of work in the case company requirements. An iteration in Agilefant may have one or more textual iteration goal attached to it, but releases in Agilefant do not have release goals. Therefore, Agilefant's iteration goals are only a rough match to the book review requirement of iteration goals. The Agilefant iteration also has an iteration backlog attached to it, which corresponds to the iteration backlog in the book review requirements and tasks in an iteration container of work in the case company requirements.

6.1.3 Containers of work in Agilefant

All containers of work in Agilefant belong to the same type and to different time horizons. The containers of work are the product backlog, the project backlog and the iteration backlog. All these containers of work contain backlog items. Backlog items can be moved between the containers of work. However, a backlog item always belongs to exactly one container of work. For example, a backlog item in an iteration belongs to the iteration backlog of the iteration, but not to the project backlog of the project that the iteration is a part of. This corresponds to the way the tasks in a project and the tasks in an iteration work in the case company requirements, but is in conflict with the way the containers work in the book review requirements where a lower-level backlog is a subset of a higher-level backlog.

While Agilefant does not have a specific container of work for requirements, the iteration goals of an iteration can be roughly matched with the requirements in an iteration in the case company requirements. Each iteration in Agilefant can have a set of iteration goals which can be moved between iterations.

6.1.4 Pieces of work in Agilefant

Agilefant has three different pieces of work, which are the backlog item, the task, and the logged effort entry. The backlog item and the task in Agilefant are comparable to their namesakes in the book review requirements. The task in Agilefant does not have a corresponding piece of work in the case company requirements. The piece of work resembling the task in the case company requirements is the Agilefant backlog item. The properties of the backlog item that match the case company requirement's are the original estimate, the effort left, the description and the priority. In addition, the iteration goal piece of work in Agilefant can be roughly used as the case company requirement piece of work.

The backlog item in Agilefant roughly corresponds to the backlog item in the book review requirements. Properties of the backlog item that match book review requirements are the name, the description, the status, the original estimate, and the effort left in hours. The state of a backlog item can be one of several, including "done". The state property of the Agilefant backlog item matches very closely the status property in the case company requirements, as every state has a corresponding status. The "done" status can be used in a fashion similar to the done flag in the book review requirements. The Agilefant backlog item has also a list of tasks attached to it. A similar relationship exists in the book review requirements, but unlike in Agilefant, the task in book review requirements can be related to more than one backlog item or none at all.

The task in Agilefant differs substantially from the task in the book review requirements. Tasks in Agilefant always belong to exactly one backlog item and cannot be moved between backlog items. They do not belong to a separate container of work. A task in Agilefant has a description and a status as properties that match the book review requirements, but it has no other matching properties. The status of an Agilefant task is selected from a list which contains choices that correspond to the choices of status in the book review requirement's task.

The iteration goal in Agilefant and the requirement in the case company requirements seem different on superficial level. However, iteration goals can be easily formed so that they match requirements by stating that the iteration goal is to implement certain requirement. Just as a requirement can be related to many tasks in the case company, an iteration goal can be related to many backlog items in Agilefant. However, the iteration goal does not possess any of the requirement's properties in addition to this relationship.

Effort spent entries for a tasks in the case company can be created as effort log entries for backlog items in Agilefant. Unlike in the case company requirements, the effort log entries are always attached to a backlog item and cannot be moved between backlog items. The properties of effort log entries in Agilefant are a very close match to the requirements from the case company. The matching properties of effort log entry are the amount, the date, the comment, the work type and one or more work performers.

6.1.5 Selection of work in Agilefant

Agilefant has a per-person daily work view which shows the backlog items assigned to the selected person. It also shows the selected person's weekly work load. However, this view does not show a list of tasks assigned to the person, as in the book review requirements. The task in Agilefant does not have a responsible person property. For the same reason, the XP-style, stack-based task selection in the book review requirements is not possible in Agilefant.

Most of the requirements for selection of work in the case company are related to the requirements for views which are presented in Section 6.1.7. The case company has one additional requirement that is related to selection of work; namely, the requirement for the tool to send e-mail notifications of new events. Agilefant does not have this feature.

6.1.6 Monitoring progress in Agilefant

As the task has no effort left estimate in Agilefant, no book review requirements style iteration burn-down chart can be drawn based on the effort left estimates of tasks. Agilefant is capable of drawing an iteration burn-down chart of effort left estimate sums of backlog items plotted on each day of an iteration. This does not match the burn-down chart requirements from the book review, as backlog items in it have only one effort estimate at the start of each iteration and the time scale of a release burn-down chart is in iterations. However, the iteration burn-down chart in the case company requirements is based on the effort left estimates of tasks and corresponds to the iteration burn down chart in Agilefant.

Figure 6.6 shows an example of Agilefant's iteration burn-down chart. The solid red dots show sums of effort left estimates of backlog items in the iteration. The effort left estimate sum of current day is shown with a hollow red dot. The gray line shows a predicted velocity based on the speed of change in the effort left sum. The blue line shows a reference velocity based on linear effort decrement speed.

Agilefant has a view that shows a list of backlog items in an iteration or release. Backlog items in the list are color-coded according to their status. While total numbers of backlog items of different status are not shown, the color-coding can be used as a visual clue to asses the progress of an iteration. This partially fills the second type of monitoring work requirement from the book review. Because the amounts and status of backlog items in a release cannot be compared to those in a iterations belonging to it in Agilefant, the monitoring progress on release level part of the requirement is not met.

Individual developers' progress is monitored in the book review requirements by the sum of the tasks' effort left estimates. This is not possible in Agilefant, as tasks have no effort left estimates.

6.1.7 Views in Agilefant

Agilefant has a daily work view of a developer functionality. This view contains a list backlog items the selected developer is responsible for and a table which shows the effort left sums for the developer spread over the upcoming weeks. This view is the only view feature that corresponds to the requirements of case company for views.

6.1.8 Work-hour reports in Agilefant

Agilefant is capable of creating work-hour reports. A work-hour report can be generated for selected backlogs, a selected time period, and selected persons. The report contains sums of effort log entries for each time period and each backlog item and can be expanded to show details of every logged effort entry created in the selected time period. As the backlog

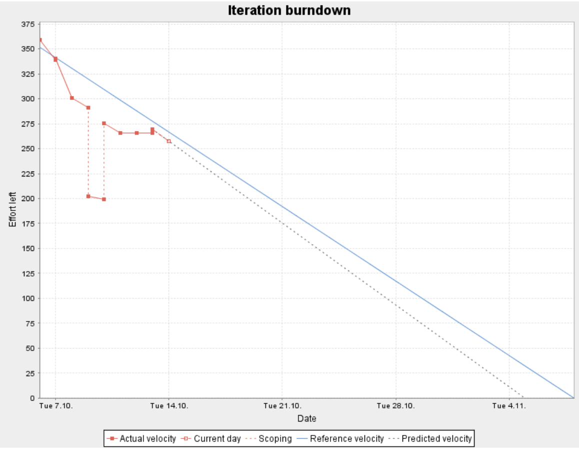


Figure 6.6: Agilefant iteration burn-down example

item in Agilefant does not have the billable flag property and the logged effort entry does not have a work type, the work-hour reports in Agilefant cannot contain this information.

6.2 Results for Mingle

Mingle requires extensive configuration before it can be used in project management. To give the reader of this thesis a better image of how the configuration is done, the conceptual model and configuration of Mingle is briefly described in Section 6.2.1 along with the general introduction to the tool. Figure 6.7 contains a high-level conceptual model of the concepts in Mingle. Unlike the most other conceptual maps in this thesis, the concepts in the map are not categorized in the concept types presented in Section 5.1. Instead, Mingle's card based meta-model is shown.

Conceptual maps colored by how good of a match the concepts have in the tool are shown in Figure 6.8 for the book review concept map and in Figure 6.9 for the case company concept map. See the beginning of Chapter 6 for information on the color-coding.

Results for the time horizons, containers of work, pieces of work, selection of work, and monitoring progress in Mingle, compared to the book review and case company requirements, are presented in Sections 6.2.2–6.2.5. Results that handle requirements exclusively from the case company are found in Section 6.2.6 for the requirements for views and Section 6.2.7 for the requirements for work-hour reports.

6.2.1 Introduction to Mingle

Mingle [ThoughtWorks, 2008] is an agile project management and team collaboration tool from ThoughtWorks, Inc. The tool is Web-based and is used with a Web browser, but it must be installed in to the user's environment as ThoughtWorks does not provide it as a software service. The tool requires an external database service to be available in the installation environment but requires no other external services.

Mingle's conceptual model is extremely flexible and configurable. The tool has three primary concepts which store information, which are cards, properties, and card trees. The

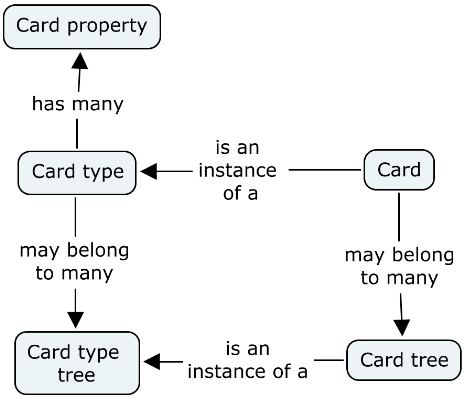


Figure 6.7: High-level conceptual model of Mingle concepts

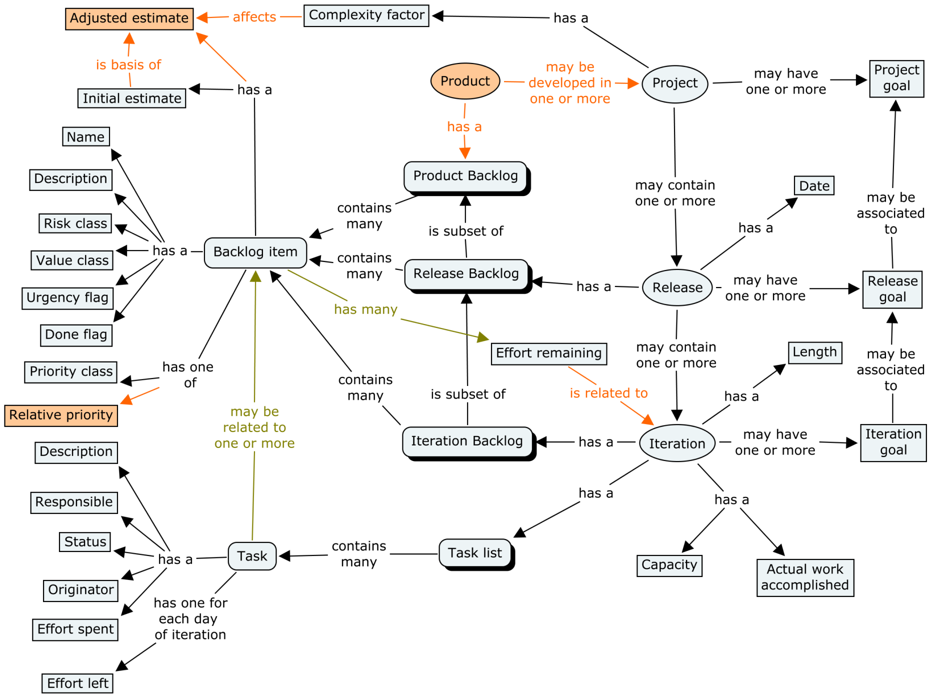


Figure 6.8: Conceptual map of the book review concepts colored by match with Mingle

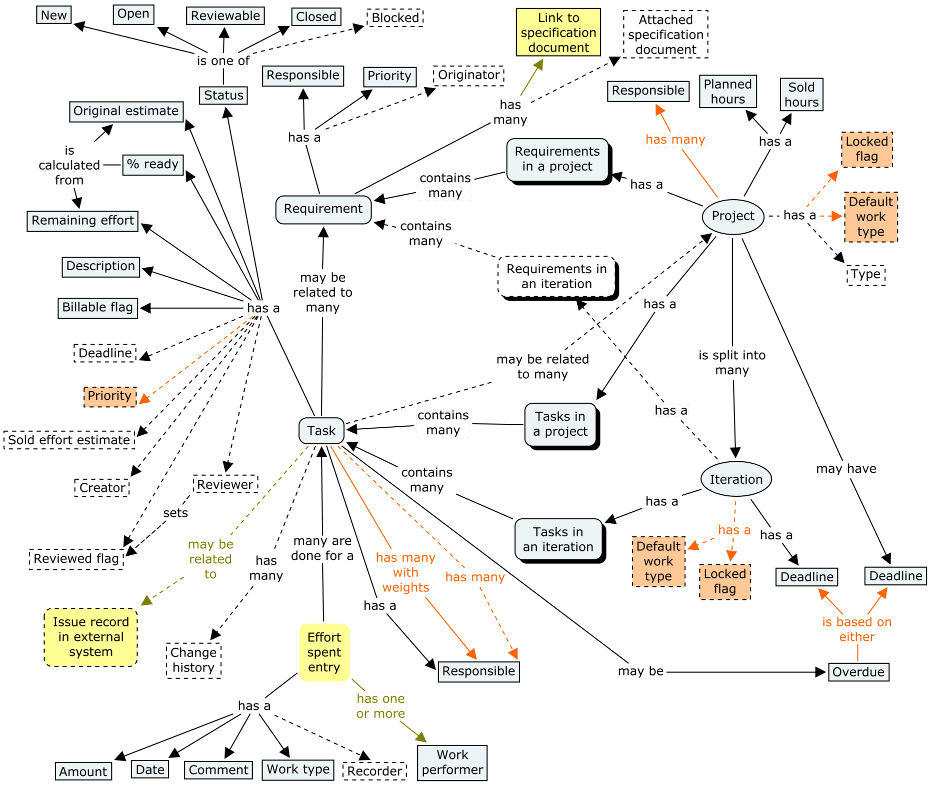


Figure 6.9: Conceptual map of the case company concepts colored by match with Mingle

most important concept in Mingle is the card. Every card has a name and a description. Cards in Mingle are divided into configurable card types and each card in a type has the same configuration. Card types can be configured by creating card properties and assigning them to card types. Many card types can have the same property, which enables filtering many card types with the same property value. The card property types supported by Mingle are a text or number selected from a configurable list, a free text or number, a date, a user selected from the list of users in the tool, or a formula that calculates a value from the other numerical properties of the card type. Each card can have many different properties.

Card trees are formed by linking cards into tree structures. The tree structures are configured by creating card type trees which describe relationships between card types. Each card type can belong to many card type trees. Card type trees cannot contain loops; however, loops can be created by using the same card types in several card type trees. Each card type in a card type tree can have strictly one direct parent type and one direct child type, but the actual cards can skip levels in a tree; for example a card can be direct child of its grandparent type card in a card tree. The number of children a card has in a tree is not limited. Card types that belong to card type trees can have aggregate properties, which are calculated from the properties of it's child cards.

Mingle offers several different ways to display and filter cards. All cards can be viewed as a graphical grid of cards or as a list of cards. Cards displayed in the grid can be filtered, sorted, and colored according to their properties. The list of cards can be filtered by properties and card types and configured to show any property values card types have. Card trees can be displayed as graphical card trees or as hierarchical lists. Cards in card trees can be filtered and hierarchical lists can be configured to show any properties card types have.

Mingle contains a wiki system which allows dynamic content to be placed on wiki pages. Dynamic content can be created using SQL-style language. The language is quite complex, but some functional examples are provided in an example project in a default Mingle installation. Users of the tool cannot be expected to learn the dynamic content creation language to efficiently use the tool. Based on this, the feature descriptions in this thesis concerning the dynamic content creation language are based on the examples given in the example project.

Figure 6.10 shows an example of a card tree view in Mingle. The tabs on the top of the screen are quick links to saved filters and views. The cards in the screen belong to a planning card tree which contains releases, iterations, stories, and tasks. The tree has been filtered to show only release 1 and iterations 1 and 2.

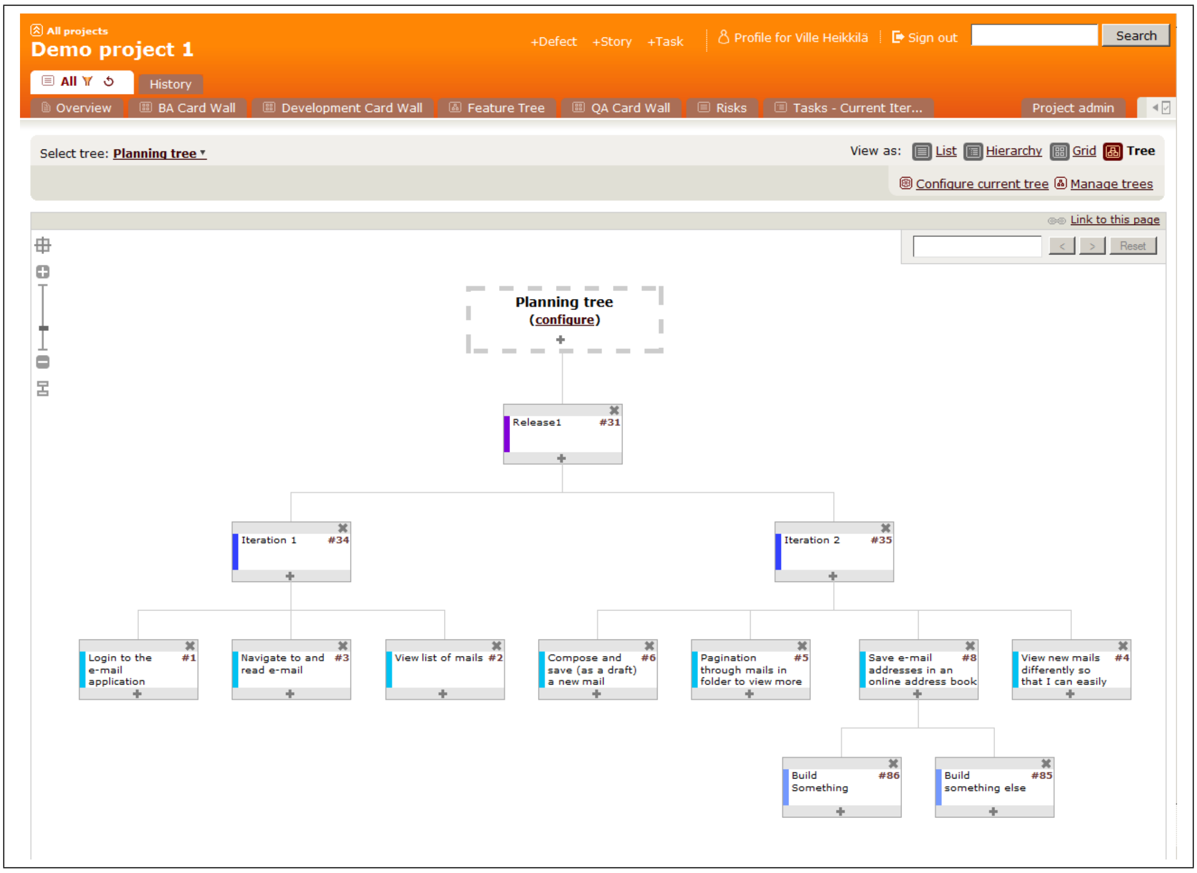


Figure 6.10: Mingle card tree view example

6.2.2 Time horizons and containers of work in Mingle

Time horizons are implemented as cards in Mingle. Card types corresponding to the book review product time horizon, project time horizon, release time horizon, and iteration time horizon can be created in Mingle. Likewise, card types corresponding to the case company project and iteration time horizons can be created.

The containers of work in the book review requirements can be implemented in Mingle by creating a card type tree in which the product is the highest level card type and has project, release, iteration, and, finally, backlog item as hierarchy of child card types. Cards of the backlog item card type can then be located under any higher level node in the tree. This also means that backlog items can be located under projects, which is not a part of the book review requirements. Cards in a card tree always belong to all parents in all hierarchy levels. This corresponds to the book review requirements of backlogs that are subsets of parent backlogs. The task list container of work in the book review requirements can be created in Mingle by creating a card type tree in which the iteration card type has the task as child card type.

Similarly, the containers of requirements in the case company requirements can be created in Mingle by creating a card type tree in which the project card type has the iteration as a child card type and the iteration has the requirement card type as a child card type. The containers of tasks in case company requirements can be created with a similar tree with the exception that the lowest level card type is the task.

The different level goals the time horizons have in the book review requirements can be created as card types in Mingle and the connections from the time horizons to the goals can be created as two-level card type trees in which the time horizon card type is the parent and the goal card type is the child. The hierarchy of the goals can also be created as a card type tree in which the project goal is the highest level card type and has the release goal as a child card type, which, in turn, has the iteration goal as a child card type.

The other properties of the time horizons in the book review and case company can be created as card type properties in Mingle. In the book review requirements these properties are the release date for the release and the length, the capacity, and the actual work accomplished for the iteration. For the project in the case company requirements, the properties are the deadline, the sold hours, the planned hours, and the type. Card type properties in Mingle cannot contain multiple values, so only one responsible person can be appointed to the case company project time horizon card type's responsible property. The work-hour reporting related properties, the default work type, and the locked flag, could be created as properties in Mingle. However, the tool lacks the functionality to employ these values as explained in the requirements (see Section 5.9) and thus, these properties are considered missing.

6.2.3 Pieces of work in Mingle

The backlog item piece of work from the book review can be created in Mingle as the backlog item card type. The properties that can be created for the backlog item card type in Mingle that exactly match the properties of the backlog item in the book review are the name, the description, the risk class, the value class, the priority class, and the initial estimate. The properties urgency and done, which belong to the flag type in the requirements, can be implemented as value properties with two possible values. The priority of the backlog item piece of work can be implemented as a priority class property. As properties can have only one value in Mingle, a backlog item card can have only one effort left value. The project card type card can have the complexity factor property, but this value cannot be used to automatically calculate adjusted estimates for backlog items. A card type tree that contains the backlog item cards type as the parent and the task card type as the child can be created in Mingle.

A task card type corresponding to the book review task piece of work can be created in Mingle. All properties the book review task has can be created in Mingle. These properties are the name, the description, the responsible person, the status class, the originator person, the effort spent value, and the effort left value. As tasks cards are the child node in the card type tree, each task may be related to only one backlog item in Mingle.

The requirement piece of work from the case company can be created as the requirement card type in Mingle. The properties of the requirement card type that match the case company requirement are the responsible person, the priority class, and the originator. Specification document files can be uploaded to Mingle and attached to requirement cards. Mingle does not support URL type properties and, as a result, the links to an external specification requirement can be implemented only as a single non-link free text property.

The task piece of work from the case company can be created as the task card type in Mingle. The task card type's properties matching the case company task's requirements are the description, the responsible person, the status class, the original estimate, the remaining effort, the percentage done, the deadline, the sold effort estimate, the creator person, and the reviewer person. The overdue flag, the billable flag, and the reviewed flag can be implemented as properties with two possible values. Change history is recorded for all cards in Mingle and it corresponds to the case company task's change history requirement. The task's relationship to many requirements can be implemented as a card tree with the task card type as the parent and the requirement card type as the child. Many requirement cards can be connected to one task card using this card tree. Similarly, the relationship between one task and many projects can be implemented as a card type tree which has the task card type as parent and the project card type as the child.

The effort spent entry piece of work of the case company requirements can be created in Mingle as the effort spent entry card type. This card type can be attached to the task card type by creating a card type tree where the task card type is the parent and the effort spent entry card type is the child. The effort spent entry card type can be given the a performer property, but unlike the requirement from the case company, this property can have only one value. The other properties of the case company effort spent entry can be implemented as properties in Mingle. These properties are the amount, the date, the comment, the work type, and the recorder person.

6.2.4 Selection of work in Mingle

Cards can be filtered by the card type or by any property in Mingle. This enables creation of a filter that shows only the task cards assigned to the selected person, which corresponds to the requirement from the book review. A quick link for the filter can be created for quick access. The stack-based task list can be created in Mingle by filtering the cards by status and arranging them in to alphabetical order. The topmost not started task card can be then selected from the list. Mingle does not have the ability to send e-mail when a property value changes, which would correspond to the requirement from the case company. Other work selection-related requirements from the case company are related to views and are described in Section 6.2.6.

6.2.5 Monitoring progress in Mingle

Release and iteration burn-down charts corresponding to the book review requirements and the case company requirements can be created with the dynamic content creation language in Mingle. The language has some limitations; for example, the iteration or release that the burn-down chart is drawn for must be selected from the workspace project properties and card tree relationships cannot be used in selection of cards or properties. Because card tree relationships cannot be used in the dynamic content creating language, a display of task effort left sums of individual developers per iteration cannot be created.

Figure 6.11 shows an example of charts in Mingle. The chart shows the release burn-up of an release. The black line on the top of the chart is the total amount of work that needs to be done in the release and the different colored lines show the amounts of work in different states. The dotted trend lines show the average speeds of increment of the done work in the different states.

Mingle's card filtering and sorting functionality can be used to create a filter which corresponds to the display of "done" backlog items requirement from the book review. Different status backlog items can be colored with different colors and then filtered and sorted based on which release or iteration is their parent in the card tree.

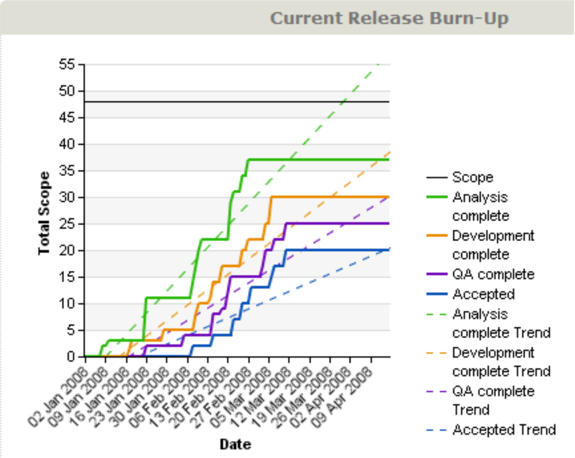


Figure 6.11: Mingle release burn-up example

6.2.6 Views in mingle

The customized filters, views, and dynamic content generation language in Mingle enable creation of customized views and pages which can be saved for later retrieval. However, there are some limitations to the views. The most important limitation, considering the daily work of developer view, is that the required information cannot be combined into one view and several saved card filters must be used instead. Wiki pages created in Mingle cannot contain card filter views, but they can contain many pieces of dynamically generated content.

A card filters showing all task cards a person is set as the responsible, or reviewer can be created and saved. This matches the first two sub-requirements for the daily work of a developer view requirement in the case company. Another card filter that shows the selected person's effort spent entry cards for the selected time period can be created and saved. This view roughly corresponds to the case company requirements for the view showing the current and previous weeks' effort spent entries. The filter also roughly matches the requirement for the effort spent calendar. The dates used in these card filters must be hand-selected each time the filter is used. The effort left per iteration view requirement can be implemented by creating a card list view which shows iterations and their task effort left sums, which can be implemented as an aggregate property in the iteration card type. Similar to the filters described above, the status meeting view requirement can be implemented with a card filter.

The work division view requirement cannot be implemented in Mingle. The tool's data filtering and aggregation features are not powerful enough for the implementation of the view. The card type property aggregation function does not allow conditional operations and each dynamically generated content object can only use single card type as the data source.

The ongoing project's status view requirement is simple to implement with Mingle's card list view by simple filtering by card type and then selecting the required property values to be displayed in the list.

6.2.7 Work-hour reports in Mingle

The work-hour reporting functionality can be implemented in Mingle by summing and filtering effort spent entry cards. Aggregate properties of card types in Mingle cannot use conditional logic when calculating values. Because of this limitation, the work-hour reporting functionality in the tool is very limited when compared to the requirements from the case company. Sum of effort spent entry card's spent work property values can be reported as an aggregate property value of an iteration or project and it is calculated from all effort spent entry cards in the card tree under the project or iteration. None of the more advanced reports can be generated because the tool lacks conditional filtering of aggregate properties. Even if the tool had such functionality, the conditions would have to be changed in the project setting by hand each time a new report needs to be created. The effort spent reporting requirements from the case company the tool lacks are the hours of a person in time period and the hours of a person per work type.

6.3 Results for Rally Enterprise Edition

This section presents the results of the Rally Enterprise Edition (Rally EE) review. For more general information on the tool, see Section 6.3.1. Figure 6.13 presents the color-coded book review concept map and Figure 6.14 presents the color coded case company concept map.

Figure 6.12 presents a high-level conceptual model of the concepts in Rally EE. The later sections describe the mapping of concepts from the requirements to Rally EE features and explain the other features in Rally EE. The results of comparing features of Rally EE to the requirements from the book review and case company are described in Sections 6.3.2–6.3.6. The requirements for views and work-hour reporting are from the case company only and the results for these requirements are in Section 6.3.7 and Section 6.3.8.

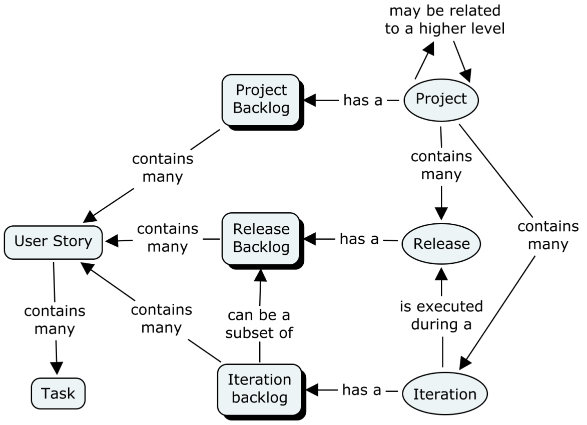


Figure 6.12: High-level conceptual model of Rally EE concepts

6.3.1 Introduction to Rally Enterprise Edition

Rally Enterprise Edition [Rally Software Development, 2008] is an agile software development life cycle management software. It contains functionality for project management and requirements tracking, as well as test and defects tracking. Rally is developed by Rally Software Development Corporation. Limited functionality Community Edition of Rally is also available. Rally Enterprise Edition can be installed to the user's environment. Rally Software also offers Rally as a service hosted on the development corporation's servers.

Rally follows XP- and Scrum-style philosophy to software development management in which software is developed in projects which consist of releases and iterations. Actual work is managed with work objects that are user stories, tasks, and defects, which can be scheduled to releases or iterations. In addition, user stories can form hierarchies. In Rally EE, projects may form hierarchies and one instance of the tool can contain several workspaces. Each workspace in Rally EE is completely isolated from other workspaces and has it's own settings and project hierarchies. Default fields of work objects in Rally EE can be configured to be hidden and new fields can be created. User story prioritization can be configured by workspace to be done either in relation to other backlog items or by a number value.

Rally EE offers many views that display information on iterations and releases. In addition to simply listing all user stories, tasks, and defects, the views can be used to filter and sort work objects. User stories can be moved between projects, releases, and iterations by using a drag-and-drop interface. Rally EE also has a dashboard view which displays status information on iterations, releases, and work objects. It also has a "My Home" view, which can be configured with widgets to show different kinds of information. Some examples of widgets are a list of tasks assigned to the current user and an iteration burndown chart.

Figure 6.15 shows an user story list in Rally EE. The "User Stories" view can show all user stories stored in the tool. The stories can be filtered and sorted by different properties

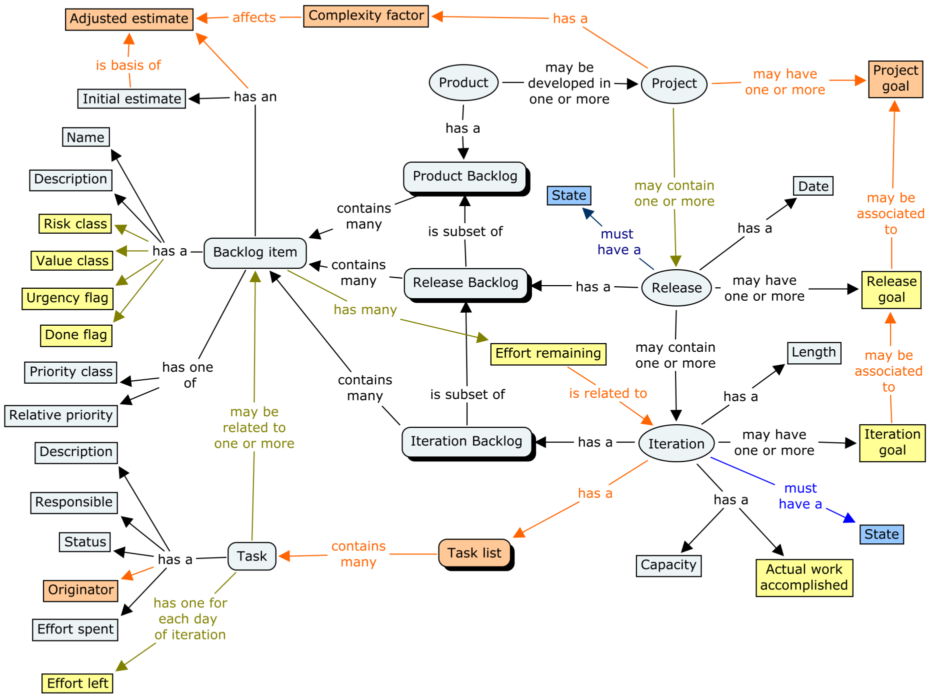


Figure 6.13: Conceptual map of the book review concepts colored by match with Rally EE

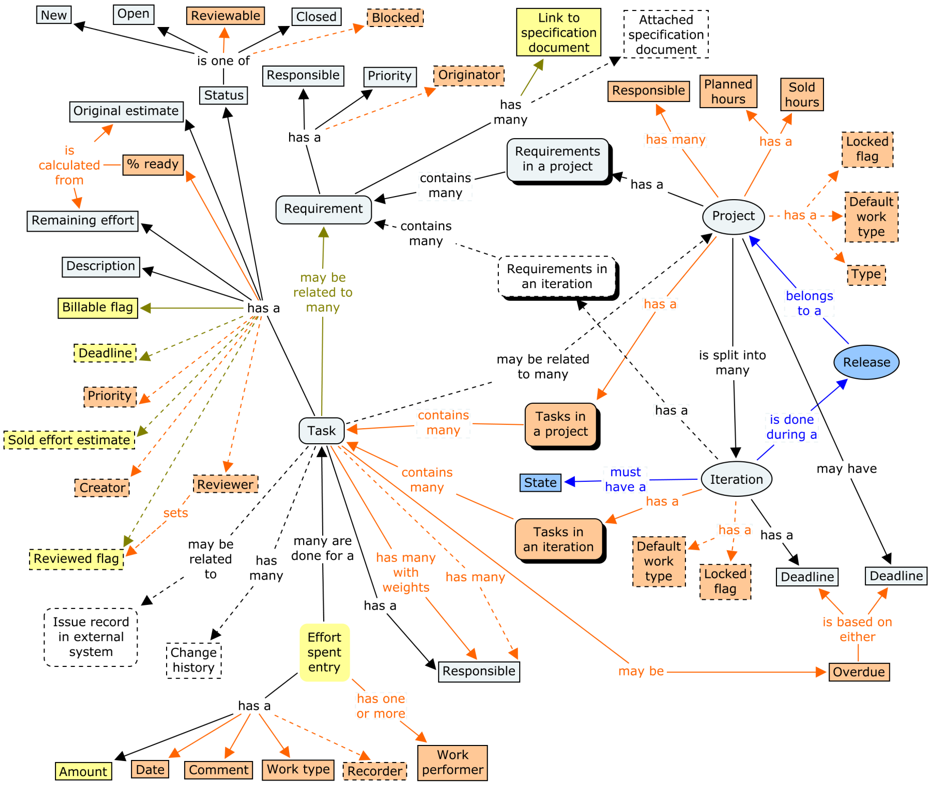
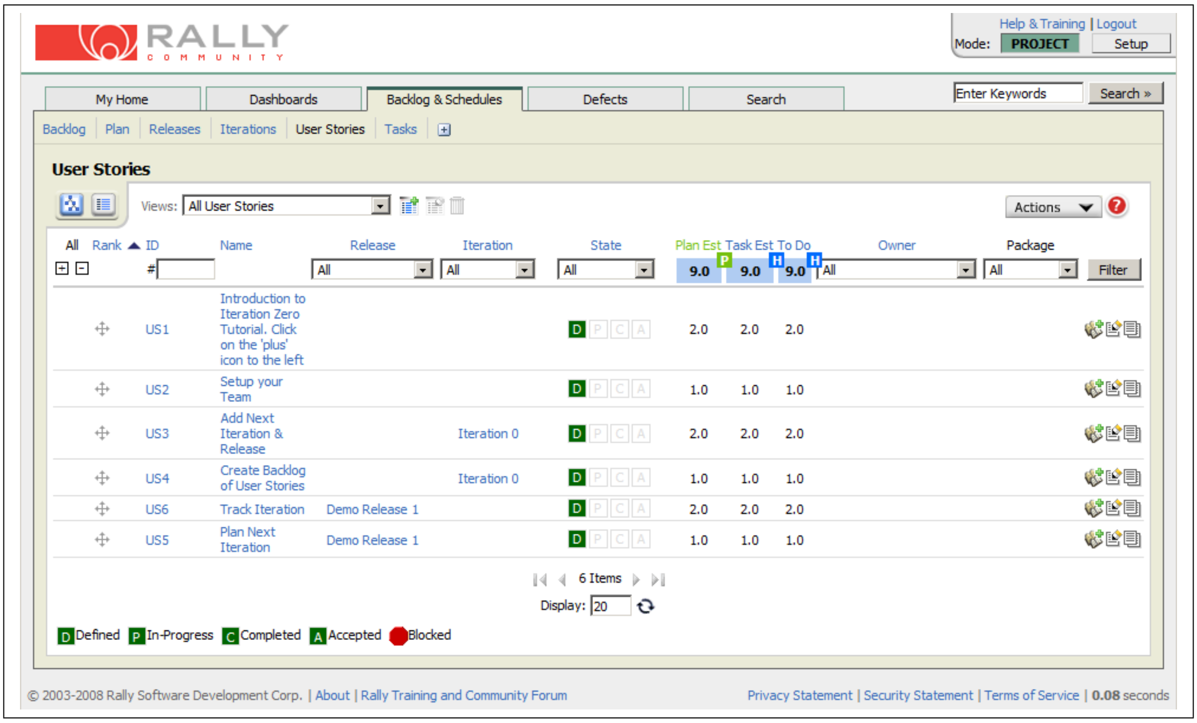


Figure 6.14: Conceptual map of the case company concepts colored by match with Rally EE

and filters can be saved as custom views. The priority order of the stories in the list can be changed by dragging and dropping stories into different locations in the list.



User Stories

Rank	ID	Name	Release	Iteration	State	Plan Est	Task Est	To Do	Owner	Package
			All	All	All	9.0	9.0	9.0	All	All
	US1	Introduction to Iteration Zero Tutorial. Click on the 'plus' icon to the left			D P C A	2.0	2.0	2.0		
	US2	Setup your Team			D P C A	1.0	1.0	1.0		
	US3	Add Next Iteration & Release		Iteration 0	D P C A	2.0	2.0	2.0		
	US4	Create Backlog of User Stories		Iteration 0	D P C A	1.0	1.0	1.0		
	US6	Track Iteration	Demo Release 1		D P C A	2.0	2.0	2.0		
	US5	Plan Next Iteration	Demo Release 1		D P C A	1.0	1.0	1.0		

D Defined P In-Progress C Completed A Accepted ● Blocked

© 2003-2008 Rally Software Development Corp. | About | Rally Training and Community Forum Privacy Statement | Security Statement | Terms of Service | 0.08 seconds

Figure 6.15: User Story list view in Rally Enterprise Edition

6.3.2 Time horizons in Rally EE

Rally EE has three different time horizons, including the project, the release, and the iteration. Projects can form project hierarchies. This enables the creation of an upper hierarchy-level project which corresponds to the product in the book review and a lower hierarchy project which corresponds to the project in the book review requirements. Projects can also be created without hierarchy, which corresponds to the project in the case company. Projects in Rally EE have none of the properties in the case company requirements for the project time horizon.

The release time horizon in Rally EE is a fairly good match to the release time horizon in the book review requirements. Unlike in the book review requirements, releases in Rally EE always belong to a project. A release in Rally EE must have an end date, which matches the date property of a release in the book review. Releases in Rally EE have a free-text theme field, which is a rough match to the release goal book review requirement. However, the themes cannot be associated to project goals, as such property does not exist in Rally EE. Each release in Rally EE must have a state, which is selected from a list containing three options, which are planning, active, and accepted. Release time horizons must be used in Rally EE to create iterations, as a release must be chosen when assigning user stories (see 6.3.4) to iterations. This differs from the case company requirements.

The iteration time horizon in Rally EE is a good match to the iteration time horizon in the book review and case company. The iteration in Rally EE has an end date which matches the length property in the book review and the deadline property in the case company. The iteration can also have themes, which roughly correspond to the iteration goals requirement in the book review. The themes are recorded in a free-text format and thus cannot be linked to release themes. The default work type and the locked flag in the case company requirements are not supported. The iteration in Rally EE has a resources property which roughly matches the capacity requirement in the book review requirements. The unit of resources can be configured to be hours, points, or any other units. The iteration also has an actuals field, which is automatically calculated from the actuals fields of the tasks (see 6.3.4) in the iteration. It only roughly corresponds to the actual work accomplished in the book review requirements, as the value cannot be manually set. Like the release, the iteration in Rally EE must have a state which is selected from the same options as in the release.

All time horizons in Rally EE have a container of work associated to them. The Section 6.3.3 describes the containers of work in Rally EE in more depth.

6.3.3 Containers of work in Rally EE

Rally EE has only one type of container of work, which is the backlog and contains user stories. Tasks, which are the second pieces of work in Rally EE (see 6.3.4), have no container of work, as they always belong to an user story.

Each workspace (see 6.3.1) in Rally EE has one backlog containing all stories that have been created in that workspace. In addition to belonging to the workspace backlog, every user story must also belong to a project. This is a good match to the project backlog in the book review requirements and the requirements in a project in the case company requirement.

Stories in the workspace backlog can be selected into releases, which is a good match to the release backlog requirement in the book review. Stories in the workspace backlog can also be selected to iterations, which is a good match to the iteration backlog requirement in the book review and the requirements in an iteration requirement in the case company.

The containers of work containing user stories in Rally EE are implemented as subsets of higher level containers. Each user story in a lower hierarchy level backlog also belongs to one higher level hierarchy backlog.

6.3.4 Pieces of work in Rally EE

Pieces of work are called work products in Rally EE. Rally has two different kinds of pieces of work, which are the user story and the task. The properties the pieces of work have in Rally EE can be configured on a per-workspace basis. Default properties can be hidden, which hides them from all views in the tool. New properties can also be created and their type may be boolean, date, decimal number, configurable drop-down list, integer, text, string, or Web link. The pieces of work were configured to match the requirements as closely as possible, while any unnecessary default properties were hidden. The values of user created custom properties of work products in Rally EE are not displayed in any work product list and are only displayed in detailed views which show only one work product at a time. This greatly diminishes the usefulness of custom properties and, consequently, the created custom properties are considered only rough matches to the requirements. Tasks in Rally EE have no separate container of work and they always belong to a user story.

The user story in Rally EE is a good match to the backlog item in the book review requirements. The properties of the user story that are a good match to the properties of the backlog item are the name, the description, and the initial estimate. The unit used in the initial estimate can be freely configured. Workspaces in Rally EE can be configured to prioritize user stories in either relative priority order or by decimal priority values, which matches the prioritization requirements in the book review. The done flag requirement in the book review can be matched with a state property of the user story. The state can have several values from “defined” to “accepted”, but using the intermediate values is not required and the two values can be used as true and false values for the done flag. The book review property requirements risk class, value class, and urgency flag can be created as custom properties in Rally EE. However, the values of these properties are not shown on any user story lists. The user story in Rally EE has an effort remaining field, but the field is automatically calculated from the effort remaining values of the tasks belonging to the user story and, consequently, the field is only a rough match to the effort remaining property of the backlog item in the book review requirements.

The user story in Rally EE is a good match to the requirement piece of work of the case company requirements. The responsible and priority property requirements have a good match in Rally EE. User stories can have attached files, which matches the attached specification documents requirement of the case company. A custom URL property can be added to the user story in Rally EE, but the field accepts only well-formed www-addresses and is only a rough match to the links to specification documents requirement in the case company.

The task in Rally EE is a good match to the task in the book review requirements and to the task in the case company requirements. Tasks in Rally EE always belong to exactly one user story, which is against the book review requirements where the task can be related to none to many backlog items or case company requirements where the task can be related none to many requirements.

Rally EE’s task has several properties which are good matches to the book review task property requirements, which are the description, the responsible, the status, and the effort spent. The status of a task in Rally EE can be one of defined, in-progress, or completed, and the three choices match the three choices the task status has in the book review requirements. Tasks in Rally EE have an original effort estimate and a to-do effort estimate. This is only a rough match to the effort left per day in iteration requirement in the book review where tasks have an effort left estimate for each day of the iteration the belong to.

Rally EE’s task has properties that are comparable to the original estimate, the remaining effort, and the description in the case company requirements. The status of the task in Rally EE has three options which match three of the five options in the case company requirements. The custom properties that can be created for Rally EE’s task to match the case company requirements are the billable flag, the deadline date, the sold effort estimate, the reviewed flag, and the issue record in an external system. These are only rough matches because of the weaknesses of custom properties in Rally EE. The issue record in an external system case company requirement can be created as URL property in Rally EE’s task. If the external system records can be accessed with well formed URL consisting of a www-address and identifier parameter, the URL property is a good match to the requirement. Rally EE keeps track of all changes done to tasks and this corresponds to the change history requirement of the case company task. Tasks in Rally EE can have only one person set as responsible. Tasks in Rally EE cannot be prioritized. Rally EE does not provide functionality to assist in splitting tasks.

Rally EE has an add-on which enables importing bugs from Bugzilla. Bugs are imported as issues, which are separate type pieces of work. The imported issues can be assigned to users and iterations. This partially fills the case company requirement of importing Bugzilla bugs as tasks.

The only location in which spent effort can be reported in Rally EE is tasks' actual effort property. No separate piece of work exists that matches the effort spent entry of the case company requirements. The actual effort property contains only one value which must be updated when additional effort needs to be recorded. This property is only an extremely rough match to the case company's effort spent entry piece of work and its amount property.

6.3.5 Selection of work in Rally EE

Rally EE has a "My Home" page which contains widgets showing different kinds of information about the project. The widgets shown on the page can be selected from a palette of available widgets. One of the widgets is called "My Tasks". This widget lists tasks that have been selected to a release or iteration and have been assigned to the current user logged in to the tool. With the exception that tasks that are not in an iteration or release are not shown and the user whose tasks are shown cannot be explicitly selected, this widget matches exactly the book review requirement for the task list showing the tasks assigned to the selected person.

The tasks view in Rally EE shows all tasks of a project and the view can be filtered by an iteration or release. The tasks in the task view can be ordered into alphabetical or internal ID order. Corresponding to the stack based task list requirement of the book review the topmost incomplete task can then be selected for implementation.

Rally EE has notification rules feature which allows e-mail notifications to be sent based on notification rules. The rules can be formed based on work item type and event. A rule can be created that is exact match to the e-mail notification requirement of case company.

6.3.6 Monitoring progress in Rally EE

Rally EE is capable of drawing release burn-down charts. The time period shown in the release-burn down charts in Rally EE is one release and one data point is drawn for each day in the release. The release burn-down charts in Rally EE show two data sets. The first is the planned work and it is calculated by summing the planned estimates of the user stories in the release. The second is the remaining work which is calculated by summing the planned estimates of the user stories in the release which are not in the "accepted" state. The second data set is only a rough match to the release burn-down chart requirement of the book review, as the release burn-down chart in the book review shows the amount of work estimated to be left in the backlog items per each iteration but Rally EE's release burn-down chart only accepts all work left or completely done values for user stories and plots a value for each bygone day of the release.

Rally EE also draws iteration burn-down charts for the iteration time horizon. These charts show three data sets drawn on each bygone day of the iteration. The first data set is a bar chart showing the amount of work in user stories calculated from the planned estimates of the user stories that have accepted status and belong to the iteration. The data set second is a bar graph showing the amount of work left in the tasks that belong to the user stories selected to the iteration. The third data set is the ideal burn-down line, which is a line drawn from the task effort left sum data point in the first day of the iteration to a zero value data point at the last day of the iteration. Figure 6.16 shows an example of Rally EE's iteration burn-down chart. Disregarding that tasks always belong to user stories in Rally EE and that the chart is a bar chart instead of a line chart, the second data set in the chart is a good match to the iteration burn-down chart requirement of the book review and case company.

Rally EE has a view that shows all user stories that belong to the chosen project. This list also displays the iteration or release the user story belongs to and shows the user story's status. This view roughly matches the book review requirement of the display of number of backlog items in different containers and states.

The "My Tasks" widget on the Rally EE's "My Home" view shows the tasks that are assigned to the current user of the tool. The list of tasks shows effort left estimates and the list can be filtered by iteration, which matches the task effort left sum display requirement of the book review.

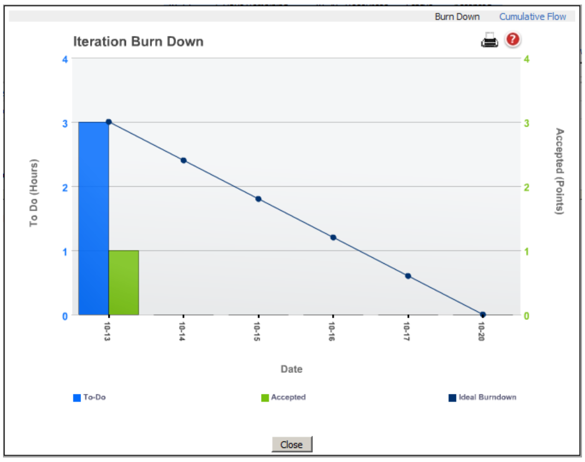


Figure 6.16: Rally Enterprise Edition iteration burn-down example

6.3.7 Views in Rally EE

The developer daily work view requirements that are filled by Rally EE are the list of assigned tasks and the effort left per iteration. These requirements are filled by the "My Tasks" widget described in previous section. As reviewers cannot be assigned to tasks, the list of tasks to review requirement is not filled. The effort spent in Rally EE is only recorded in one field in each task and, consequently, the view requirements of the list of effort spent entries, the effort spent calendar and, the effort spent per iteration are not filled. In addition, the requirement for the status meeting view is not filled.

Rally EE has a view that shows a list of users and the tasks with effort left sums assigned to each user. This list can be filtered by an iteration or horizon. The view is a rough match to the work division view of the case company requirements. Rally EE has no view that is even a rough match to the ongoing projects' status view requirement of the case company.

6.3.8 Work-hour reports in Rally EE

Effort spent can be recorded in Rally EE only in tasks' actuals field. A view containing list of all tasks in a project exists in Rally EE and the tasks on the list can be filtered by iteration, release, and user. The list also shows sums of the actuals of the tasks that are listed. This view correspond to two case company work-hour reporting requirements, which are the hours in an iteration or project and the hours of a person in a time period. The sub-requirements of these two requirements that are not filled are the non-billable hours, the billable hours, and the selected time period hours. The requirements are not filled because the custom properties or a selected time period cannot be used in to filter the list. The hours per person per work type requirement is also not filled because the work type is a custom property.

6.4 Results for ScrumWorks Pro

This section presents the results of the ScrumWorks Pro review. For general information on the tool, see Section 6.4.1. Figure 6.18 shows the color-coded book review concept map and Figure 6.19 contains the color-coded case company concept map.

Figure 6.17 presents a high-level conceptual model of the concepts in ScrumWorks Pro. Later sections describe the comparison of the book review and case company requirements to ScrumWorks Pro's features. Results of the comparison of ScrumWorks Pro's features to the requirements from the book review and case company are described in Sections 6.4.2–6.4.6. Requirements for views and work-hour reporting are from the case company only and the results for these requirements are in Section 6.4.7 and Section 6.4.8.

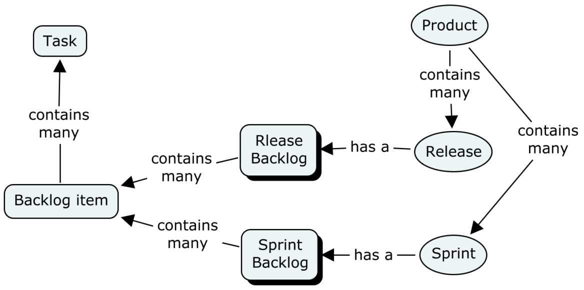


Figure 6.17: High-level conceptual model of ScrumWorks Pro concepts

6.4.1 Introduction to ScrumWorks Pro

ScrumWorks Pro [Technologies, 2008] is an agile software development management tool designed primarily to be used with the Scrum process. The tool is developed by Danube

)

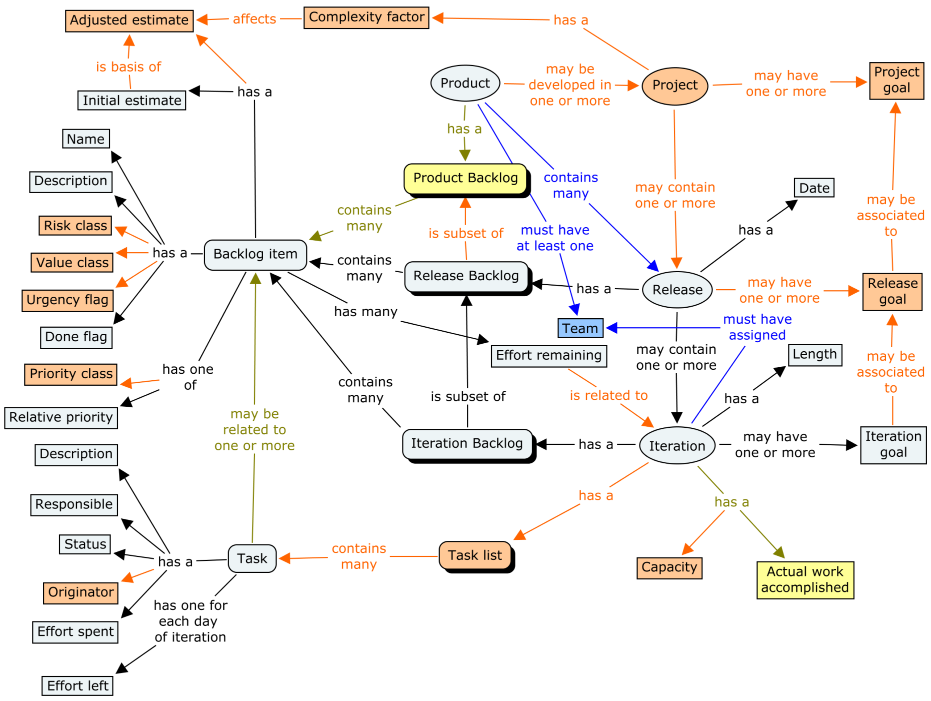


Figure 6.18: Conceptual map of the book review concepts colored by match with Scrum-Works Pro

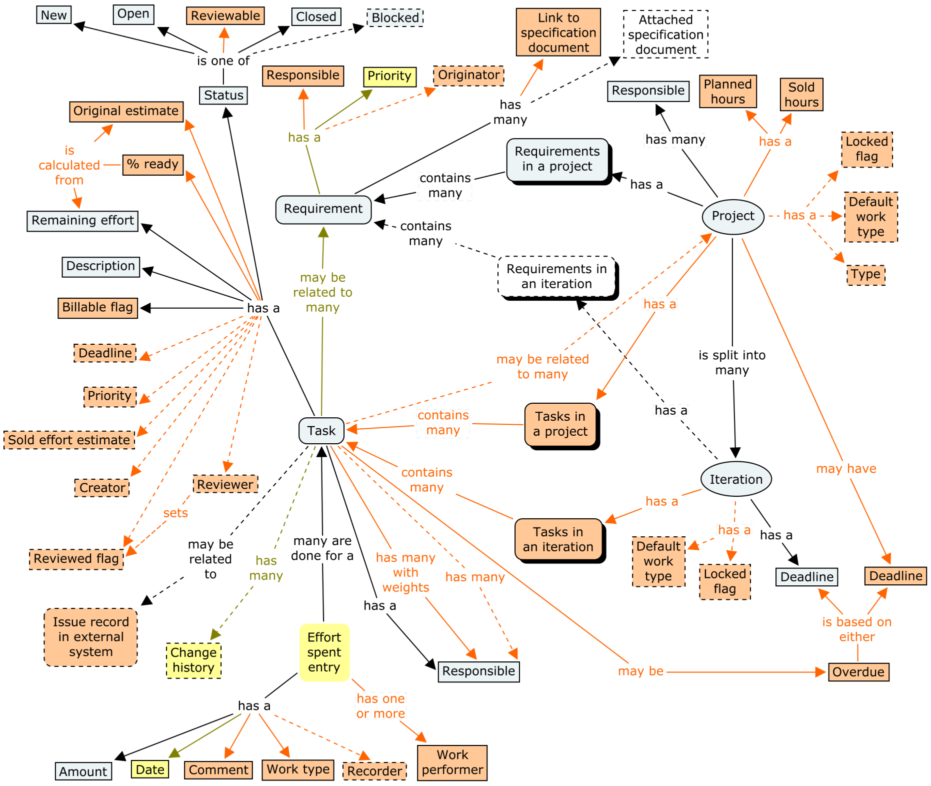


Figure 6.19: Conceptual map of the case company concepts colored by match with Scrum-Works Pro

Technologies, which also provides Scrum training. Limited functionality ScrumWork Basic version of the software is also available for free. ScrumWorks Pro must be installed in the user's environment and is not provided as software service by Danube. The tool requires no external services from the installation environment.

ScrumWorks Pro must be first accessed through a Web page. After connecting to the Web page, the user has two alternative ways to access the software. The first way is ScrumWorks Web access. The Web-based interface provides a list of all backlog items and tasks in a selected product and sprint and allows manipulation of the backlog items and tasks. The tasks that are assigned to the current user of the system can be highlighted in a "My Tasks" view. The view also displays a sprint-burn down chart of the selected sprint. The second way to access ScrumWorks Pro is to use a Java Webstart application, which is a Java application loaded from the Web. Using the Webstart application requires that a Java runtime environment is installed on the client machine. The Webstart application contains much more functionality than the Web-based user interface and is the main way the software is used. It however does not offer a view corresponding to the "My Tasks" view in the Web page interface.

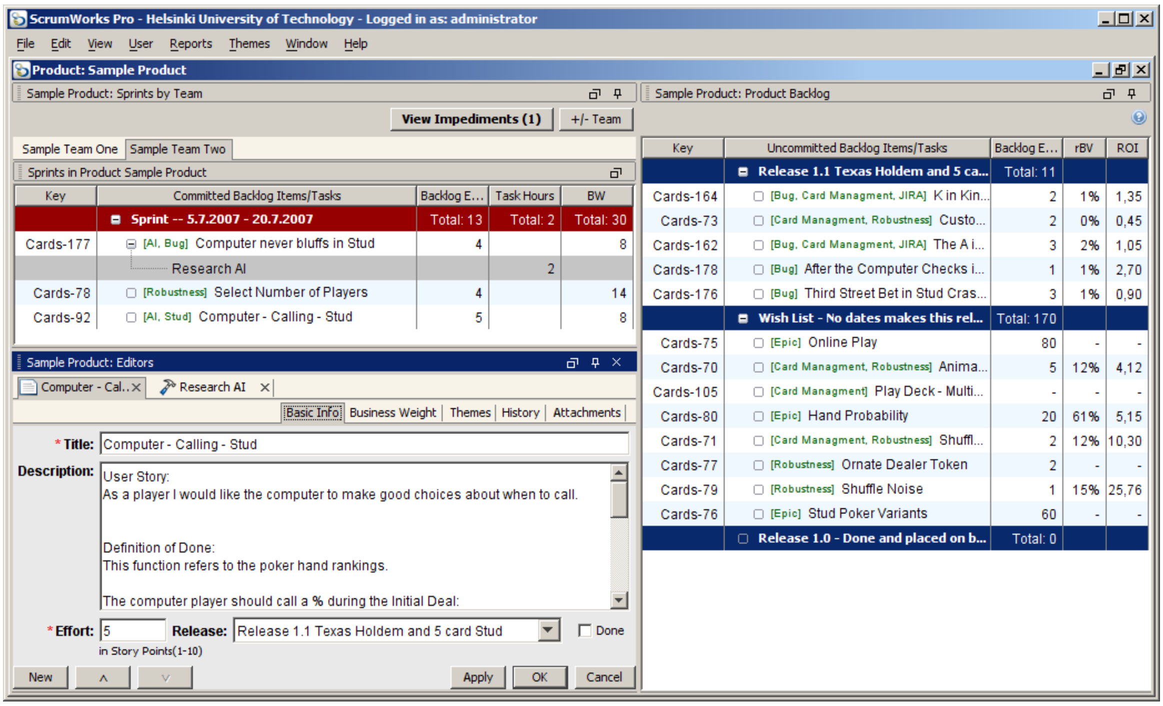
One ScrumWorks Pro server can contain many products at the same time. Each product acts like a separate workspace. Programs that group several products can also be created. Work is planned on high level by creating backlog items. The backlog items can then be assigned to releases and sprints. Low-level work is planned by creating tasks under the backlog items.

Figure 6.20 shows an example of ScrumWorks Pro Java Webstart user interface. The right side of the screen contains the product backlog. All backlog items that have not been selected to a sprint are listed in the product backlog. The product backlog is split into releases. The top left corner of the screen shows sprint backlogs. Backlog items can be dragged and dropped between the different backlogs. The order of the backlog items can also be changed with a drag-and-drop user interface. Tasks are shown as sub-items in the backlog lists. The bottom left corner of the screen contains an edit window. Selected backlog item or task can be edited in this window.

6.4.2 Time horizons in ScrumWorks Pro

ScrumWorks Pro contains three time horizons, which are the product, release, and sprint. These three time horizons are good matches to the product, release and iteration time horizons in the book review requirements. No project time horizon exists in ScrumWorks. The product time horizon in ScrumWorks Pro does not have a backlog attached to it. However, a release backlog that does not have a start or end date can be created and used as the product backlog. This also corresponds to the case company requirement for the requirements in the project container of work. As projects in ScrumWorks Pro have no properties, none of the property requirements for the project time horizon in the case company can be filled. The tool requires that at least one team is created and every product must have at least one team assigned to it.

The release time horizon in ScrumWorks Pro corresponds to the release time horizon in the book review requirements. The release time horizon in ScrumWorks has an end date which matches the date property requirement of the book review, but does not have the release goals property. ScrumWorks Pro's release has a container of work that contains



Key	Committed Backlog Items/Tasks	Backlog E...	Task Hours	BW
	Sprint -- 5.7.2007 - 20.7.2007	Total: 13	Total: 2	Total: 30
Cards-177	[AI, Bug] Computer never bluffs in Stud	4		8
	Research AI		2	
Cards-78	☐ [Robustness] Select Number of Players	4		14
Cards-92	☐ [AI, Stud] Computer - Calling - Stud	5		8

* Title: Computer - Calling - Stud

Description: User Story:
As a player I would like the computer to make good choices about when to call.

Definition of Done:
This function refers to the poker hand rankings.

The computer player should call a % during the Initial Deal:

* Effort: 5 in Story Points(1-10)

Release: Release 1.1 Texas Holdem and 5 card Stud

☐ Done

Key	Uncommitted Backlog Items/Tasks	Backlog E...	rBV	ROI
	Release 1.1 Texas Holdem and 5 ca...	Total: 11		
Cards-164	☐ [Bug, Card Managment, JIRA] K in Kin...	2	1%	1,35
Cards-73	☐ [Card Managment, Robustness] Custo...	2	0%	0,45
Cards-162	☐ [Bug, Card Managment, JIRA] The A i...	3	2%	1,05
Cards-178	☐ [Bug] After the Computer Checks i...	1	1%	2,70
Cards-176	☐ [Bug] Third Street Bet in Stud Cras...	3	1%	0,90
	Wish List - No dates makes this rel...	Total: 170		
Cards-75	☐ [Epic] Online Play	80	-	-
Cards-70	☐ [Card Managment, Robustness] Anima...	5	12%	4,12
Cards-105	☐ [Card Managment] Play Deck - Multi...	-	-	-
Cards-80	☐ [Epic] Hand Probability	20	61%	5,15
Cards-71	☐ [Card Managment, Robustness] Shuffl...	2	12%	10,30
Cards-77	☐ [Robustness] Ornate Dealer Token	2	-	-
Cards-79	☐ [Robustness] Shuffle Noise	1	15%	25,76
Cards-76	☐ [Epic] Stud Poker Variants	60	-	-
	☐ Release 1.0 - Done and placed on b...	Total: 0		

Figure 6.20: ScrumWorks Pro Java-Webstart user interface

backlog items. This matches the release backlog requirement from the book review.

The sprint time horizon in ScrumWorks Pro is a good match to the iteration time horizon in the book review requirements and the case company requirements. At least one team must be assigned to a sprint when it is created. Sprints are always done in the context of some project, which matches the case company requirement but conflicts with the book review requirement. The sprint has a start and end date which corresponds to the length in the book review requirements and to the deadline in the case company requirements. Sprint can have a free-text goal field, which is a rough match to the iteration goal book review requirement. The actual work accomplished property in the book review requirements can be calculated from the estimates of the backlog items that have been done in a sprint in ScrumWorks Pro. Sprints have a sprint backlog as a container of work and backlog items can be moved between release backlogs and sprint backlogs. This corresponds to the iteration backlog requirement of the book review and the requirements in the project requirement of the case company.

6.4.3 Containers of work in ScrumWorks Pro

ScrumWorks Pro has one kind of container of work which contains backlog items (see 6.4.4). Sprints and releases both have this container of work which is called the sprint backlog or the release backlog depending on which time horizon it belongs to. A release backlog that belongs to a release that does not have a start and end date can be likened to the project backlog container of work in the book review requirements.

Backlog items are moved between the containers of work in ScrumWorks Pro and can only belong to one container of work at any time. Tasks (see 6.4.4) do not have a separate container of work in ScrumWorks Pro.

6.4.4 Pieces of work in ScrumWorks Pro

ScrumWorks Pro has two different pieces of work, which are the backlog item and the task. These match the backlog item and task pieces of work in the book review requirements. ScrumWork Pro's backlog item is good match to the case company requirement's requirement piece of work. The properties the backlog item has that match the book review backlog item requirements are the title, the description, the done flag, and the estimate. ScrumWork Pro's backlog item can have only one effort estimate. The effort estimate value can be set when the backlog item is created and it can be changed later, but no history of previous estimates is shown and, therefore, the requirement for original estimate property of backlog item from the book review is not filled. Backlog items in ScrumWorks Pro are always prioritized in a relative order, which does not match the case company prioritization requirement of priority class based prioritization. The backlog item in ScrumWorks Pro has a list of tasks attached to it. Files can be uploaded and attached to backlog items, which matches the attached specifications documents requirement of the case company.

The task is the second piece of work in ScrumWorks Pro. It is a good match to the task in the case company and the book review requirements. Tasks always belong to one backlog item but they can be moved between backlog items. No separate container of work for tasks exists in ScrumWorks Pro. This is in conflict with the requirements of the book review and case company.

The task in ScrumWorks Pro has a status which can be one of not started, in progress, impeded, or done. This is a good match to the task status property requirement in the book review, but the list lacks the reviewable status of the case company task requirements.

The book review task property requirements that have a good match in ScrumWorks Pro are the description, the responsible person, the status, the effort spent, and the effort left. The only missing task property is the originator. The effort left and the effort spent in ScrumWorks Pro task can be recorded per day and the total sum is also shown for the spent effort. This matches the book review requirements.

ScrumWork's task lacks all but one of the second priority requirements of the case company. The change history second priority requirement is partially filled by the effort left and effort spent history feature. The first priority task property requirements that are filled by ScrumWorks Pro are the description, the remaining effort, and the one responsible person. Tasks cannot be assigned multiple responsible persons in ScrumWorks Pro.

The case company effort spent entry piece of work does not have a corresponding piece of work in ScrumWorks Pro. Spent effort can be recorded in tasks, which is a very rough match to the effort spent entry requirement. One effort spent value can be recorded per day and sum of these values is also shown. Many effort left entries cannot be recorded for a task during one day and no one other than the task's responsible person can record spent effort.

6.4.5 Selection of work in ScrumWorks Pro

The only selection of work requirement filled by ScrumWorks Pro is the requirement for task list showing the tasks assigned to the selected person. ScrumWorks Pro has a "My

Tasks” display, which shows all backlog items and tasks of the selected project and the tasks of the current user can be highlighted in this view.

6.4.6 Monitoring work in ScrumWorks Pro

ScrumWorks Pro draws a sprint burn-down chart based on task’s effort left estimates. Figure 6.21 shows an example of ScrumWorks Pro’s sprint burn-down chart. The red dots show sums of task effort left estimates in the sprint. This matches the book review and case company requirements for an iteration burn down chart. ScrumWorks Pro has a view that shows all backlog items in all releases and iterations. Backlog item’s done flags are also shown in this view. The view is a good match to the book review requirement of a list of all backlog items “done” compared to the other backlog items. ScrumWorks Pro does not draw release burn-down charts and it does not have a view for task’s effort left sums per user.

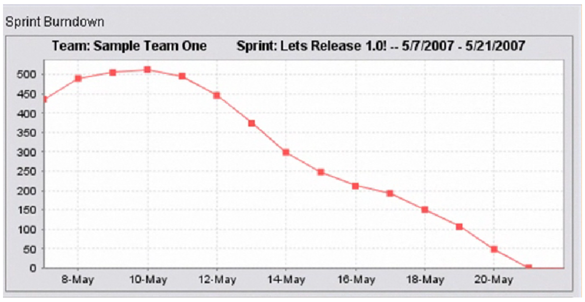


Figure 6.21: ScrumWorks Pro sprint burn-down example

6.4.7 Views in ScrumWorks Pro

The “MyTasks” view in ScrumWorks is only a partial match to the daily work of a developer view requirement, as only list of assigned tasks is shown. Other daily work of a developer view requirements or any other views related requirements are not filled by ScrumWorks Pro. ScrumWorks Pro does have a view that displays tasks and sums of tasks’ effort left values, but this view cannot be filtered to show only tasks of the selected user.

6.4.8 Work-hour reports in ScrumWorks Pro

ScrumWorks’ ability to generate work-hour reports is limited to showing selected sprint’s effort spent history table. This table shows each task’s effort spent entries per day and sums of the effort spent entries values. This is a partial match to the hours in an iteration or project case company requirement. Other hour reporting requirements are not filled by ScrumWorks Pro.

6.5 Summary of the tool requirements and features

This section contains tables which summarize the results of the tool reviews. See the other sections in this chapter for a detailed descriptions of the results.

Table 6.1 displays a comparison of the time horizon requirements from the book review and tool features. Table 6.2 contains a comparison of time horizon requirements from the case company and tool features. Table 6.3 displays a comparison of the pieces of work requirements from the book review and tool features. Table 6.4 displays a comparison of the pieces of work requirements from the case company and tool features. Table 6.5 compares the selection of work requirements from the book review and the case company to the tool features. Table 6.6 compares the monitoring progress requirements from the book review and the case company to tool features. Table 6.7 contains the comparison of the view requirements from the case company and the tool features. Table 6.8 contains the comparison of the work-hour reporting requirements from the case company and tool features.

The first column of each table contains requirements from the case company or book review. The following three columns contain matching features from the tools, if available. A long line (—) denotes that the requirement was not filled by the tool in that column. The requirements from the case company are denoted by a priority. The priority one requirements are denoted by ^{1}, the priority two requirements are denoted by ^{2}, and the requirements related to work-hour reporting are denoted by ^{T}. **Boldface** text denotes the main requirements and features. The normal face requirements and features in the rows following the boldface requirements and features are sub-requirements and sub-features related to the boldfaced main requirements and features.

Table 6.1: Book review time horizon requirements compared to the tools’ features

Book review	Agilefant	Mingle	Rally EE	ScrumWorks Pro
Product time horizon	**Product time horizon**	**Product card type**	**Upper hierarchy-level project in project hierarchy**	**Product time horizon**
Product backlog	Product backlog	Card tree hierarchy	Workspace backlog with product filtering	Release backlog without start and end dates
Project time horizon	**—**	**Project card type**	**Lower hierarchy-level project in a project hierarchy**	**—**
Optional connection to a product	—	Card tree connection to a product card	Belongs to an upper level project in hierarchy	—
Project goal	—	Project goal card type	—	—
Release time horizon	**Project time horizon**	**Release card type**	**Release time horizon**	**Release time horizon**
Date	End date	Date	End date	End date
Release backlog as a subset of a product backlog	Project backlog separate from a product backlog	Card tree hierarchy	Release backlog as a subset of a workspace backlog	Release backlog separate from a product backlog
Release goals with optional connection to a project goal	—	Release goal cards with a card tree connection to a project goal card	Free-text theme field	—
Optional connection to a project	Mandatory connection to a product	Card tree connection to a project card	Mandatory connection to a product	Mandatory connection to a product
Iteration time horizon	**Iteration time horizon**	**Iteration card type**	**Iteration time horizon**	**Sprint time horizon**
Length	Start and end date	Length	Start and end date	Start and end date
Capacity	—	Capacity	Resources	—
Actual work accomplished	Sum of done backlog items’ original estimates	Actual work accomplished	Sum of tasks’ actuals fields	Sum of done backlog items’ original estimates
Task list	—	Card type tree	—	—
Iteration backlog as a subset of a release backlog	Iteration backlog separate from a project backlog	Card tree hierarchy	Iteration backlog as a subset of a workspace backlog	Sprint backlog separate from a release backlog
Iteration goals with optional connection to a release goal	Iteration goals	Iteration goal cards with a card tree connection to a release goal card	Free-text theme field	Free-text goal field
Optional connection to a release	Mandatory connection to a project	Card tree connection to a release card	Mandatory connection to a project	—

Table 6.2: Case company time horizon requirements compared to the tools' features

Case company	**Agilefant**	**Mingle**	**Rally EE**	**ScrumWorks Pro**
[1] **Project time horizon**	**Project time horizon**	**Project card type**	**Project time horizon**	**Project time horizon**
[1] Responsible persons	—	One responsible person	—	—
[1] Deadline	End date	Deadline	End date	—
[1] Sold hours	—	Sold hours	—	—
[1] Planned hours	—	Planned hours	—	—
[1] Type	Type	Type	—	—
[1] Tasks in the project	Project backlog	Card type tree	—	—
[1] Requirements in the project	—	Card type tree	User Stories in the project	Backlog items in a dateless release backlog
[T] Default work type	—	—	—	—
[T] Locked flag	—	—	—	—
[1] **Iteration time horizon**	**Iteration time horizon**	**Iteration card type**	**Iteration time horizon**	**Sprint time horizon**
[1] Deadline	End date	Deadline	End date	End date
[1] Mandatory connection to a project	Mandatory connection to a project	Card tree connection to a project card	Mandatory connection to a project	Mandatory connection to a project
[1] Tasks in the iteration	Iteration backlog	Card type tree	—	—
[2] Requirements in the iteration	Iteration's goals	Card type tree	User stories in the iteration	Backlog items in the iteration
[T] Default work type	—	—	—	—
[T] Locked flag	—	—	—	—

Table 6.3: Book review pieces of work compared to the tools' features

Book review	**Agilefant**	**Mingle**	**Rally EE**	**ScrumWorks Pro**
Backlog item	**Backlog item**	**Backlog item card type**	**User story**	**Backlog item**
Name	Name	Name	Name	Title
Description	Description	Description	Description	Description
Risk class	—	Risk class	Custom field risk class	—
Value class	—	Value class	Custom field value class	—
Urgency flag	—	Urgency value	Custom field urgency flag	—
Done flag	Done status	Done value	Done status	Done flag
Priority class or relative priority	Priority class	Priority class	Priority class or relative priority	Relative priority
Effort remaining for each iteration	Current effort remaining	Current effort remaining	To Do calculated from task effort left estimates	—
Initial estimate	Original estimate	Initial estimate	Planned estimate	Estimate
Adjusted estimate	—	—	—	—
Optional relationship to many tasks	Each tasks belongs to one backlog item	Card tree relationship to many task cards	Each tasks belongs to one user story	Each task belongs to one backlog item
Task	**Task**	**Task card type**	**Task**	**Task**
Name	Name	Name	Name	Name
Description	Description	Description	Description	Description
One responsible	—	One responsible	One responsible	One responsible
Status class	Status class	Status class	Status class	Status class
One originator	—	One originator	—	—
Effort spent	—	Effort spent	Effort spent	Effort spent per day
Effort left	—	Effort left	Effort left	Effort left
Optional relationship to many backlog items	Always belongs to one backlog item	Card tree relationship to one backlog item card	Always belongs to one user story	Always belongs to one backlog item

Table 6.4: Case company pieces of work compared to tool features

Case company	Agilefant	Mingle	Rally EE	ScrumWorks Pro
^{1} **Requirement**	**Iteration goal**	**Requirement card type**	**User story**	**Backlog item**
^{1} One responsible	—	One responsible	One responsible	—
^{1} Priority class	—	Priority class	Priority class	Relative priority
^{1} Links to specification documents	—	Non-link free text URL	Strictly formatted URL	—
^{2} Attached specification documents	—	Attached specification documents	Attached specification documents	Attached specification documents
^{2} One originator	—	One originator	—	—
^{1} **Task**	**Backlog item**	**Task card type**	**Task**	**Task**
^{1} Description	Description	Description	Description	Description
^{1} One responsible	One responsible	One responsible	One responsible	One responsible
^{1} Many responsible persons with weights	—	—	—	—
^{2} Many responsible persons	Many responsible persons	—	—	—
^{1} Overdue flag	—	Overdue value	Custom field overdue flag	—
^{1} Status class	State class	Status class	Status class	Status class
^{1} Original estimate	Original estimate	Original estimate	Original estimate	—
^{1} Remaining effort	Effort left	Remaining effort	To-Do estimate	Remaining effort
^{1} Percentage done based on the original estimate and remaining effort	Percentage done based on the status of the tasks belonging to the backlog item	Percentage done based on the original estimate and remaining effort	—	—
^{1} Billable flag	—	Billable value	Custom field billable flag	—
^{2} Deadline	—	Deadline	Custom field deadline date	—
^{2} Relative Priority	Relative priority	—	—	—
^{2} Sold effort estimate	—	Sold effort estimate	Custom field sold effort estimate	—
^{2} One creator	—	One creator	—	—
^{2} One reviewer	—	One reviewer	—	—
^{2} Reviewed flag	—	Reviewed value	Custom field reviewed flag	—
^{2} Change history	—	Change history	Change history	Effort history
^{1} Related requirements	One related iteration goal	Card tree relationship to many requirement cards	One related user story	One related backlog item
^{2} Related projects	—	Card tree relationship to many projects	—	—
^{2} Related issue in external system	—	—	Strictly formatted URL	—
^{2} Splitting tasks to several tasks	—	—	—	—
^{2} Importing bugs as tasks from Bugzilla	—	—	Importing bugs as issues from Bugzilla	Importing bugs as backlog items from Bugzilla

Table 6.4: Continued case company pieces of work compared to tool features

Case company	Agilefant	Mingle	Rally EE	ScrumWorks Pro
[1] **Effort spent entry**	**Effort log entry**	**Effort spent entry card type**	**Task's actual effort field**	**Tasks's hours spent field**
[1] Amount	Amount in hours	Amount	Amount	Amount
[1] Date	Date	Date	—	Date
[1] Comment	Comment	Comment	—	—
[1] Work type	—	Work type	—	—
[2] One recorder	—	One recorder	—	—
[1] Work performers	Work performers	One work performer	—	—
[1] Target task	Target backlog item	Card tree relationship to one task	Target task	Target task

Table 6.5: Work selection requirements and the tool features

Work selection	Agilefant	Mingle	Rally EE	ScrumWorks Pro
Book review				
Task list showing the tasks assigned to the selected person	—	Card filter showing the task cards assigned to the selected person	Task list showing the tasks assigned to the current user	Task list highlighting the tasks assigned to the current user
Stack based task list	—	—	List of tasks ordered by ID or name	—
Case company				
[2] E-mail notifications for newly assigned tasks	—	—	E-mail notification rule for newly assigned tasks	—

Table 6.6: Progress monitoring requirements and tool features

Monitoring progress	**Agilefant**	**Mingle**	**Rally EE**	**ScrumWorks Pro**
Book review				
Release burn-down chart	—	Programmed release burn-down chart	Release burn-down chart based on user story states and original estimates	—
Iteration burn-down chart based on the effort left estimates of tasks	—	Programmed iteration burn-down chart based on the effort left estimates of tasks cards	Iteration burn-down chart based on the effort left estimates of tasks	Sprint burn-down based on tasks' effort estimates
Display of number of backlog items "done" compared to the items in iteration backlog and release backlog	Status color-coded list of backlog items in an iteration or release	Card filter coloring "done" cards belonging to different iteration and release card trees	Filterable list of all user stories, containers and states	List with done flags of all backlog items in all containers
Display of task effort left sum of an individual developer per iteration	—	—	Filterable list of a developer's assigned tasks and the tasks' effort left estimates	—
Case company				
[1] Iteration burn-down chart based on tasks' effort left estimates	Iteration burn-down chart based on backlog items' effort left estimates	Programmed iteration burn-down chart based on tasks' effort left estimates	Iteration burn-down chart based on tasks' effort left estimates	Sprint burn-down chart based on tasks' effort left estimates

Table 6.7: Case company views and tool features

Required view	**Agilefant**	**Mingle**	**Rally EE**	**ScrumWorks Pro**
[1] **Daily work of developer view**	**Daily work of developer view**	**Saved card filter pages**	**My Tasks widget**	**My Tasks view**
[1] List of assigned tasks	List of assigned backlog items	List of assigned task cards	List of assigned tasks	List of assigned tasks
[1] List of tasks to review	—	List of task cards to review	—	—
[1] List of current and previous weeks' effort spent entries	—	List of effort spent cards of the selected time period	—	—
[1] Effort spent calendar	—	List of effort spent cards with the selected date	—	—
[2] Effort left per iteration	Effort left per day	Sum of effort left values of task cards in the selected iteration	Filterable list of tasks and the tasks' effort left sums	—
[1] **Work division view**	**—**	**—**	**—**	**—**
[1] **Status meeting view**	**—**	**List of effort spent cards for the selected person and time period**	**—**	**—**
[1] **Ongoing projects' status view**	**—**	**Table of projects, spent hours, sold hours and estimated hours**	**—**	**—**

Table 6.8: Case company work-hour reporting requirements and the tool features

Required report	**Agilefant**	**Mingle**	**Rally EE**	**ScrumWorks Pro**
^{T}**Hours in an iteration or project**	**Hours in an iteration or project**	**Hours in an iteration or project**	**Hours in an iteration or project**	**Hours in a sprint**
^{T} Hour totals in iterations if a project	Hour totals in iterations if a project	Hour totals in iterations of the selected project card	Sum of selected iteration's tasks' effort spent	—
^{T} Total hours	Total hours	Total hours	Total hours	Total hours
^{T} Non billable hours	—	—	—	—
^{T} Billable hours	—	—	—	—
^{T} **Hours of a person in time period**	**Hours of a person in time period**	**—**	**Hours of a person in time period**	**—**
^{T} Selected person	Selected person	—	Selected person	—
^{T} Selected time period	Selected time period	—	—	—
^{T} Hours per iteration or project	Hours per iteration or project	—	Sum of tasks' effort spent values in the selected iteration	—
^{T} **Hours of a person per work type**	**—**	**—**	**—**	**—**

Chapter 7

Discussion

This chapter contains discussion on the results of the thesis. The features the tools have and their compliance with the requirements is discussed in Section 7.1. The different conceptual models of the tools and their weaknesses are discussed in Section 7.2. The validity of the results is discussed in Section 7.3.

7.1 Discussion on the tool features and the requirements

This section contains discussion on the tools' features and how well the features fulfill the requirements from the book review and case company. Each individual tool is first discussed separately: Section 7.1.1 contains discussion on Agilefant, Section 7.1.2 on Mingle, Section 7.1.3 on Rally Enterprise Edition and Section 7.1.4 on ScrumWorks Pro. Section 7.1.5 then discusses the requirements and the features of the tools in general.

7.1.1 Agilefant and the requirements

Agilefant fails to fill a crucial part of the requirements from the book review. It differs notably from the requirements on a conceptual level regarding the backlog item and task pieces of work. In the book review requirements, the backlog items are long-term planning pieces used to plan complete iterations and tasks are used for short-term planning of iterations. Agilefant's backlog item fills both roles. Tasks in Agilefant serve only the role of a to-do list of a backlog item. These backlog items and tasks both lack many of the properties identified in the book review. The most important missing properties are the effort-left estimate and the responsible of a task. The lack of these properties precludes forming a task list for a person or an iteration burn-down chart, which both are very important tools for managing the agile software development work.

The project time horizon in Agilefant corresponds to the release time horizon in the book review requirements and the project time horizon presented in the book review requirements does not exist in Agilefant. Unlike the time horizons in the book review requirements, the time horizons in Agilefant must form a hierarchy; a lower-level time horizon cannot exist without a higher-level time horizon. Agilefant does not offer a hierarchy of goals from project to release to iteration, as it only supports iteration goals. The purpose of the hierarchy of goals in the combined requirements of the book review is to give high abstraction-level guidelines for the more concrete planning, which uses backlog items and tasks. The lack of higher-level goals diminishes the usefulness of the iteration goals, as higher-level goals cannot be used as guidelines for them.

The different role of tasks in Agilefant compared to the book review requirements is the most severe conceptual deficiency. Instead of being the main way to plan and manage work in iterations, as in Scrum and Extreme Programming, the tasks in Agilefant are to-do lists of individual backlog items. The difference prevents using Agilefant for planning iterations and managing work during iterations. This deficiency alone is severe enough so that Agilefant cannot be considered to have sufficient features to be used in agile software development management as presented in the book review requirements. The lack of a hierarchy of goals and the missing project time horizon, which alone would not have been sufficient to consider Agilefant unfit, strengthen this conclusion.

Mapping both the task and the backlog item of the book review requirements to the backlog item in Agilefant could have been an alternative way to do the conceptual mapping. Many-to-one mappings without explicit support in a tool were avoided in the conceptual mappings of all tools, as such mappings may result in serious problems when a tool is actually used. The same information may need to be saved in several places, which can cause synchronization problems. The two concepts mapped into the same concept must be differentiated by an explicit naming scheme or in some other way. Without the support from a tool, small errors quickly cause problems in referential integrity. Searching and sorting functionalities in tools become less useful, as a tool may not be able to easily differentiate between the two concept types mapped into the one concept type in the tool. The impact of these issues on the usefulness of a tool are very hard to measure without extensive real world usage and such trade-offs should be done by an adopter of a tool instead of the author of this thesis.

Agilefant fails to fill some of the priority one requirements from the case company. The most crucial deficiency is the lack of properties matching the requirement piece of work from the case company requirements. While Agilefant's iteration goal can be matched with the requirement piece of work, iteration goals lack such high-priority requirement properties as the responsible person and priority.

Agilefant's release time horizon can be matched with the project time horizon in the case company requirements. However, the release time horizon lacks many priority one properties the case company project time horizon has. In addition to the conceptual deficiencies, Agilefant also lacks most of the priority one views required in the case company. Because of the lack of matching properties for the requirement and the lacking views (see Section 6.1.7), the features of Agilefant cannot be considered sufficient for managing the daily work of the software development in the case company.

7.1.2 Mingle and the requirements

Mingle's flexible conceptual model of configurable card types and card type trees allows creation of card types and trees which match the conceptual models of the book review and case company. This is very different from Agilefant, in which the only available customization option is turning work-hour reporting on or off. However, Mingle has several notable weaknesses. Properties of card types cannot contain several values regardless of the property type, which prevents assigning multiple people to the case company's task or project or creating effort spent entry cards with multiple work performers. Using several card type trees enables modeling any relationship between concepts in the book review and case company requirements.

Cards and card trees must also be used when entering work hours and generating work-hour reports, and this is the first major weakness in Mingle. The aggregate card property model in Mingle allows only creation of very simple work-hour reports which are limited to sums of all work-hour reports under selected task, requirement or iteration. More complex reporting for a certain person, work type, or time period is not possible.

The second major weakness in Mingle is the programming language that is used to generate dynamic content in wiki pages. While the need to learn new programming language to generate burn-down charts and other content is an issue in itself, the programming language can only use one card type as data source for each piece of generated content. In addition, card type trees cannot be used as data sources. This prevents generation of content which requires combining data from several card types or requires data from card type tree relationships.

Mingle does fulfill the requirements from the book review quite well. Nearly all concepts and relationships can be modeled in it, and simple burn-down graphs can be created using the dynamic content generation language. Based on research done in this thesis, Mingle does have sufficient features for managing the daily work of agile software development organization according to the requirements from the book review.

While the case company requirements concerning generation of work-hour reports are considered less reliable than the other requirements, Mingle's failure to fill most of these requirements clearly indicates that the tool is not suitable for an environment where recording workhours and creating advanced work-hour reports is important. This also means that Mingle does not sufficiently fulfill the requirements from the case company.

7.1.3 Rally Enterprise Edition and the requirements

Rally software claims to be designed for generic agile software development. Comparing the book review requirements and Rally EE's features indicates that this indeed is the case. Many features of Rally EE are very close matches to book review requirements. The most significant difference is the lack of a separate container for tasks. Every task in Rally EE must belong to a user story, but tasks in the book review requirements have their own container and may be related to many backlog items or none at all.

Rally EE's custom properties feature allows some customization, although not nearly as much as Mingle's card model. The problem in Rally EE, considering the book review requirements, is that custom properties given to user stories do not show in any of the list views. Custom properties value class and risk class are mostly used during iteration planning. Viewing them from the individual user story view is not a huge problem, but the urgency flag custom property is an important indicator and should be visible in every list view.

Rally EE also contains some minor conceptual differences to the book review requirements. Tasks and user stories have only one effort left value instead of one for each day for tasks and one for each iteration for backlog items, the user story effort left value is calculated from stories effort left values and cannot be edited, and iterations' actual work accomplished values are automatically calculated from tasks' actuals fields and cannot be edited. In addition, Rally EE lacks hierarchy of goals for time horizons.

While Rally EE shares some of Agilefant's deficiencies, the major difference is that in Rally EE, tasks do have the properties that are required for using them for planning and managing iteration-level development work. The mandatory connection of task and user stories in Rally EE is a major weakness, but it can be circumvented, for example, by creating one "general" user story which contains all tasks that are not related to other user stories. However, this does not remove the problem that a task can only be related to one user story. Regardless, Rally EE does support most of the requirements from the book review and, based on this study, it has sufficient features for managing the daily work of agile software development according to the book review.

The lack of a separate container for tasks in Rally EE is a much bigger problem when compared to the case company requirement than when compared to the book review requirement. Tasks in the case company requirements are much more independent from requirements than tasks in the book review requirements are from backlog items. Tasks in Rally EE lack many second priority properties from the case company requirements. However, this is not considered a serious deficiency. Tasks also lack two first-priority requirements, which are many responsible persons with weights and reviewable status. User stories in Rally EE are quite good match to requirements in the case company.

Deadline is the only case company requirement that has a match in Rally EE time horizons' properties. Rally EE's project and iteration time horizons lack all other required properties. This is serious deficiency, as planned hour and sold hours properties of case company's projects are an important aspect in the project management. However, the most severe deficiency in Rally EE is the complete lack of effort spent entry concept of the case company. Effort spent can be only recorded in the single actuals field in tasks and no additional information, such as work type or date, can be given. The spent effort is only reported as sums of tasks' actuals field in user stories and iterations. This fails most of the requirements concerning work-hour reporting and many requirements concerning views. The lack of proper work-hour entering and reporting functionality is the main reason Rally EE does not sufficiently fulfill the requirements the case company has for a agile software development management tool.

7.1.4 ScrumWorks Pro and the requirements

ScrumWorks Pro is developed by Danube Technologies, a company that also provides Scrum training. ScrumWork Pro's very bare-bones feature set could be a result of developing only features that match the Scrum method as taught by the company. Comparing ScrumWorks Pro's features to the Scrum requirements, which were presented in Chapter 4, reveals that ScrumWorks Pro does not even completely fill the requirements Scrum has for a tool according to the book review. This indicates that if ScrumWorks is based on Danube's Scrum method, the method differs from the Scrum method described in the books used as source in the book review (see Section 3.2).

The largest difference between ScrumWorks Pro and the book review requirements is the way tasks are used. As in Agilefant and Rally EE, tasks do not have their own container and they always belong to a higher-level piece of work. Tasks also lack many of the properties of the book review requirements and most of these lacking properties come from the XP side of the book review results, which indicates that ScrumWorks Pro is indeed meant to be used only with the Scrum process.

ScrumWorks Pro also has many other deficiencies. The tool lacks the hierarchy of goals as only sprints have a field for themes. There is no way to create a time horizon that would correspond to the project time horizon in the book review requirements. ScrumWorks Pro also lacks about half of the views of the book review requirements.

While ScrumWorks Pro does have deficiencies in features compared to the book review requirements, the most critical deficiency concerning tasks can be worked around in a similar manner as in Rally EE. A "general" backlog item, which contains all tasks that are not related to any real backlog item, can be created. However, the lack of customization, views, and properties are major problems. Based on this research, ScrumWorks Pro does not have sufficient features for managing the daily work of agile software development according to the book review.

Compared to the case company requirements, ScrumWorks Pro has very few matching features. The project and iteration time horizons exist, but lack most of the properties as do the requirement and task pieces of work. Work hours in ScrumWorks Pro are only recorded to tasks and none of the advanced work-hour reports can be created. The only case company view requirement that is implemented in ScrumWorks Pro is the list of assigned tasks.

Based on the very weak match of ScrumWorks Pro's features to the case company requirements, the tool clearly does not have sufficient features for managing the daily work of agile software development according to the case company requirements.

7.1.5 The tools in general and the requirements

All of the commercial tools have severe deficiencies in work-hour recording and reporting functionality compared to the case company requirements. In Mingle, the problem arises from the relatively complex functionality required to create the work-hour reports. Such functionality could be difficult to implement because of the great level of customization the tool offers. Consequently, the problem is somewhat inherent to the tool.

The other two commercial tools do not have such an inherent problem and the lack of advanced work-hour recording and reporting functionality might be a design decision. These design decisions may stem from the agile methods upon which the tools are based. Recording work hours is not a part of the Scrum process and in XP, work hours are only recorded on the task level and for the purpose of improving future effort estimates. Both of these practices ignore work-hour recording and reporting requirements stemming from business needs in real world organizations. The actual requirements vary between organizations, but two examples of work-hour recording and reporting needs from the case company of this thesis are work-hour reporting for workers who are paid by the hour and work-hour reporting that is required by a project client. A separate system for recording and reporting work hours could arguably be used for these purposes, but using many systems which contain parts of the same information creates other problems, such as keeping the information synchronized between the systems and generating reports that require input from several systems.

Every tool lacked some of the properties which were presented in the requirements. The reasons for the missing properties varied among the tools. Rally EE simply lacked some of the properties. Some of the missing properties could be created as custom properties, but the custom properties lacked important functionality. Mingle's biggest property-related deficiency was that multiple values could not be given to one property. Agilefant's conceptual model is quite different from the conceptual models presented in the requirements and the deficiencies in properties stem mostly from that. ScrumWorks Pro has a very bare-bones Scrum-style approach to backlog management and combined with the lack of configuration options, this results in many missing properties.

7.2 Discussion on tool conceptual model categories and weaknesses

The tools included in this research can be divided into two conceptual model categories. The first category is the tools which have mostly fixed conceptual models. Agilefant, Rally and ScrumWorks belong to this type. The tools of this type offer very limited customization options and work best with a process which reflects the conceptual model in the tool. The problem with this tool type is that the usefulness of the tool is significantly reduced if the organization's process or conceptual model is not compatible with the tool. In this situation, the organization has three options. It can change its processes to fit the tool, workarounds for the differences can be created, or the tool can be discarded. Disregarding problems that are related to process change, the first option might cause the organization to work less efficiently. The second option results in extra work, as the workarounds must be institutionalized and enforced. If third option is chosen and the incompatible tool is discarded, the organization must locate a new tool with better conformance or avoid using a tool altogether.

The second category is tools that have no fixed conceptual model. The tool fitting this category in this research is Mingle. While Mingle has some shortcomings in its concept customization options, the biggest problem in this category tools is the difficulty of using complex relationships between concepts and properties. As demonstrated by Mingle, the generation of relatively simple aggregate reports in tools that belong to this category may be difficult or completely impossible. If an organization has needs that require these functionalities, it has three options. If implementing the functionality is not completely impossible, the organization may spend enough resources to customize the tool to fit its needs. Another option might be that the organization uses the tool only on a basic level or the tool may be discarded and another tool selected instead. Each of these options may cause significant amount of extra work for the organization.

7.3 Discussion on threats to validity of results

This section contains discussion on the threats of validity of the results presented in this thesis. This section is divided into subsection that follow the structure of the thesis: Section 7.3.1 discusses threats to the validity of agile method, book and tool selection, Section 7.3.2 discusses threats to the validity of requirements from the book review, Section 7.3.3 discusses threats to the validity of requirements elicited from the case company and Section 7.3.4 discusses threats to the validity of the tool review results.

7.3.1 Agile method, book and tool selection validity

The prime threat to the real world relevancy of this thesis is the possibility that the selected agile methods are not the most prominent agile methods. This could a be result of bias in the popular article sources that were used in the method selection. To the author's knowledge, no peer-reviewed scientific study on distribution of agile software development methods in software industry has been published. A survey on the topic [VersionOne, inc, 2008] was conducted and published by VersionOne, the developer of the VersionOne tool. According to the survey, Scrum was the most followed agile method with 49.1%, Scum/XP hybrid was the second most followed with 22.3%, and XP the third most followed with 8.0%. The survey was not published in a peer reviewed venue and subsequently cannot be considered completely reliable. A systematic research literature review on agile software development was conducted by Dybå and Dingsøyr [2008]. Of the 33 publications reviewed in the paper, 25 publications discuss XP and 3 publications discuss Scrum. The results of these two publications reaffirm the conclusion that Scrum and XP are the most prominent agile software development methods and, consequently, the risk that wrong agile methods were selected for this research is very small.

Another related problem is the relative youth of agile software development methods. The books selected for the book review represent only the state of the respective methods at the times of writing of the books. It is possible that the methods have significantly changed since the writing of the books and, consequently, the requirements elicited from the books might not represent the current state of the methods. There had been some changes between the two included books in both methods, but most of the changes had made the two methods more similar to each other and it is quite likely that this direction is intentional. In practice this means that XP has adopted some Scrum aspects and Scrum has taken on some XP aspects. Since these are the two methods included in this research, further changes in those directions do not notably affect the validity of the results of this thesis. Other major changes in the methods are a threat to the validity of the results, but considering the nature of the previous changes in the methods, such changes are quite unlikely.

The selection of books that were used as the source of requirements is also a risk considering the validity of the results in this thesis. If the books that were selected do not correctly portray the agile methods that were selected, the resulting requirements are irrelevant. The four books that were selected as the sources for book review requirements are all written by authors who are considered to be the original authors of their respective agile methods [Highsmith, 2002]. Considering that the goal was to find out the requirements that the agile methods present when done by-the-book, the risk that the requirements are invalid because the wrong books were selected is not significant.

The probability that some relevant tools were not included in the big list of agile software development tools is quite high. However, the goal of this thesis was not to prove that no tool with sufficient functionality exist, but to determine if a tool with sufficient functionality could be identified with reasonable effort. The tools which were selected should present a good sample of the tools that are publicly available and the inclusion and exclusion criteria for tools are clearly presented. Hence, the selection of tools that were reviewed is not a significant risk to the validity of this research.

7.3.2 Book review requirements validity

Incorrect requirements or the omission of requirements in the book review are significant risks to the validity of the results in this research. A book review protocol was created and followed when the book review was performed. However, the protocol has several weaknesses. The most significant weakness is that only one person performed the review. Using multiple reviewers is a well-established and recommended method for data analysis because it reduces the effects of protocol breaches and omissions. Another weakness in the book review protocol is the lack of coding of the findings of the review. The lack of coding restricted the analysis of the results to only qualitative analysis and no quantitative analysis of requirement validity or importance was possible. Based on these weaknesses in the book review protocol, it is quite likely that there are some errors in the book review requirements.

The related work introduced in Section 1.1 and described in detail in Appendix A provides a point of comparison for the validity of the book review requirements. The existing research concentrates on Extreme Programming practices and therefore can only be compared to the Extreme Programming part of requirements described in Chapter 4. In addition, most of the referenced papers give only a brief description of the methods used to elicit the requirements and the purpose of some of the papers is to enhance the XP way of requirements management with methods and information that comes from outside the agile software development methodology. Because of these two issues, the requirements from related literature cannot reliably be used to find requirements that are missing from the set of XP requirements presented in this research. However, the commonalities between the requirements in the related literature and the Extreme Programming results can be used to fortify the correctness of the requirements. The commonalities between the requirements presented in the related work and the XP requirements presented in this thesis are described in the next paragraphs. Some discussion on discrepancies is also included for the sake of completeness.

The scenario framework presented by Breitman and do Prado Leite [2002] mostly contains properties that are not elicited from the XP method. The framework includes two properties taken directly from XP. These are priority and risk, which both are also included in the Extreme Programming requirements in this thesis.

Rees [2002] lists information that a user story card contains in XP according to the paper's author. The properties shared by the XP requirements in this thesis and the list in the paper are the title, estimated implementation time, risk level and description. The list also includes information that, according to this thesis' results, belongs to tasks instead of user stories.

The two papers describing XPSWiki [Pinna et al., 2003a,b] contain requirements covering the whole XP process. Both the release and the iteration XP time horizons described in this thesis are described in the paper. Iterations in the paper have project velocity, which matches Extreme Programming's actual work accomplished in an iteration property that is described in this thesis. In addition to the two time horizons, the papers describe XP project time horizon. The reason for excluding the project time horizon from the XP requirements in this thesis are described in Section 4.2.2. Commonalities in user story properties between the papers and this thesis' results are the description, estimate and priority. XP tasks are also described in the paper, but the only common property is the task's responsible person.

Kääriäinen et al. [2003] describe the XP process and the resulting requirements only on a quite high abstraction level and do not systematically describe what properties the individual time horizons or pieces of work have. However, some commonalities between the paper and the results for XP in this thesis can be found. The release and iteration time horizons are included as are the user story and task pieces of work. In the paper user stories have a priority and an estimate and tasks have an estimate. These three properties also exist in the XP requirements in this thesis.

The related work described above includes most of the XP concepts described in Chapter 4. While some of the concepts were not mentioned in any related work, it is important to keep in mind that the related work concentrates on describing requirements for a new tool being developed. This is an important distinction to the requirements analysis that was conducted for this thesis. Most of the related work papers describe only those XP aspects that are ultimately included in their tool and the authors may have knowingly omitted describing aspects of XP they considered unnecessary in their tools.

The requirements as a whole can be considered valid for the purposes of this research. The final book review requirements were constructed from four different sources with a notable overlap of requirements. This mitigates the risk of requirements omission in one source. While every book review requirement is considered to be equally important, regardless of the number of sources that mention it, the results of a tool's conformity to the requirements will give a learned reader a good idea of the usefulness of the tool for his or her purposes.

7.3.3 Case company requirements validity

The possible problems in the validity of the case company requirements are quite small. Interviews were conducted by two researchers who independently produced the notes that were used in the requirements analysis. This reduces the probability of omissions and errors in the requirements resulting from errors and omissions in the interview notes. In addition, each interview tape was listened in its entirety during the requirement analysis. The biggest risk is omission in the requirements resulting from the relatively small number of interviewees. As only three persons were interviewed, it is possible that some of the case company's requirements did not come up in the interviews. However, the requirements that were found can be considered to be highly valid, as they were validated in separate meetings which included a notable number of case company's employees.

The case company requirements considering work-hour reports are more risky than the other requirements, as they were created according to discussions in a single meeting with two case company employees. In addition, they were not validated in a larger meeting. The possible problems in the validity of these requirements are taken into account when the results considering case company requirements are discussed and these requirements are not used as the sole basis for any conclusions in this thesis.

A big threat to the validity of the case company results generalization is the case company's possible lack of representativeness. The interview results from the case company suggest that the software development process used in the case company is not actually very agile. As the purpose of the case company study was not to evaluate the agility of the software development process in the company, no concrete conclusions can be made on the level of agility in or the representativeness of the case company. Consequently, the requirements from the case company cannot reliably be generalized to other agile software development organizations.

7.3.4 Tool review validity

The tools reviewed in this research differed notably in their conceptual models, configuration capabilities, and user interfaces. For this reason, it was not feasible to create an exact protocol for the tool review. As a result, the tool reviews were executed in a relatively informal manner, which is the biggest threat to the validity of the tool review results. The matching of the conceptual requirements and the concepts in the tools was attempted in a such way that maximized the conformance of the tool's features to the requirements. However, the existence of a better way to match the concepts is not unfeasible.

It is very unlikely that a better way to match the concepts would invalidate the conclusions of this research regarding the tools, as the most serious deficiencies concerned work-hour recording and reporting functionality. It is possible that the tools that were found to have deficiencies in this area had the related functionality hidden in a such way that it was not found in the review. However, this is quite unlikely since information in the tools' websites gave no indication of additional work-hour reporting functionality. One exception to this was Agilefant, which had a significant portion of the required work-hour reporting functionality, but was considered insufficient because of conceptual deficiencies.

Another weakness in the tool review was the treatment of all book review requirements as equally important. While making such prioritization decisions was not considered to be viable because only the books were used as sources, it is possible that there are significant differences in the real importance of the different requirements. However, some decisions on the importance of the different requirements had to be made when the final conclusions were drawn. A different interpretor might well come into different conclusions based on the same data.

Chapter 8

Conclusions

This chapter concludes the thesis. Section 8.1 provides conclusions on the results. Section 8.2 summarizes the contributions of this thesis. Section 8.3 provides directions for further research.

8.1 Conclusions on the results

The high-level research goal of this research is as stated: “Does any tool available for free evaluation have sufficient functionality for managing the daily work of an agile software development organization?” Based on the results of this thesis, tools for managing agile software development by-the-book do exist, but real world organizations have additional requirements that are not filled by the tools included in this research.

The requirements for an agile software development tool were gathered from practitioner guidebooks of Extreme Programming and Scrum, and a case company. Two of the four analyzed tools, Mingle and Rally Enterprise Edition, did sufficiently fulfill the requirements from the practitioner guidebooks. However, they did not sufficiently fulfill the requirements from the case company. The two other tools, Agilefant and ScrumWorks Pro, did not fulfill the requirements from either source sufficiently well. These results show that tools for managing agile software development done by-the-book do exist, but the real world agile software development organization had additional requirements which were not filled by the tools included in the review.

The tool review results in this research reveal that there is variance in the conceptual models of the available tools. Mingle has an extremely flexible conceptual model, but effort is required to configure the tool before it can be used. Agilefant and ScrumWorks Pro both have a rigid conceptual model based on the software development processes the tools are aimed to be used with. This mostly limits their usefulness to those development processes. Rally Enterprise Edition has a somewhat configurable conceptual model which reflects Extreme Programming and Scrum development processes. The differences in the conceptual models reveal that care must be taken when a tool is chosen to find the tool that is best fit to the process which is used in the software development.

The research conducted in this thesis shows that the conceptual models behind Extreme Programming and Scrum agile software development methods have much in common. A concise set of requirements for a tool can be created by combining the requirements from the two methods and the set of requirements can be used in the evaluation of tools. Similarly, a set of requirements can be elicited from an agile software development organization and this set can be used in the evaluation of tools for that organization.

8.2 Summary of contributions

The results of this thesis show that tools for managing agile software development done by-the-book do currently exist, but real world development organizations may have additional requirements that may not be filled by the tools that are publicly available at the present time. This information can be used by organizations searching for a tool for managing software development and by organizations developing agile software development tools to create tools that better match the actual needs of real world agile software development organizations.

The approach taken in this research is novel in two ways: previous work on the subject has concentrated solely on Extreme Programming practices and on building new tools. This research compares the requirements of Extreme Programming, Scrum, and a case company to the features of several publicly available agile software development management tools instead of proposing the creation of a new tool. In this thesis, the requirements for the two agile software development methods were extracted in a rigorous and systematic fashion. Most of the previous work is very light on details of how the requirements were created and what sources were used.

Organizations that are searching for a tool for agile software development management can use the results of the tool review as a baseline for their search. Even if none of the tools analyzed in this thesis fit the needs of the organization, the results can provide guidance on what types of deficiencies and problems the other tools might contain. The lists of all found open-source and commercial tools also provides a starting point for the search.

The requirements presented in this research can be used in the development of new or existing tools. The requirements can be extracted and implemented in a straightforward fashion. The descriptions of the deficiencies and weaknesses in the tools reviewed in this thesis help to avoid recreating the same problems when developing other tools.

While the goals of this thesis did not include the creation of a framework for analyzing agile methods, software development organizations, and tools, a such framework was created as a by-product of the research. While this thesis does not include explicit instructions on how to use the framework to evaluate tools, the description of the research process should provide enough information for a successful use of the framework by other researchers.

8.3 Further research

This thesis presents the requirements of one real world agile software development organization. The needs of an agile software development organization depend on many factors, such as the software development process, the size of the organization, and the type of the software being developed. The needs of a small development team developing a computer game are very different from the needs of a large organization developing safety-critical software. Further research in the requirements of different real world software development organizations is required to form generalizations on what kind of requirements different kinds of organizations typically have. Creation of a such generalized framework would help software development organizations in the selection of tools by removing the need to perform rigorous analysis of the organization's requirements. Such a framework would also help tool developers to include functionality that better fits the selected organization type.

The commercial tools that were chosen for review in this research were selected according to their different conceptual models. Several other commercial tools exist. Review of the other tools would allow broader generalizations on the main deficiencies and problems in the current publicly available agile software development tools. It is also possible that one of the other commercial tools would have sufficiently fulfilled the requirements from both sources.

As described in this thesis, Extreme Programming and Scrum are currently the most prominent agile methods by a clear margin. However, the current agile software development methodology is far from stable and mature. New methods, such as Unified Process-based AgileUP and OpenUP, have been proposed while older agile methods, such as Feature Driven Development and Dynamic Systems Development Methodology, are still in use. In the event that a method other than XP or Scrum gains popularity, further research will be in order to determine the requirements of the method and the availability of tools for the method.

Bibliography

M. Angioni, D. Carboni, S. Pinna, R. Sanna, N. Serra, and A. Soro. Integrating XP project management in development environments. *Journal of Systems Architecture*, 52(11): 619–626, 2006.

Kent Beck. *Extreme Programming explained: embrace change*. Addison-Wesley, Reading MA, 2000.

Kent Beck. Embracing change with Extreme Programming. *Computer*, 32(10):70, 1999.

Kent Beck and Cynthia Andres. *Extreme Programming explained: embrace change*. Addison-Wesley, Boston, MA, 2004.

Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cocburn, Ward Cunningham, Martin Fowler, James Grenning, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Maric, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, Oct 2008. URL `http://agilemanifesto.org/`.

Karin K. Breitman and Julio Cesar Sampaio do Prado Leite. Managing user stories. In *Proceedings of the International Workshop on Time-Constrained Requirements Engineering*, 2002.

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

Fedorenko. Devplanner homepage, Oct 2008. URL `http://www.devplanner.com/`.

Helsinki University of Technology. Agilefant homepage, Oct 2008. URL `http://www.agilefant.org/wiki/display/AEF/Agilefant+Home`.

James A. Highsmith. *Agile software development ecosystems*. Addison-Wesley, Boston, MA, 2002.

J. Kääriäinen, J. Koskela, J. Takalo, P. Abrahamsson, and K. Kolehmainen. Supporting requirements engineering in Extreme Programming: Managing user stories. In *Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications*, 2003.

J. Kääriäinen, J. Koskela, P. Abrahamsson, and J. Takalo. Improving requirements management in Extreme Programming with tool support - an improvement attempt that failed. In *Proceedings of the 30th Euromicro Conference*, pages 342–351, 2004.

Craig Larman. *Agile and iterative development: a manager's guide*. Addison-Wesley, Boston, MA, 2004.

Marco Melis, Walter Ambu, Sandro Pinna, and Katiuscia Mannaro. Requirements of an ISO compliant XP tool. In *Proceedings of the Extreme Programming and Agile Processes in Software Engineering, 5th International Conference*, pages 266–269, 2004.

Merriam-Webster. Merriam-webster online dictionary, Oct 2008. URL `http://www.merriam-webster.com/dictionary/backlog`.

Mozilla Organization. Status counts for firefox, Jul 2008a. URL `https://bugzilla.mozilla.org/`.

Mozilla Organization. Bugzilla, Jul 2008b. URL `http://www.bugzilla.org/`.

Stephen R. Palmer and John M. Felsing. *A practical guide to feature-driven development*. Prentice Hall, Upper Saddle River, NJ, 2002.

Sandro Pinna, Paolo Lorrai, Michele Marchesi, and Nicola Serra. Developing a tool supporting xp process. In *Proceedings of the 2003 Extreme Programming and Agile Methods Conference*, pages 151–160, 2003a.

Sandro Pinna, Simone Mauri, Paolo Lorrai, Michele Marchesi, and Nicola Serra. XPSwiki: An agile tool supporting the planning game. In *Proceedings of the 2003 Extreme Programming and Agile Processes in Software Engineering Conference*, pages 1014–1014, 2003b.

Mary Poppendieck. Lean software development. In *Proceedings of the 29th International Conference on Software Engineering*, page 165, Long Beach, CA, 2007. IEEE Computer Society.

Mary Poppendieck and Tom Poppendieck. *Lean software development: an agile toolkit*. Addison Wesley, Boston, MA, 2003.

Rally Software Development. Rally software development homepage, Oct 2008. URL `http://www.rallydev.com/`.

M. J. Rees. A feasible user story tool for agile software development? In *Proceedings of the Ninth Asia-Pacific Software Engineering Conference*, page 22, 2002.

L. Rising and NS Janoff. The scrum software development process for small teams. *IEEE software*, 17(4):26, 2000.

Outi Salo and Pekka Abrahamsson. Agile methods in European embedded software development organisations: A survey on the actual use and usefulness of Extreme Programming and Scrum. *IET Software*, 2(1):58–64, 2008.

Stephen R. Schach. *Object-oriented and classical software engineering*. McGraw-Hill, Boston, 2001.

Ken Schwaber. *The enterprise and Scrum*. Microsoft Press, Redmond, WA, 2007.

Ken Schwaber and Mike Beedle. *Agile software development with Scrum*. Prentice-Hall, Upper Saddle River, NJ, 2002.

Nicolas Serrano. Bugzilla, ITracker, and other bug trackers. *IEEE software*, 22(2):11, 2005.

Danube Technologies. ScrumWorks Pro homepage, Oct 2008. URL `http://danube.com/scrumworks/pro`.

ThoughtWorks. Mingle homepage, Oct 2008. URL `http://studios.thoughtworks.com/mingle-project-intelligence`.

VersionOne, inc. 3rd Annual "State of Agile Development" Survey. Technical report, 2008.

Appendix A

Related literature

This appendix presents literature that is related to tools for agile software development. Some research concerning tools for management of requirements in agile context is presented, but all included literature discusses tools that are either geared toward Extreme Programming or more specifically managing user stories in a XP context.

Some of the existing literature on tools for agile software development discuss tool support for aspects that are not directly related to work management. Two examples of this are pair programming[1] and continuous integration[2]. As these aspects are not intimately related to managing work, this literature is not examined here.

Breitman and do Prado Leite[3] propose that user stories should be recorded with a scenario notation and describe a scenario framework which divides the information of scenarios into a core and extra mechanism classes, each containing one or several attributes. Additional information is also derived from the relationships between classes. Some of the attributes in the core scenario class are derived from Beck[4], while the the rest of the proposed attributes and extra mechanism classes are based on the authors' experiences on requirements management. The paper also describes an XML structure for storing the scenario framework information.

Rees[5] presents requirements and implementation of DotStories tool for managing user stories. The paper professes that the data model requirements for the DotStories tool are adapted quite directly from Beck[4] with some additions the author has deemed necessary. In addition to the data model, the user interface of the tool is described in detail. The DotStories tool concentrates on providing an electronic alternative for index cards more traditionally used for recording user stories and covers only the iteration level requirements management part of agile software development.

---

[1] Satoshi Atsuta. Extreme Programming support tool in distributed environment. In Proceedings of the 28th International Computer Software and Applications Conference, volume 2, page 32, New York, NY, 2004. IEEE.

[2] Martin Lippert, Stefan Roock, Robert Tunkel, and Henning Wolf. Stabilizing the XP process using specialized tools. In Proceedings of the 8th International Conference on Agile Processes in Software Engineering and eXtreme Programming, 2001.

[3] Karin K. Breitman and Julio Cesar Sampaio do Prado Leite. Managing user stories. In Proceedings of the International Workshop on Time-Constrained Requirements Engineering, 2002.

[4] Kent Beck. Extreme Programming explained: embrace change. Addison-Wesley, Reading MA, 2000.

[5] M. J. Rees. A feasible user story tool for agile software development? In Proceedings of the Ninth Asia-Pacific Software Engineering Conference, page 22, 2002.

Two papers written by Pinna et al.[6] [7] describe XPSwiki tool which according to the papers supports XP requirements gathering and planning game. XPSwiki is based on wiki technology; it is an editable web site with additional support for obligatory structure for individual pages. High level non-functional requirements for the tool are described in addition to functional and data structure requirements. The user interface and the usage of the tool is also described. The requirements described in the paper cover such aspects as teamwork, project planning and project management using user stories and tasks. The process of the tool's development is also described. The requirements for the tool stem from the authors' experiences with XP and from the experiences they had using the development versions of the tool. Plans for the future development are also presented. The plans include support for other agile methods and integration with other tools used in software development.

Angioni et al.[8] have written another XPSwiki-related article, which presents XPSuite tool suite consisting of XPSwiki and XP4IDE tools. XP4IDE integrates the process data contained in XPSwiki to an integrated development environment (IDE). In addition to enabling access to the data from the IDE, the XP4IDE tool allows automatic recording of the effort spent on code artifacts and tasks based on how long editor windows are open.

Kääriäinen et al.[9] report a failed attempt to improve requirements management in Extreme Programming with a computerized StoryManager tool. The report concentrates on describing the research project around the tool and does not describe the exact features of the tool. The case reported is also relatively small, covering only 6 releases over 8.4 weeks with total of 19 user stories and 92 tasks. Another paper by Kääriäinen et al.[10] describes StoryManager more extensively. The tool is implemented as a plug-in to the Eclipse IDE. The tool is used to manage both stories and tasks, which in the tool form a hierarchy where each story can have many tasks attached to it. Each story and task has an equal set of mandatory attributes. In addition, stories and tasks can have optional attributes that are defined at project level and can differ between tasks and stories.

---

[6] Sandro Pinna, Paolo Lorrai, Michele Marchesi, and Nicola Serra. Developing a tool supporting XP process. In Proceedings of the 2003 Extreme Programming and Agile Methods Conference, pages 151-160, 2003.

[7] Sandro Pinna, Simone Mauri, Paolo Lorrai, Michele Marchesi, and Nicola Serra. XPSwiki: An agile tool supporting the planning game. In Proceedings of the 2003 Extreme Programming and Agile Processes in Software Engineering Conference, pages 1014-1014, 2003.

[8] M. Angioni, D. Carboni, S. Pinna, R. Sanna, N. Serra, and A. Soro. Integrating XP project management in development environments. Journal of Systems Architecture, 52(11): 619-626, 2006.

[9] J. Kääriäinen, J. Koskela, P. Abrahamsson, and J. Takalo. Improving requirements management in Extreme Programming with tool support - an improvement attempt that failed. In Proceedings of the 30th Euromicro Conference, pages 342-351, 2004.

[10] J. Kääriäinen, J. Koskela, J. Takalo, P. Abrahamsson, and K. Kolehmainen. Supporting requirements engineering in Extreme Programming: Managing user stories. In Proceedings of the 16th International Conference on Software & Systems Engineering and their Applications, 2003.

Appendix B

Interview introduction and questions

This appendix presents introduction for the interviewees for the interview and the interview questions used. The introduction in Section B.1 was given to each of the interviewees before the interview was started. The next two sections present the outlines of the interview question structures used in the interviews for the requirements of a tool for managing agile software development. As the interviews were carried out in Finnish, the interview questions are also in Finnish. Section B.2 contains the outline of questions for software developers. Section B.3 contains the outline of questions for managers.

B.1 Introduction for the subjects in Finnish

Seuraavat asiat kerrotaan jokaiselle haastateltavalle ennen varsinaisen haastattelun alkua:

- Haastattelun tarkoitus on tutkia mitä vaatimuksia teidän yrityksenne toimintatavat ohjelmistokehittäjän päivittäisen työn hallintaan ja seurantaan tarkoitetulle ohjelmalle tai työkalulle. Tarkoitus ei ole arvostella sinun työtapojasi, mitata sinun työtehokkuuttasi tai millään muullakaan tavalla kohdistua erityisesti sinun henkilökohtasiin ominaisuuksiisi. Kysymyksissä voi olla hieman toistoa johtuen siitä, että samaa asiaa saatetaan lähestyä eri näkökannalta. Tarkoitus ei ole testata, että muistatko mitä vastasit aikaisemmin.

- Haastattelusta syntynyttä tallennetta käsitellään luottamuksellisesti eikä niitä tulla luovuttamaan kenellekään tutkimusryhmämme ulkopuoliselle. Tallenne tuhotaan sen jälkeen, kun sen säilyttäminen tukimuksen kannalta ei ole enää tarpeellista. Haastattelunaikaiset muistiinpanot tullaan käsittelemään myös luottamuksellisesti. Jos haastattelunaikaiset muistiinpanot julkaistaan, ne käsitellään siten, ettei haastateltavan henkilöllisyy selviä niistä. Haastattelusta syntyvästä julkaistavasta analyysimateriaalista ei myöskään tule selviämään haastateltavien henkilöllisyys.

- Syystä riippumatta voit jättää vastaamatta mihin tahansa haastatelukysymykseen. Sinun ei tarvitse kertoa, miksi et halua vastata kysymykseen. Haastattelu voidaan keskeyttää milloin tahansa, jos niin haluat.

B.2 Questions for developers

1 Kerro eilisestä työpäivästäsi.
1.1 Oliko se mielestäsi tyypillinen työpäivä?
1.1.1 Ei
1.1.1.1 Milloin sinulla mielestäsi oli viimeksi tyypillinen työpäivä?
1.1.1.1.1 Kerro siitä päivästä.
2 Mitä työtehtäviisi kuuluu?
2.1 Onko olemassa jokin ryhmä, johon sinä kuulut?
2.1.1 Mikä tämän ryhmän tehtävä on?
2.1.2 Onko olemassa työtehtäviä, jotka kuuluvat ryhmälle kokonaisuudessaan?
2.1.2.1 Kyllä
2.1.2.1.1 Mihin ryhmälle kuuluvat työt kirjataan ylös?
2.1.2.1.1.1 Liittyykö työn kirjaamiseen virallinen prosessi?
2.1.2.1.1.1.1 Kyllä
2.1.2.1.1.1.1.1 Mitä askeleita prosessissa on?
2.1.2.1.1.1.1.1.1 Mitä informaatio prosessin k.o. askeleessa käytetään?
2.1.2.1.1.1.1.1.2 Ketä osallistuu prosessin k.o. askeleen suorittamiseen?
2.1.2.1.1.1.2 Ei
2.1.2.1.1.1.2.1 Pitäisikö siihen liittyä mielestäsi jokin prosessi?
2.1.2.1.1.1.2.1.1 Kyllä
2.1.2.1.1.1.2.1.1.1 Miksi?
2.1.2.1.1.1.2.1.1.2 Millainen prosessi?
2.1.2.1.1.1.2.1.2 Ei
2.1.2.1.1.1.2.1.2.1 Miksi?
2.1.2.1.1.1.2.2 Kuka työt kirjaa?
2.1.2.1.1.2 Mitä informaatiota työstä kirjataan ylös?
2.1.2.1.1.3 Muuttuuko kirjattu informaatio?
2.1.2.1.1.3.1 Kyllä
2.1.2.1.1.3.1.1 Mistä muutokset johtuvat?
2.1.2.1.1.3.1.2 Kuka ne kirjaa?
2.1.2.1.1.3.1.2.1 Miksi?
2.1.2.1.1.3.2 Ei
2.1.2.1.1.3.2.1 Pitäisikö sen muuttua?
2.1.2.1.1.3.2.1.1 Miksi?
2.1.2.1.1.4 Onko tämä ainut tapa, jolla tehtävää työtä kirjataan?
2.1.2.1.1.4.1 Ei
2.1.2.1.1.4.1.1 Miten muuten tehtävää työtä kirjataan?
2.1.2.1.1.4.1.1.1 Liittyykö työn kirjaamiseen virallinen prosessi?
2.1.2.1.1.4.1.1.1.1 Kyllä
2.1.2.1.1.4.1.1.1.1.1 Kuvaile prosessia?
2.1.2.1.1.4.1.1.1.2 Ei
2.1.2.1.1.4.1.1.1.2.1 Pitäisikö siihen liittyä mielestäsi jokin prosessi?
2.1.2.1.1.4.1.1.1.2.1.1 Kyllä
2.1.2.1.1.4.1.1.1.2.1.1.1 Miksi?
2.1.2.1.1.4.1.1.1.2.1.1.2 Millainen prosessi?
2.1.2.1.1.4.1.1.1.2.1.2 Ei
2.1.2.1.1.4.1.1.1.2.1.2.1 Miksi?
2.1.2.1.1.4.1.1.2 Kuka työt kirjaa?
2.1.2.1.1.4.1.1.3 Mitä informaatiota työstä kirjataan ylös?
2.1.2.1.1.4.1.1.4 Muuttuuko kirjattu informaatio?
2.1.2.1.1.4.1.1.4.1 Kyllä
2.1.2.1.1.4.1.1.4.1.1 Mistä muutokset johtuvat?
2.1.2.1.1.4.1.1.4.1.2 Kuka ne kirjaa?
2.1.2.1.1.4.1.1.4.1.2.1 Miksi?
2.1.2.1.1.4.1.1.4.2 Ei
2.1.2.1.1.4.1.1.4.2.1 Pitäisikö sen muuttua?
2.1.2.1.1.4.1.1.4.2.1.1 Miksi?
2.1.2.2 Ei
2.1.2.2.1 Pitäisikö sellaisia olla?
2.1.2.2.1.1 Miksi?
3 Kun tulit työpaikalle, millä perusteella valitsit mitä rupeat tekemään?
3.1 Käytitkö tietokoneohjelmaa, josta ilmeni mitä sinun seuraavaksi kannattaisi tehdä?
3.1.1 Kyllä
3.1.1.1 Miten työkalusta ilmenee se, mitä sinun seuraavaksi kannattaisi tehdä?
3.1.1.2 Voitko itse vaikuttaa siihen, mikä työkalun mukaan on seuraava tehtävä joka sinun kannattaisi tehdä?

3.1.1.2.1 Ei
3.1.1.2.1.1 Haluaisitko kuitenkin vaikuttaa siihen?
3.1.1.2.1.1.1 Miksi?
3.1.1.3 Teitkö niinkuin työkalu ehdotti?
3.1.1.3.1 Kyllä
3.1.1.3.1.1 Oliko työkalun tekemä valinta mielestäsi oikea?
3.1.1.3.1.1.1 Miksi?
3.1.1.3.2 Ei
3.1.1.3.2.1 Miksi?
3.1.2 Ei
3.1.2.1 Olisiko sinulla kuitenkin mahdollisuus käyttää jotain työkalua, joka kertoisi sinulle mitä sinun seuraavaksi kannattaisi tehdä?
3.1.2.1.1 Kyllä
3.1.2.1.1.1 Miksi et käytä k.o. työkalua?
3.1.2.1.2 Ei
3.1.2.1.2.1 Olisiko mielestäsi sellaiselle työkalulle tarvetta?
3.1.2.1.2.1.1 Miksi?
4 Onko sinulla tällä hetkellä sopiva määrä työtehtäviä?
4.1 Kyllä
4.1.1 Miksi sinulla on sopiva määrä työtehtäviä?
4.2 Ei
4.2.1 Miksi sinulle ei ole sopivaa määrää työtehtäviä?
4.3 Onko se normaali tilanne?
4.3.1 Kyllä
4.3.2 Ei
4.3.2.1 Mistä tilanne johtuu?
4.3.2.2 Mikä on normaali tilanne?
4.4 Kuka valitsee sinulle kuuluvat työtehtävät?
4.5 Millä perusteella sinulle asetetut työtehtävät on valittu?
4.6 Milloin sinulle kuuluvat työtehtävät on valittu?
5 Millaisella aikajänteellä suunnittelet tulevaa työtäsi?
5.1 Mitä informaatiota tämän suunnitelman tekoon käytetään?
5.1.1 Mihin kysetä informaatiota käytetään?
5.2 Onko olemassa pidemmän aikajänteen suunnitelma?
5.2.1 Mitä informaatiota tämän suunnitelman tekoon käytetään?
5.2.1.1 Mihin kysetä informaatiota käytetään?
5.2.2 Onko olemassa vielä pidemmän aikajänteen suunnitelma?
5.2.2.1 Mitä informaatiota tämän suunnitelman tekoon käytetään?
5.2.2.1.1 Mihin kysetä informaatiota käytetään?
5.3 Onko olemassa lyhyemmän aikajänteen suunnitelma?
5.3.1 Mitä informaatiota tämän suunnitelman tekoon käytetään?
5.3.1.1 Mihin kysetä informaatiota käytetään?
5.3.2 Onko olemassa vielä lyhyemmän aikajänteen suunnitelma?
5.3.2.1 Mitä informaatiota tämän suunnitelman tekoon käytetään?
5.3.2.1.1 Mihin kysetä informaatiota käytetään?
6 Mihin sinulle kuuluvat työt kirjataan ylös?
6.1 Liittyykö työn kirjaamiseen virallinen prosessi?
6.1.1 Kyllä
6.1.1.1 Mitä askeleita prosessissa on?
6.1.1.1.1 Mitä informaatio prosessin k.o. askeleessa käytetään?
6.1.1.1.2 Ketä osallistuu prosessin k.o. askeleen suorittamiseen?
6.1.2 Ei
6.1.2.1 Pitäisikö siihen liittyä mielestäsi jokin prosessi?
6.1.2.1.1 Kyllä
6.1.2.1.1.1 Miksi?
6.1.2.1.1.2 Millainen prosessi?
6.1.2.1.2 Ei
6.1.2.1.2.1 Miksi?
6.1.2.2 Kuka työt kirjaa?
6.2 Mitä informaatiota työstä kirjataan ylös?
6.3 Muuttuuko kirjattu informaatio?
6.3.1 Kyllä
6.3.1.1 Mistä muutokset johtuvat?
6.3.1.2 Kuka ne kirjaa?
6.3.1.2.1 Miksi?
6.3.2 Ei
6.3.2.1 Pitäisikö sen muuttua?

6.3.2.1.1 Miksi?
6.4 Onko tämä ainut tapa, jolla tehtävää työtä kirjataan?
6.4.1 Ei
6.4.1.1 Miten muuten tehtävää työtä kirjataan?
6.4.1.1.1 Liittyykö työn kirjaamiseen virallinen prosessi?
6.4.1.1.1.1 Kyllä
6.4.1.1.1.1.1 Kuvaile prosessia?
6.4.1.1.1.2 Ei
6.4.1.1.1.2.1 Pitäisikö siihen liittyä mielestäsi jokin prosessi?
6.4.1.1.1.2.1.1 Kyllä
6.4.1.1.1.2.1.1.1 Miksi?
6.4.1.1.1.2.1.1.2 Millainen prosessi?
6.4.1.1.1.2.1.2 Ei
6.4.1.1.1.2.1.2.1 Miksi?
6.4.1.1.2 Kuka työt kirjaa?
6.4.1.1.3 Mitä informaatiota työstä kirjataan ylös?
6.4.1.1.4 Muuttuuko kirjattu informaatio?
6.4.1.1.4.1 Kyllä
6.4.1.1.4.1.1 Mistä muutokset johtuvat?
6.4.1.1.4.1.2 Kuka ne kirjaa?
6.4.1.1.4.1.2.1 Miksi?
6.4.1.1.4.2 Ei
6.4.1.1.4.2.1 Pitäisikö sen muuttua?
6.4.1.1.4.2.1.1 Miksi?
7 Seurataanko työsi edistymistä?
7.1 Kyllä
7.1.1 Miten työsi edistymistä seurataan?
7.1.1.1 Onko seurantaan olemassa virallinen prosessi?
7.1.1.1.1 Kyllä
7.1.1.1.1.1 Mikä tämä prosessi on?
7.1.1.1.1.1.1 Onko tämä prosessi tarpeellinen?
7.1.1.1.1.1.1.1 Miksi?
7.1.1.1.2 Ei
7.1.1.1.2.1 Pitäisikö siihen olla prosessi?
7.1.1.1.2.1.1 Miksi?
7.2 EI
7.2.1 Pitäisikö sitä seurata?
7.2.1.1 Kyllä
7.2.1.1.1 Miksi?
7.2.1.1.2 Miten sitä pitäisi seurata?
7.2.1.1.2.1 Miksi?
7.2.1.2 Ei
7.2.1.2.1 Miksi
8 Onko työtehtävillesi asetettu aikarajoja?
8.1 Kyllä
8.1.1 Millaisella tarkkuudella aikaraja asetetaan?
8.1.2 Kuka aikarajan asettaa?
8.1.3 Millä perusteella aikaraja asetetaan?
8.1.4 Ylittyykö aikaraja koskaan?
8.1.4.1 Kyllä
8.1.4.1.1 Mitä siitä seuraa?
8.1.4.2 Ei
8.1.4.2.1 Miksi?
8.2 Ei
8.2.1 Pitäisikö aikaraja olla?
8.2.1.1 Miksi?
8.3 Miten tärkeitä akarajat mieletäsi ovat?
8.3.1 Miksi?
9 Pidätkö kirjaa työtunneistasi?
9.1 Kyllä
9.1.1 Millä tarkkuuella työtunnit kirjataan?
9.1.2 Kohdennetaanko työtunnit jotenkin?
9.1.2.1 Kyllä
9.1.2.1.1 Mihin työtunnit kohdennetaan?
9.1.2.1.2 Millä tarkkuudella työtunnit kohdennetaan?
9.1.2.2 Ei

9.1.2.2.1 Pitäisikö ne sinun mielestäsi kohdentaa jotenkin?
9.1.2.2.1.1 Kyllä
9.1.2.2.1.1.1 Miten?
9.1.2.2.1.1.1.1 Miksi?
9.1.2.2.1.2 Ei
9.1.2.2.1.2.1 Miksi?
9.1.3 Miten tärkeänä pidät työtuntien kirjaamista?
9.1.3.1 Miksi?
9.2 Ei
9.2.1 Pitäisikö työtunneista mielestäsi pitää jotenkin kirjaa?
9.2.1.1 Ei
9.2.1.1.1 Miksi?
9.2.1.2 Kyllä
9.2.1.2.1 Miten?
9.2.1.2.1.1 Miksi?
10 Keskeytyykö työsi koskaan sen takia, että saat jonkin tärkeämmän uuden tehtävän?
10.1 Kyllä
10.1.1 Millä tavalla tämä uusi tehtävä kommunikoidaan sinulle?
10.1.1.1 Jos uusia tehtäviä kerrotaan suullisesti, miten muistat ne?
10.1.2 Millä perusteella tiedät, että uusi tehtävä on tärkeämpi kuin kesken jättämäsi työ?
10.2 Ei
10.2.1 Osaatko kertoa miksi näin on?
11 Kerroit käyttäväsi työkalua X. Onko työkalussa mielestäsi puutteita?
11.1 Kyllä
11.1.1 Mitä puutteita?
12 Käytätkö työsi hallintaan tai ohjaamiseen vielä jotain työkalua, joka ei tullut aikaisemmin ilmi?
12.1 Kyllä
12.1.1 Mitä työkalua?
12.1.1.1 Mihin sitä käytetään?
12.1.1.1.1 Onko työkalu hyvä siihen mihin te sitä käytätte?
12.1.1.2 Puuttuuko siitä joitain ominaisuuksia?
12.1.1.2.1 Kyllä
12.1.1.2.1.1 Mitä?
12.2 Ei
12.2.1 Pitäisikö jokin työkalu vielä olla?
12.2.1.1 Kyllä
12.2.1.1.1 Millainen työkalu?
12.2.1.1.1.1 Miksi?
12.2.1.2 Ei
12.2.1.2.1 Miksi?
13 Toimiiko se, miten työtehtäviä jaetaan ja seurataan teillä hyvin?
13.1 MItä hyvää siinä on?
13.2 MItä huonoa siinä on?
13.3 Onko sinulla parannnusehdotuksia?

B.3 Questions for managers

1 Mitä työtehtäviisi kuuluu?
1.1 Kuulutko johonkin tiettyyn ryhmään tai ryhmiin?
1.1.1 Mikä tämän ryhmän tehtävä on?
1.1.2 Mikä on roolisi tässä ryhmässä?
2 Mistä ohjelmistolle tulevat vaatimukset ovat lähtöisin?
2.1 Mihin nämä vaatimukset kirjataan?
2.1.1 Miksi?
2.2 Missä muodossa nämä vaatimukset kirjataan?
2.2.1 Kuka ne kirjaa?
2.2.2 Mitä informaatiota vaatimukset sisältävät?
2.2.2.1 Miksi kukin informaatio kirjataan?
2.3 Mitä vaatimuksille tämän jälkeen tapahtuu?
2.3.1 Onko tätä varten määritelty tietty prosessi?
2.3.1.1 Mitä askeleita tässä prosessissa on?
2.3.1.1.1 Kuka suorittaa kunkin askeleen?
2.3.1.1.1.1 Miksi?
3 Millä perusteella seuraava kehitettävä vaatimus valitaan?

3.1 Kuka valinnan tekee?
3.1.1 Miksi?
3.2 Otetaanko muut olemassaolevat vaatimukset huomioon vaatimusta valittaessa?
3.2.1 Kyllä
3.2.1.1 Onko vaatimusten välillä tietty tärkeysjärjestys?
3.2.1.1.1 Kyllä
3.2.1.1.1.1 Kuka tärkeysjärjestyksen asettaa?
3.2.1.1.1.1.1 Miksi?
3.2.1.1.2 Ei
3.2.1.1.2.1 Pitäisikö niillä olla tärkeysjärjestys?
3.2.1.1.2.1.1 Miksi?
3.2.1.2 Otetaanko vaatimusten väliset riippuvuudet jotenkin huomioon?
3.2.1.2.1 Kyllä
3.2.1.2.1.1 Miten?
3.2.1.2.2 Ei
3.2.1.2.2.1 Miksi?
3.2.2 Ei
3.2.2.1 Pitäisikö ne ottaa huomioon?
3.2.2.1.1 Miksi?
4 Miten yksittäinen kehittäjä saa tietää, mitä hänen pitää tehdä?
4.1 Missä muodossa nämä työtehtävät kerrotaan?
4.1.1 Kuka tämän tekee?
4.1.1.1 Miksi?
4.1.2 Mitä informaatiota nämä työtehtävät sisältävät?
4.1.2.1 Miksi juuri tämä informaatio?
4.1.2.2 Missä tätä informaatiota säilytetään?
4.1.3 Missä näitä työtehtäviä säilytetään?
4.2 Käytetäänkö tähän jotain tiettyä työkalua?
4.2.1 Kyllä
4.2.1.1 Mitä työkalua?
4.2.1.2 Mitä informaatiota se kertoo?
4.2.2 Ei
4.2.2.1 Tarvitsisitko työkalua?
4.2.2.1.1 Miksi?
5 Miten seuraat työn edistymistä?
5.1 Millä tasoilla seuraat työn edistymistä?
5.1.1 Miten seuraat yksittäisen kehittäjän työn edistymistä?
5.1.2 Miten seuraat ryhmän työn edistymistä?
5.1.3 Miten seuraat yksittäisen vaatimuksen kehityksen edistymistä?
5.1.4 Miten seuraat työn edistymistä kokonaisuudessaan?
5.2 Käytetäänkö tähän jotain tiettyä työkalua?
5.2.1 Kyllä
5.2.1.1 Mitä työkalua?
5.2.1.2 Mitä informaatiota se kertoo?
5.2.2 Ei
5.2.2.1 Tarvitsisitko työkalua?
5.2.2.1.1 Miksi?
5.3 Mistä havaitset, jos työ ei etene ajallaan?
5.4 Miten reagoit, jos työ ei etene ajallaan?
5.4.1 Miten kommunikoit kehittäjille tilanteen?
5.4.2 Miten tilanne korjataan?
5.4.2.1 Kuka osallistuu tähän?
5.4.2.1.1 Miksi?
5.5 Seurataanko ajankäyttöä jotenkin?
5.5.1 Kyllä
5.5.1.1 Miten sitä seurataan?
5.5.1.2 Miksi sitä seurataan?
5.5.2 Ei
5.5.2.1 Pitäisikö sitä seurata?
5.5.2.1.1 Miksi
6 Käytätkö joitain työkaluja, jotka eivät vielä tulleet esille?
6.1 Mitä?
6.1.1 Mihin tarkoitukseen käytät k.o. työkalua?
7 Millainen mielestäsi olisi täydellinen työn ohjaukseen ja hallintaan käytettävä työkalu?
8 Toimiiko työn ohjaus ja hallinta mielestäsi?
8.1 Mitä huonoa siinä on?

8.2 Mitä hyvää siinä on?
8.3 Kehitysehdotuksia?

Appendix C

List of the identified tools

This appendix presents all tools identified in the search of agile software development management tools. The tools are divided into two tables. The first table (Table C.1) contains open source tools. The second table (Table C.2) contains commercial closed-source tools. The tools are presented in alphabetical order. Tools that did not fit any exclusion criteria are **boldfaced** in the list.

The first column contains the name of the tool, the second column contains the author. The subsequent columns contain the exclusion criteria of the tools, which are described in the list bellow. The numbers on the list correspond to the numbers in the column headers. Note that the evaluation of the tools was done in a "lazy" way; After one of exclusion criterion matched with a tool, no further exclusion criteria were evaluated. Matching exclusion criterion is marked with "X" in the cell.

Identifying an individual author especially in open source projects cannot always be done. In such cases one of the contributors of the project was selected and augmented with "et al." Some authors don't reveal their reals name but use aliases instead. In such cases the alias is used as the author's name.

1. The tool is not in a mature development stage.
2. The development of the tool is not alive (no more than 1 active developer, no new release in year 2008).
3. The tool is not primarily directed toward managing daily work of agile software development.
4. The tool is not available for evaluation.

Table C.1: Identified open-source tools

Tool	Author	1. Not mature	2. Not alive	3. Not agile	4. Not available
Agile!	Arnaud Prost	X			
Agilefant	TKK / SoberIT				
AgilePlanner	Frank Maurer et al.				
AgileTrack	x10gimli		X		
Agilo for Scrum	agile42	X			
BagLock	Eckhart Köppen		X		
CollabScrum	Reuben J. Ravago	X			
Digital User Stories	Pl4gue	X			
EclipseXP	Peter Hagg		X		
eXtreme Management Tool	Zest Software				
FireScrum	Diogo Verissimo et al.	X			
FreeProjectTracker	Stefan Roock		X		
Gamma Development Tracker	Adam Blinkinsop	X			
GoSprint	Gulsharan Goraya et al.	X			
IceScrum	Stéphane Maldini et al.	X			
jscrum	Pavel Konnikov et al.	X			
LESA	Brian O'Neill	X			
Neutrino	Bil Simser et al.	X			
NXPlanner	Stephen Starkey	X			
nPlanner	Dave Sanders et al.	X			
OpenEPM	Ben Floyd et al.	X			
OpenERP	Tiny company			X	
Open workbench	Openworkbench.org			X	
PHPScrum	EACOMM Corporation	X			
PrjPlanner	Quentin Crain	X			
Project Dune	Gerard Toonstra et al.				
Project Planning and Tracking System	Erik Bos et al.		X		
Really Simple Story Queue	Don Kelly	X			
richPlanner	j2ee711	X			
Rugby	Pieter	X			

Table C.1: Identified open-source tools continued

Tool	Author	1. Not mature	2. Not alive	3. Not agile	4. Not available
SCManager	Mark French	X			
Scrinch	Julien Piaser et al.	X			
SCRUM	Francesco Mondora et al.	X			
Scrumaster	Agustin J. Lopez	X			
ScrumWare	Andrew Romanenco	X			
scrum-rabbit	Angelo J. R. Pereira et al	X			
Scrum Project Scheduler	Mikel Alcón	X			
Scrum Vision	Laurent Carbonnaux	X			
Scrum Zen	pmartins_		X		
Story Server	Mathias Kolehmainen	X			
TaskJuggler	Chris Schläger			X	
TWiki XP Tracker Plugin	Peter Thoeny	X	X		
User Story.NET	Jason Pettys	X			
UserStory-FeatureTracker	Jason Don Stracner	X			
XPCGI	Joi Ellis	X			
XPlanner	Jacques Morel et al.		X		
XPstorm	Juergen Ebert		X		
XPSwiki	Sandro Pinna et al.		X[1]		
XPTracker	Andy Korth et al.		X		
XPWeb	Olivier Chirouze et al.		X		
XP4IDE	Manuela Angioni et al.	X			
XP Roadmap	Arlo Belshee	X			
XP Studio	Remon Sinnema	X			

---

[1] The XPSwiki project web page did not respond to requests, and all other information found regarding XPSwiki was outdated

Table C.2: Identified commercial closed-source tools

Tool	Author	1. Not mature	2. Not alive	3. Not agile	4. Not available
Agileplan	Agilecor			X	
Accept	Accept Software				X
CA Clarity	CA			X	
ClariZen	Clarizen			X	
DevPlanner	Fedorenko				
DevSuite	TechExcel			X	
Electric Cloud	Electric Cloud			X	
Focal Point	Telelogic			X	
Mingle	ThoughtWorks				
OnTime	Axosoft				
Product Pathfinder	Nihito Technologies			X	
Project	Microsoft			X	
Qpack	Orcanos			X	
Rally	Rally Software Development				
ReqDB	Requirements Management			X	
Tracker	Dotner			X	
Scope Manager	Select		X		
ScrumDesk	ScrumDesk				
ScrumWorks Pro	Danube Technologies				
StoryManager	StoryManager				X
VersionOne	VersionOne				
VisionProject	Visionera				
Visual Studio Team System 2008 Team Suite	Microsoft			X	
XP Plan-it	IT Works Solutions		X		